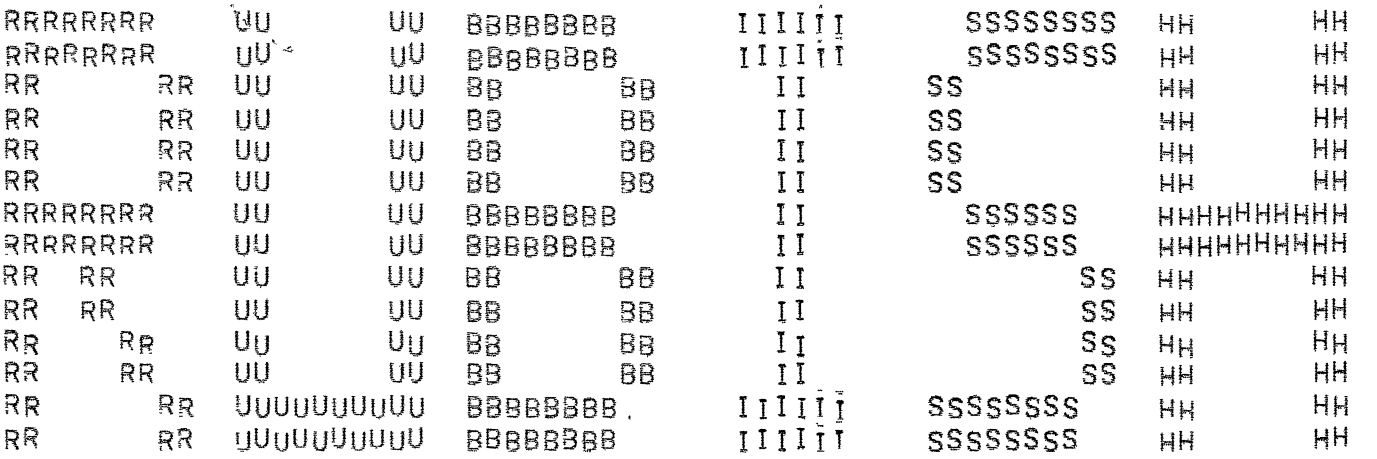
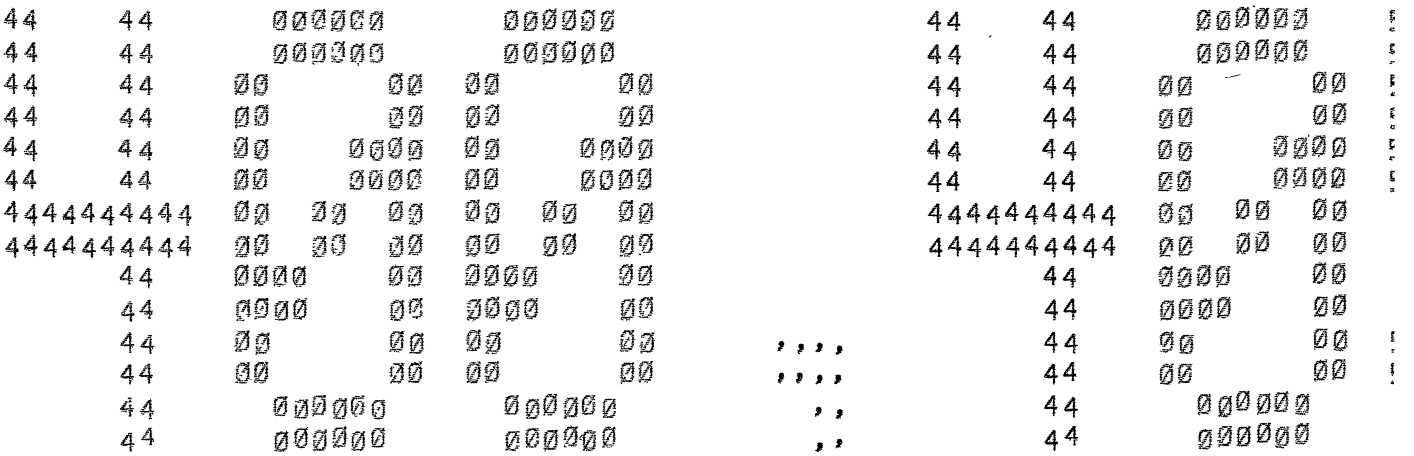
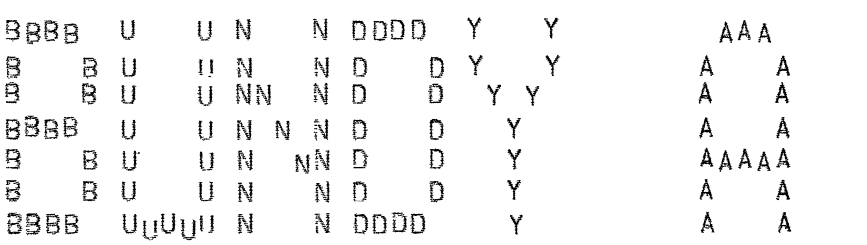






LPTSPL VERSION 6(344) RUNNING ON LPT500
START USER BUNDY A [400,405] JOB INFER SEQ. 11603 DATE 01-MAR-78 15:
REQUEST CREATED: 01-MAR-78 15:22:59
FILE: DSKA0:RUBISH[400,405] CREATED: 01-MAR-78 15:20:00 <155> PRINTED:
QUEUE SWITCHES: /PRINT:ARROW /FILE:ASCII /COPIES:1 /SPACING:1 /LIMIT:67
FILE WILL BE RENAMED TO <055> PROTECTION

```
INA(2240,LBS,TONS).

INA(60,MINS,HRS).

INA(60,SECS,MINS).

INA(1760,YDS,MLS).

INA(3,FT,YDS).

INA(12,INS,FT).

FACTOR(1,←U,←U) :- ISUNIT(←U), !.

FACTOR(←U,←U,←U) :- ISCONST(←U), !.  /*FOR SQUARES ETC*/

FACTOR(←C,←U1,←U) :- INA(←C,←U1,←U), !.

FACTOR(←C,←U1,←U) :- COMP(←U1),
     ←U1 =..←←.←ARGS1 ,  ←U =..←S.←ARGS ,
    SFACTOR(←CS,←ARGS1,←ARGS),  ←C =..←S.←CS , !.

FACTOR(1/←C,←U1,←U) :- BEFORE(←U,←U1), FACTOR(←C,←U,←U1), !.

FACTOR(←C1.←C2,←U2,←U) :- INA(←C1,←U1,←U),
    FACTOR(←C2,←U2,←U1), !.

SFACTOR([],[],[]) :- !.

SFACTOR(←C.←CS,←U1.←US1,←U.←US) :- FACTOR(←C,←U1,←U),
    SFACTOR(←CS,←US1,←US).

BEFORE(←U,←U1) :- DIMENSIONS(←U,←D), POSN(←D,←U,←P),
    POSN(←D,←U1,←P1), ←P<←P1, !.

POSN(M,LBS,1).

POSN(M,TONS,2).

POSN(T,SECS,1).

POSN(T,MINS,2).

POSN(T,HRS,3).

POSN(L,INS,1).

POSN(L,FT,2).

POSN(L,YDS,3).

POSN(L,MLS,4).

ISUNIT(←U) :- POSN(←D,←U,←N).
```

```
/*IS L A GROUND CALL?*/
GROUNDTEST(←L,GROUND) :- ←L=..←PROP.←ARGS,
  CHECKLIST(BOUND,←ARGS), !.

GROUNDTEST(←L,GEN).

/*IS L A PURE RELATION*/
FUNCTEST(←L,PRED) :- ←L=..←PROP.←ARGS,
  ISRELN(←PROP), !.

/*IS L A FUNCTION CALL?*/
FUNCTEST(←L,FUNC) :- ←L=..←PROP.←ARGS,
  SEPERATE(←PROP,←ARGS,←PARGS,←FARGS),
  CHECKLIST(BOUND,←PARGS), !.

FUNCTEST(←L,PRED).

/*IS CALL SILLY?*/
SILLY(←L) :- ←L=.. ←PROP.←ARGS, SEPERATE(←PROP,←ARGS,←PARGS,←FARGS),
  SEPERATE(←PROP,←NARGS,←PARGS,←NFARGS),
  ←NL =.. ←PROP.←NARGS, ?(←NL), DIFF(←FARGS,←NFARGS).

/*SEPERATE FUNCTION AND PREDICATE ARGS*/
SEPERATE(←PROP,←ARGS,←PARGS,←FARGS) :-
  ARGSTRUC(←PROP,←ARGROLES),
  SORTOUT(←ARGROLES,←ARGS,←PARGS,←FARGS).

SORTOUT([],[],[],[]).
SORTOUT(TIME.←ARS,←A.←AS,←A.←PAS,←FAS) :-
  SORTOUT(←ARS,←AS,←PAS,←FAS), !.
SORTOUT(OBJ.←ARS,←A.←AS,←A.←PAS,←FAS) :-
  SORTOUT(←ARS,←AS,←PAS,←FAS), !.
SORTOUT(VAL.←ARS,←A.←AS,←PAS,←A.←FAS) :-
  SORTOUT(←ARS,←AS,←PAS,←FAS), !.
SORTOUT(ANG.←ARS,←A.←AS,←PAS,←A.←FAS) :-
  SORTOUT(←ARS,←AS,←PAS,←FAS), !.
SORTOUT(QUAN.←ARS,←A.←AS,←PAS,←A.←FAS) :-
  SORTOUT(←ARS,←AS,←PAS,←FAS), !.

/*IS PROP A PURE RELATION NAME*/
ISRELN(←PROP) :-
  MEMBER(←PROP,[FIXED←CONTACT,FIXED,CONTACT,CONSTACCEL,CONSTVEL,
    POINT,PATH,MOMENT,PERIOD,SOLID,FORCE,FREE,PROBTYPE]).

/*INFORMATION ABOUT QUANTITY FUNCTIONS*/
/*-----------------------------------*/


/*STRUCTURE OF ARGUMENTS*/
ARGSTRUC(DURATION,[TIME,QUAN]).

ARGSTRUC(←PROP,[OBJ,ANG]) :-
  MEMBER(←PROP,[NORMAL,TANGENT]).

ARGSTRUC(←PROP,[OBJ,QUAN]) :-
  MEMBER(←PROP,[CONSTLENGTH,GROUND,RADIUS,COEFF]).

ARGSTRUC(←PROP,[OBJ,QUAN,TIME]) :-
```

```
  MEMBER(←PROP,[MASS,TENSION,DISTANCE,VARLENGTH]).

ARGSTRUC(←PROP,[OBJ,OBJ,←QUAN]) :-
  MEMBER(←PROP,[DROP,TYPICAL←DROP]).

ARGSTRUC(←PROP,[OBJ,ANG,OBJ]) :-
   MEMBER(←PROP,[ANGLE,INCLINE]).

ARGSTRUC(←PROP,[OBJ,QUAN,ANG,TIME]) :-
  MEMBER(←PROP,[ACCEL,VEL]).

ARGSTRUC(←PROP,[OBJ,OBJ,QUAN,ANG,TIME]) :-
  MEMBER(←PROP,[RELACCEL,RELVEL,REACTION]).

/*ARGUMENT STRUCTURE FOR NON QUANTITIES */

ARGSTRUC(←PROP,[TIME,VAL]) :-
    MEMBER(←PROP,[INITIAL,FINAL]).
ARGSTRUC(TYPICAL←POINT,[OBJ,VAL]).
ARGSTRUC(AT,[OBJ,VAL,TIME]).
ARGSTRUC(←PROP,[OBJ,VAL,OBJ]) :-
    MEMBER(←PROP,[END,FAREND]).
ARGSTRUC(FAREND,[OBJ,OBJ,VAL]).

/*TYPE AND DIMENSION INFORMATION ABOUT FUNCTIONS*/

TYPEINFO(←PROP,DIMLESS,1) :-
  MEMBER(←PROP,[ANGLE,INCLINE,NORMAL,TANGENT,COEFF]), !.

TYPEINFO(MASS,MASS,M) :- !.

TYPEINFO(←PROP,LENGTH,L) :-
  MEMBER(←PROP,[CONSTLENGTH,VARLENGTH,
    DISTANCE,DROP,TYPICAL←DROP,GROUND,RADIUS]), !.

TYPEINFO(DURATION,DURATION,T).

TYPEINFO(←PROP,VEL,L/T) :-
  MEMBER(←PROP,[VEL,RELVEL]), !.

TYPEINFO(←PROP,ACCEL,L/(T:2)) :-
  MEMBER(←PROP,[ACCEL,RELACCEL]), !.

TYPEINFO(←PROP,FORCE,M*(L/(T:2))) :-
  MEMBER(←PROP,[FORCE,TENSION,REACTION]), !.
```

```
:- 'LC'.
:- op(700,xfx,[in,into,from]).

/* FULL DATABASE UTILITIES
   ======================

Functions provided:

   X into D  -  Puts assertion X into database D

   X in D    -  Tests for assertion matching X in D  (X cannot be variable)

   X from D  -  Tests for assertion matching X in D  (X can be variable)

(Use 'in' to look for assertions for a given predicate and 'from' to
generate all assertions in a database)      */

/* These functions are versions of those used by Dave Warren in his
   WARPLAN P program. More efficient versions can be obtained if
   they are compiled since some bits of 'pseudo prolog' can be
   made use of     */

X into DB :- X=..[F,..A], name(F,[L,..._]), I is L /\ 7,
   item(1,DB,(Set,List)), sinto(X,Set), include(A,List,X).

include([],_,_).
include([Y,..A],[(S1,DB),..List],X) :-
   (var(Y), !, sinto(X,S1); name(Y,[L,..._]), I is L /\ 7,
   item(I,DB,S2), sinto(X,S2)), include(A,List,X).

X in DB :- X=..[F,..A], name(F,[L,..._]), I is L /\ 7,
   item(1,DB,(Set,List)), nonvar(Set),
   ((match(A,List,S),sin(X,S));sin(X,Set)), !.

match([X,..A],[_,..List],S) :- var(X), !, match(A,List,S).
match(_,[(S,_),..._],S).
match([X,..._],[(_,DB),..._],S) :- name(X,[L,..._]), I is L /\ 7,
   item(1,DB,S).

item(0,£(X,_,_,_,_,_,_,_),X).
item(1,£(_,X,_,_,_,_,_,_),X).
item(2,£(_,_,X,_,_,_,_,_),X).
item(3,£(_,_,_,X,_,_,_,_),X).
item(4,£(_,_,_,_,X,_,_,_),X).
item(5,£(_,_,_,_,_,X,_,_),X).
item(6,£(_,_,_,_,_,_,X,_),X).
item(7,£(_,_,_,_,_,_,_,X),X).

sinto(X,S) :- var(S), !, S=set(_,X,_).
sinto(X,set(S1,_,S2)) :- (abrandom,!,sinto(X,S1); sinto(X,S2)).

nonvar(X) :- var(X), !, fail.
nonvar(_).
```

```
sin(X,S) :- var(S), !, fail.
sin(X,set(_,X,_)).
sin(X,set(S,_,_)) :- sin(X,S).
sin(X,set(_,_,S)) :- sin(X,S).

:- record(r(0)).

atrandom :- recorded(r(N),P), erase(P), N1 is N+1, record(r(N1)),
   !, 0 is N mod 2.

X from DB :- item(I,DB,(Set,_)), sin(X,Set).

:- end.
```

```
/*FUNCTION CALL - PREDICATE CALL*/
/*ALAN BUNDY 30/12/76*/


:- OP(825,XFX,":="),     /*VALUE OF FUNCTIONS*/


/*DECIDE TYPE OF CALL*/
←Q := ←FUN     :-
  ←FUN =.. ←PROP.←ARGS,
  SLOTIN(←Q,←ARGS,←NARGS),
  ←NFUN =.. ←PROP.←NARGS,
  COND(CHECKLIST(NONVAR,←NARGS),UC(←PROPS,←ARGS,←Q),←TRUE),
  COND(CHECKLIST(NONVAR,←ARGS),FC(←NFUN),←NFUN).

/*FUNCTION CALL*/
FC(←NFUN) :- ←NFUN, !.

/*SLOT VALUE INTO ARG LIST*/
SLOTIN(←Q,[],[←Q]).
SLOTIN(←Q,[←OBJ],[←OBJ,←Q]).
SLOTIN(←Q,[←OBJ,←TIME],[←OBJ,←Q,←TIME]).
SLOTIN(←Q,[←OBJ,←DIR,←TIME],[←OBJ,←Q,←DIR,←TIME]).
SLOTIN(←Q,[←OBJ1,←OBJ2,←DIR,←TIME],[←OBJ1,←OBJ2,←Q,←DIR,←TIME]).


/*UNIQUENESS CHECK FOR CONFIRMATION CALLS*/
UC(←PROP,←ARGS,←Q) :-
  SORTIN(←X,←ARGS,←OARGS),
  ←OFUN =..←PROP.←OARGS,
  DBC(←OFUN),
  DIFF(←X,←Q), !,FAIL.

UC(←PROP,←ARGS,←Q).
```

```
/*RECOVER DESCRIPTION OF QUANTITY*/
FINDDESC(_Q,_QTYPE,_OBJ,_ANG,_TIME) :-
  QUANTITY(_Q,_DIM,_VS,_NUM,_PROP),
  DIMOF(_QTYPE,_DIM),
  SEPERATE(_ARGS,_PARGS,_FARGS,_NUM),
  _DESC=..[_PROP|_ARGS], _DESC,
  SPLIT1(_PARGS,_OBJ,_TIME), SPLIT2(_FARGS,_ANG).

/*SPLIT PREDICATE ARGS INTO BITS*/
SPLIT1([],_OBJ,_TIME).
SPLIT1([_OBJ],_OBJ,_TIME).
SPLIT1([_OBJ,_TIME],_OBJ,_TIME).
SPLIT1([_OBJ1,_OBJ2,_TIME],_OBJ1,_TIME).
SPLIT1([_OBJ1,_OBJ2,_TIME],_OBJ2,_TIME).

/*SPLIT FUNCTION ARGS TO GET ANGLE*/
SPLIT2([_Q],_ANG).
SPLIT2([_Q,_ANG],_ANG).


QUANTITY(_Q,1,S,2,COEFF) :- KIND(_Q,COEFF), !.

QUANTITY(_Q,T,S,2,DURATION) :- KIND(_Q,DURATION), !.

QUANTITY(_Q,L,S,3,LENGTH) :- KIND(_Q,LENGTH), !.

QUANTITY(_Q,M,S,3,MASS) :- KIND(_Q,MASS), !.

QUANTITY(_Q,M*(L/(T^2)),V,4,FORCE) :- KIND(_Q,FORCE), !.

QUANTITY(_Q,M*(L/(T^2)),S,3,TENSION) :- KIND(_Q,TENSION), !.

QUANTITY(_Q,M*(L/(T^2)),V,5,REACTION) :- KIND(_Q,REACTION), !.

QUANTITY(_Q,M*(L/T(T^2)),V,5,FRICTION) :- KIND(_Q,FRICTION), !.

QUANTITY(_Q,L/T,V,4,VEL) :- KIND(_Q,VEL), !.

QUANTITY(_Q,L/T^2,V,4,ACCEL) :- KIND(_Q,ACCEL), !.

QUANTITY(_Q,L/T^2,V,5,RELACCEL) :- KIND(_Q,RELACCEL), !.

QUANTITY(_Q,L/T,V,5,RELVEL) :- KIND(_Q,RELVEL), !.


DIMOF(DIMLESS,1).

DIMOF(MASS,M).

DIMOF(LENGTH,L).

DIMOF(DURATION,T).

DIMOF(VEL,L/T).

DIMOF(ACCEL,L/(T^2)).

DIMOF(FORCE,M*(L/(T^2))).
```

```
ISVECT(_Q) :- QUANTITY(_Q,_D,V,_NUM,_PROP).

ISSCAL(_Q) :- QUANTITY(_Q,_D,S,_NUM,_PROP).



SEPERATE([_Q],[],[_Q],1).
SEPERATE([_OBJ,_Q],[_OBJ],[_Q],2).
SEPERATE([_OBJ,_Q,_TIME],[_OBJ,_TIME],[_Q],3).
SEPERATE([_OBJ,_Q,_DIR,_TIME],[_OBJ,_TIME],[_Q,_DIR],4).
SEPERATE([_OBJ1,_OBJ2,_Q,_DIR,_TIME],[_OBJ1,_OBJ2,_TIME],[_Q,_DIR],5).



MESS(_PROP,[_Q]) :-
    MAKE(_PROP,_Q),
    TRACE(LET-_Q-BE-A-_PROP,2).

MESS(_PROP,[_OBJ,_Q]) :-
    MAKE(_PROP,_Q),
    TRACE(LET-_Q-BE-THE-_PROP-OF-_OBJ,2).

MESS(_PROP,[_OBJ,_Q,_TIME]) :-
    MAKE(_PROP,_Q),
    TRACE(LET-_Q-BE-THE-_PROP-OF-_OBJ-IN-_TIME,2).

MESS(_PROP,[_OBJ,_Q,_DIR,_TIME]) :-
    MAKE(_PROP,_Q), MAKE(DIRECTION,_DIR),
    TRACE(LET-_Q-BE-THE-_PROP-OF-_OBJ,2),
    TRACE(IN-DIRECTION-_DIR-AND-TIME-_TIME,2).

MESS(_PROP,[_OBJ1,_OBJ2,_Q,_DIR,_TIME]) :-
    MAKE(_PROP,_Q), MAKE(DIRECTION,_DIR),
    TRACE(LET-_Q-BE-THE-_PROP-OF-_OBJ1-RELATIVE-TO-_OBJ2,2),
    TRACE(IN-DIRECTION-_DIR-AND-TIME-_TIME,2).
```

```
/* FILE  xtract.

                contains        :  BRICK  set equations,  (@1)
                                   BRICK  make equations.  (@2)

*/

/*=========================================================================*/

/* BRICK  set equations.     (@1)

                entry points    :  seteqns(+,+,+,-,-).

                internals       :  chooseeqn(5),finddesc(4),relates(2),
                                   relates(2),prepare(4),indep(3),useless(3),
                                   grind(3),setobj(2).

                uses            :  BRICK  io routines,
                                   BRICK  list routines,
                                   BRICK  set routines,
                                   BRICK  evaluation routines,
                                   BRICK  multilist routines,
                                   BRICK  argument list routines,
                                   BRICK  types,
                                   BRICK  make equations.

                db active       :  NONE.

                db passive      :  NONE.

                variables       :  NONE.

*/



  seteqns([],Gs,Us,true,[]).

  seteqns([X|Xs],Gs,Us,( E & Es ),[X|Xs1])
        :- ccflag := off,
           trace('\nI am now trying to solve for %t without introducing any unkn
           chooseeqn(X,E,U,Us),
           ( trace('\n Equation : %t\n of type : %t formed.\n',[E,U],3) ;
             trace('\n Equation rejected.\n\n',3), fail
           ),
           wordsin(E,Ws),
           look(seteqns1,[X,Xs,Gs,Ws],[X]),
           memberchk(X,Ws),
           subset(Ws,[X|Gs]),
           trace('\n This equation solves for %t.\n',[X],3),
           !,
           seteqns(Xs,[X|Gs],[U|Us],Es,Xs1).

  seteqns([X|Xs],Gs,Us,(E & Es),[X|Xs1])
        :- ccflag := on,
           trace('\nNo luck - I will now accept unknowns in solving for %t.\n',[
           chooseeqn(X,E,U,Us),
           ( trace('\n Equation : %t\n of type : %t formed.\n',[E,U],3) ;
             trace('\n Equation rejected.\n',3), fail
           ),
           wordsin(E,Ws),
           look(seteqns2,[X,Xs,Gs,Ws],[X]),
           memberchk(X,Ws),
           union([X|Xs],Gs,Ys),
```

```
        subtract(Ws,Ys,Zs),
        append(Xs,Zs,Nxs),
        trace('\n This equation solves for %t but introduces %t.\n',[X,Zs],3)
        seteqns(Nxs,[X|Gs],[U|Us],Es,Xs1).

seteqns([X|Xs],Gs,Us,Es,Xs1)
      :- trace('\nI am unable to solve for %t.\n',[X],3),
        look(seteqns3,[X],[X]),
        fail.



chooseeqn(Q,Eqn,[Eqname|Eqinfo],Used)
      :- finddesc(Q,Qtype,Types,Args),
        look(chooseeqn1,[Q,Qtype,Types,Args],[Q]),
        relates(Qtype,Eqlist),
        member(Eqname,Eqlist),
        prepare(Eqname,_,Args,Eqinfo),
        look(chooseeqn2,[Eqname,Eqinfo],[Eqname]),
        index(Eqname,Eqinfo,Used),
        makeeqn(Eqn,Eqname,Eqinfo).



finddesc(Q,Qtype,Types,TP)
      :- kind(Q,Qtype,Pred,Pos),
        argstruct(Pred,N,Types,_),
        numlist(1,N,Nlist),
        mlmaplist(insert(Q,Pos),[Nlist,Varlist,TP]),
        L =.. [Pred|TP],
        dc L.


insert(Q,Pos,[Pos,_,Q]) :- !.

insert(Q,Pos,[_,Var,Var]).



relates(force,[resolve,moments]).
relates(mass,[resolve,moments]).
relates(angle,[resolve]).
relates(vel,[constvel,relvel,constaccel-N,consvenersy-N]).
relates(accel,[resolve,constaccel-N,relaccel]).
relates(duration,[timesum,constvel,constaccel-1,constaccel-2]).
relates(length,[moments,lengthsum,constvel,constaccel-2,
                        constaccel-3,consvenersy-N]).
relates(moment,[moments]).



prepare(resolve,t6,[Line,_,Time],[[Obj,Ans],Time])
      :- generalise(Line,Gline),
        point_of(Gline,P),
        not_sameplace(P,Obj,Time),
        type(particle,Obj),
        point_of(Line2,P),
        minimal(Line2),
        incline(Line2,Ans,P).


prepare(resolve,t8,[O1,O2,Val,Ans,Time],[[Obj,Ans],Time])
      :- type(particle,O1), Obj = O1  ;
        type(particle,O2), Obj = O2.
```

```
prepare(relvel,t8,[Obj1,Obj2,Val,Ans,Time],[[Obj1,Obj2,Obj3],Time])
        :- type(point_of_ref,Obj3),
           diff(Obj3,Obj1),
           diff(Obj3,Obj2).


prepare(relaccel,t8,[Obj1,Obj2,Val,Ans,Time],[[Obj1,Obj2,Obj3],Time])
        :- type(point_of_ref,Obj3),
           diff(Obj3,Obj1),
           diff(Obj3,Obj2).


prepare(moments,_,L,A)
        :- writef('Prepare moments :\nL = %t\nA = ',[L]),
           ttyflush,
           read(A).



index(Eqname,Eqinfo,Used)
        :- not member([Eqname,Eqinfo],Used),
           not useless(Eqname,Eqinfo,Used),
           !.


useless(resolve,[[Part,Dir],Period],Used)
        :- select([resolve,[Part,X],Period],Used,Rest),
           member([resolve,[Part,Y],Period],Rest).

useless(relvel,[Objs,Time],Used)
        :- member([relvel,X,Time],Used),
           seteq(Objs,X).

useless(relaccel,[Objs,Time],Used)
        :- member([relaccel,X,Time],Used),
           seteq(Objs,X).

useless(constaccel-N,Eqinfo,Used)
        :- select([constaccel-X|Eqinfo],Used,Rest),
           member([constaccel-Y|Eqinfo],Rest).




generalise(Line,Gline)
        :-  /*  ??constraint??  */
           dc partition(P,Plist),
           type(line,P),
           member(Line,Plist),
           !,
           generalise(P,Gline).

generalise(Line,Line).

/* END OF BRICK  set equations. */


/*=================================================================*/


/* BRICK  make equations.      (@2)

                entry points     :  makeeqn(?,+,+).
```

```
                internals       :  isform(3).

                uses            :  BRICK  search control,
                                   FILE  infer.

                db active       :  oldeqn(3).

                db passive      :  partition(2).

                variables       :  NONE.

*/


  makeeqn(E,N,S)   :- oldeqn(E,N,S), !.

  makeeqn(E,N,S)
        :- isform(E,N,S),
           cassertz(oldeqn(E,N,S)),
           look(makeeqn,[E,N,S],[N]),
           !.



  isform(F=M*A,resolve,[[Part,Dir],Period])
        :- cc mass(Part,M,Period),
           accel_cmpnt(Part,A,Dir,Period),
           sumforces(Part,Dir,Period,F).


  isform(Msum=zero,moments,[Point,Time])
        :- point_of(Rod,Point),
           type(rod,Rod),
           ncc incline(Rod,Dir,Point),
           summoments(Point,Rod,Msum,Dir,Time).



  isform(V13=V123,relvel,[[P1,P2,P3],Time])
        :- cc relvel(P1,P2,V12,Dir12,Time),
           cc relvel(P2,P3,V23,Dir23,Time),
           cc relvel(P1,P3,V13,Dir13,Time),
           vecadd(V12,Dir12,V23,Dir23,V123,Dir13).


  isform(A13=A123,relaccel,[[P1,P2,P3],Time])
        :- cc relaccel(P1,P2,A12,Dir12,Time),
           cc relaccel(P2,P3,A23,Dir23,Time),
           cc relaccel(P1,P3,A13,Dir13,Time),
           vecadd(A12,Dir12,A23,Dir23,A123,Dir13).


  isform(Av=D/T,avervel,[Obj,Period])
        :- cc distance(Obj,D,Period),
           cc duration(Period,T).


  isform(S=V*T,constvel,[Obj,Period])
        :- ncc constvel(Obj,Period),
           cc vel(Obj,V,Dir,Period),
           cc duration(Period,T),
           cc distance(Obj,S,Period).
```

```
   isform(V=U+(A*T),constaccel-1,[Obj,Period])
         :- ncc constaccel(Obj,Period),
            cc accel(Obj,A,Dir,Period),
            cc duration(Period,T),
            initvel(Obj,U,Dir,Period),
            finvel(Obj,V,Dir,Period).


   isform(S=(U*T)+(A*(T:2)/2),constaccel-2,[Obj,Period])
         :- ncc constaccel(Obj,Period),
            cc accel(Obj,A,Dir,Period),
            cc duration(Period,T),
            initvel(Obj,U,Dir,Period),
            cc distance(Obj,S,Period).


   isform((V:2)=(U:2)+2*A*S,constaccel-3,[Obj,Period])
         :- ncc constaccel(Obj,Period),
            cc accel(Obj,A,Dir,Period),
            cc distance(Obj,S,Period),
            initvel(Obj,U,Dir,Period),
            finvel(Obj,V,Dir,Period).


   isform(g*H=((V:2)/2)-((U:2)/2),consvenersy-1,[Obj,Period])
         :- dc cued(motion(Obj,Path,Start,Side,Period)),
            ncc free(Path,Obj,Period),
            ncc drop(Path,Start,H),
            finvel(Obj,V,Dir1,Period),
            initvel(Obj,U,Dir2,Period).


   isform(g*H=((V:2)/2)-((U:2)/2),consvenersy-2,[Obj,Period])
         :- dc cued(motion(Obj,Path,Start,Side,Period)),
            ncc free(Path,Obj,Period),
            ncc typical_drop(Path,Start,H),
            cc vel(Obj,V,Dir1,Period),
            initvel(Obj,U,Dir2,Period).


   isform(T=Sum,timesum,[_,Period])
         :- dc partition(Period,Ps),
            cc duration(Period,T),
            sumdurs(Ps,Sum).


   isform(S=Sum,lensthsum,[Path,Time])
         :- dc partition(Path,Pts),
            cc varlensth(Path,S,Time),
            sumlensth(Pts,Sum,Time).



/* END OF BRICK  make equations. */


/*=============================================================================*/

   :-end.
```

```
/*  FILE  logic.

                contains          :  BRICK  top level,  (@1)
                                     BRICK  schemata,  (@2)
                                     BRICK  search control,  (@3)
                                     BRICK  normal form,  (@4)
                                     BRICK  sameclass,  (@5)
                                     BRICK  types,  (@6)
                                     BRICK  argument list routines.  (@7)
*/

/*=============================================================================*/

/*  BRICK  top level.     (@1)

                entry points      :  go,
                                     qa(+,+).

                internals         :  setup(3),solve(4)crunch(3),
                                     goughts(1),givens(1).

                uses              :  BRICK  set equations,
                                     BRICK  io routines,
                                     BRICK  set routines,
                                     BRICK  invocation routines,
                                     BRICK  application routines,
                                     BRICK  evaluation routines,
                                     proviso(1),subst(3).

                db active         :  gought(1),given(1).

                db passive        :  NONE.
-------------------------------------------------------------------------------
                variables         :  NONE.

*/


  go :- goughts(Xs),
        givens(Gs),
        solve(Xs,Gs,Es,Xs1),
        crunch(Es,Xs1,Ans),
        trace('Answer is %t\n',[Ans],2).


  qa(Qual,Quan)
        :- Qual,
           findall(X,proviso(X),Condlist),
           trace('%t ok provided :%l',[Qual,Condlist],2),
           setup(Condlist,Xs,Gs),
           solve(Xs,Gs,Es,Xs1),
           crunch(Es,Xs1,Ans),
           subst(Solns,Condlist,Ncondlist),
           trace('New conditions are :%l',[Ncondlist],2),
           apply(Quan,[Ncondlist]).



  setup(Condlist,Xs,Gs)
        :- wordsin(Condlist,Vars),
           givens(Gs),
           subtract(Vars,Gs,Xs).
```

```
  solve(Xs,Gs,Es,Xs1)
        :- trace('\nAttemptins to solve for %t in terms of %t\n\n',[Xs,Gs],2),
           seteans(Xs,Gs,[],Es,Xs1),
           trace('\n\n\7Equations extracted : %c\n',[Es],2).


  crunch(Es,Xs1,Ans)
        :- trace('\nRest of rrosram not loaded\n',2),
           abort.


  soushts(Slist)
        :- findall(X,sousht(X),Slist).

  sivens(Glist)
        :- findall(X,siven(X),Glist).


/* END OF BRICK  top level. */

/*=============================================================================*/


/*  BRICK  schemata.     (@2)

                entry roints     :  cue(+).

                internals        :  schema(4).

                uses             :  BRICK  io routines,
                                    BRICK  application routines,
                                    BRICK  normal form,
                                    BRICK  search control.

                db active        :  PROGRAM DEPENDENT.

                db rassive       :  NONE.

                variables        :  NONE.

*/


  cue(Key)
        :- schema(Key,Decl,Ass,Def),
           trace('Pullins in schema %t\n',[Key],3),
           Key =.. [Type|Arss],
           trace('\nDeclarations  (%t)\n',[Type],4),checklist(call,Decl),
           trace('\nAssertions  (%t)\n',[Type],4),checklist(dbentry,Ass),
           trace('\nDefaults  (%t)\n',[Type],4),checklist(defs,Def),
           trace('\n\n',4),
           dbentry(cued(Key)),
           trace('\n%r\n\n',[-,50],4),
           !.


  defs(X)
        :- trace(' %t\n',[X],4),
           assertz(X).


  schema(rullsys_stan(Sys,Pull,Str,P1,P2,Time),
                [ cue(rullsys_maJ(Sys,Pull,Str,P1,90,P2,270,Time)) ],
```

```
                ],
                [] ).


schema(pullsys_maj(Sys,Pull,Str,P1,Dir1,P2,Dir2,Time),
                [ cue(pullsys_min(Sys,Pull,Str,Dir1,Dir2,Time)),
                  ncc end(Str,Lend,left),
                  ncc end(Str,Rend,risht) ],

                [ constaccel(P1,Time),
                  constaccel(P2,Time),
                  fixed_contact(P1,Lend,Time),
                  fixed_contact(P2,Rend,Time),
                  (relaccel(P1,Pull,A1,Dir1,Time)
                        :- ncc elastic(Str,zero),
                           ncc relaccel(P2,Pull,A1,Dir2,Time) ),
                  (relaccel(P2,Pull,A2,Dir2,Time)
                        :- ncc elastic(Str,zero),
                           ncc relaccel(P1,Pull,A2,Dir1,Time) ) ],

                [] ).


schema(pullsys_min(Sys,Pull,Str,Dir1,Dir2,Time),
                [ cue(strinssys(Str,Lbit,Pulpt,Rbit,Time)),
                  cc isa(particle,Pull)   ],

                [ probtype(pulley,Time),
                  fixed_contact(Pull,Pulpt,Time),
                  tansent(Lbit,Dir1),
                  tansent(Rbit,Dir2),
                  (tension(Lbit,T1,Time)
                        :- ncc elastic(Str,zero),
                           ncc coeff(Pull,zero),
                           ncc tension(Rbit,T1,Time) ),
                  (tension(Rbit,T2,Time)
                        :- ncc elastic(Str,zero),
                           ncc coeff(Pull,zero),
                           ncc tension(Lbit,T2,Time) ),
                  (tension(Str,T3,Time)
                        :- ncc elastic(Str,zero),
                           ncc coeff(Pull,zero),
                           ncc tension(Lbit,T3,Time) ) ],

                [ coeff(Pull,zero),
                  mass(Pull,zero,Time) ] ).


schema(strinssys(Str,Lbit,Pulpt,Rbit,Time),
                [ cue(linesys(Str,Lend,Rend)),
                  cue(linesys(Lbit,Lend,Pulpt)),
                  cue(linesys(Rbit,Pulpt,Rend)),
                  cc isa(strins,Str),
                  cc isa(strins,Lbit),
                  cc isa(strins,Rbit)   ],

                [ partition(Str,[Lbit,Rbit]),
                  concavity(Lbit,stline),
                  concavity(Rbit,stline)  ],

                [ elastic(Str,zero),
                  mass(Str,zero,Time) ]  ).


schema(pathsys(Name,Lend,Rend,Slope,Conv),
                [ cue(linesys(Name,Lend,Rend)),
```

```
                cc isa(rod,Name)    ],

              [ concavity(Name,Conv),
                slope(Name,Slope)     ],

              [ coeff(Name,zero) ]  ).


schema(rodsys(Rod,Time),
              [ cue(linesys(Rod,End1,End2)),
                cc incline(Rod,Dir,End1),
                cc separation(End1,End2,D,Dir,Time),
                cc c_of_gravity(Rod,Cofs)     ],

              [ concavity(Rod,stline),
                separation(End1,Cofs,D/2,Dir,Time)     ],

              []  ).



schema(line_motion(Part,Path,Time),
              [ cc end(Path,Start,left),
                cue(motion(Part,Path,Start,left,Time)) ],

              [ probtype(motion-in-a-straight-line,Time),
                concavity(Path,stline),
                slope(Path,hor)  ],

              []  ).


schema(motion(Part,Path,Start,Side,Time),
              [ cc farend(Path,Finish,Start),
                cc final(Time,End),
                cc incline(Path,Ans1,Finish),
                condturn3(Path,Start,Ans1,Ans2),
                cc vel(Part,V,Ans2,End),
                cc typical_point(Path,Point),
                cc initial(Time,Begin)  ],

              [ at(Part,Finish,End),
                fixed_contact(Part,Finish,End),
                fixed_contact(Part,Start,Begin),
                fixed_contact(Part,Point,Time)  ],

              [ (probtype(motion-in-a-straight-line)
                     :- ncc constvel(Part,Time)      \\
                        ncc constaccel(Part,Time)  ),
                (V > zero
                     :- top(Path,Start),
                        ncc probtype(roller-coaster,Time)  )
                                                          ]  ).


schema(linesys(Line,Lend,Rend),
              [ csensym(line,Line),
                cc end(Line,Lend,left),
                cc end(Line,Rend,right),
                cc isa(point,Lend),
                cc isa(point,Rend)   ],

              [],

              []  ).
```

```
  schema(timesys(Per,Mom1,Mom2),
                [ ccreate(period,Per),
                  cc initial(Per,Mom1),
                  cc final(Per,Mom2),
                  cc isa(moment,Mom1),
                  cc isa(moment,Mom2)   ],

                [],

                []  ).


/* END OF BRICK  schemata. */

/*=============================================================================*/

/* BRICK  search control.     (@3)

                entry points     :  cc(+),
                                    ncc(+),
                                    pc(+),
                                    ccreate(+,?),
                                    declare(+).

                internals        :  goal(2),prove(5),postmortem(6)
                                    higher_truth(4),functest(2),groundtest(2),
                                    silly(1),makprvals(3),prargs(2).

                uses             :  BRICK  normal form,
                                    BRICK  argument list routines,
                                    BRICK  multilist routines.

                db active        :  trap(1).

                db passive       :  NONE.

                variables        :  ccflag.

*/


  dc P :- goal(P,db,nocreate).

  ncc P :- goal(P,Both,nocreate).

  cc P :- goal(P,Both,create).

  pc P :- goal(P,Both,nocreate).



  goal(P,Acc,Cn)
        :- not subgoal(exact,P),
           access(P,L,Acct),
           method(Acct,Acc,L,Cn).



  method(edb,_,[Lacc,Lup],nocreate) :- Lacc.

  method(edb,_,[Lacc,Lup],create)
```

```
method(dbinf,Acc,L,Cn)
      :- sroundtest(L,Gs),
         functest(L,Fp),
         prove(Acc,L,Gs,Fp,Result),
         postmortem(L,Gs,Fp,Cn,Result).


prove(db,L,Gs,func,succ) :- ?(L), !.

prove(db,L,sround,pred,succ) :- ?(L), !.

prove(db,L,sen,pred,succ) :- ?(L).


prove(inf,L,sround,func,fail) :- silly(L), !.


prove(inf,L,Gs,func,succ) :- L, !.

prove(inf,L,sround,pred,succ) :- L, !.

prove(inf,L,sen,pred,succ) :- L.

prove(Acc,L,Gs,pred,succ) :- hisher_truth(Acc,L,Gs,pred).

prove(_,_,_,_,fail).



postmortem(P,Gs,Fp,Cn,succ) :- !.

postmortem(P,Gs,func,create,fail) :- deref(ccflags,on), declare(P), !.

postmortem(P,sround,pred,create,fail) :- dbentry(P), !.



hisher_truth(Acc,L,Gs,Fp)
      :- L =.. [Pred|Arss],
         isok(Pred),
         arsstruct(Pred,N,Types,_),
         mlmaplist(reptime,[Types,Arss,TP]),
         NL =.. [Pred|TP],
         prove(Acc,NL,Gs,Fp,succ).


reptime([Type,Ars,Supt])
      :- typesatis(Type,time),
         !,
         nonvar(Ars),
         subtime(Ars,Supt).

reptime([_,X,X]).


isok(constvel).
isok(constaccel).
isok(fixed).
isok(contact).
isok(fixed_contact).
isok(at).
isok(free).
isok(srghtuse).
```

```
sroundtest(L,sround)
      :- atomic(L),
         !.

sroundtest(L,sround)
      :- nonvar(L),
         L =.. [Pred|Arss],
         checklist(sroundtest(sround),Arss),
         !.

sroundtest(L,sen).


functest(L,func)
      :- L =.. [Pred|Arss],
         arsstruct(Pred,N,_,Fmap),
         memberchk(val,Fmap),
         mlmaplist(farssbound,[Fmap,Arss]),
         !.

functest(L,pred).

farssbound([ars,X]) :- nonvar(X).
farssbound([val,_]).


silly(L)
      :- L =.. [Pred|Arss],
         arsstruct(Pred,N,_,Fmap),
         mlmaplist(valtovar,[Fmap,Arss,TP]),
         NL =.. [Pred|TP],
         ?(NL),
         diff(Arss,TP).


valtovar([val,Ars,Var]).

valtovar([ars,Ars,Ars]).


/*-------------------------------------------------------------------------*/


ccreate(Type,X)
      :- csensym(Type,X),
         ( isa(T1,X) -> (T1=Type \\ error(create,[X,Type,T1],break))
                      ; asserta(isa(Type,X)),
                        trace(' Let %t be a new %t\n',[X,Type],1),
                        trace(' %t unit-asserted by ccreate.\n',[isa(Type,X)
         ),



declare(L)
      :- L =.. [Pred|Arss],
         arsstruct(Pred,N,Types,Fmap),
         mlmaplist(makprvals(Pred),[Fmap,Types,Arss],Obj),
         mlmaplist(prarss(Obj),[Fmap,Types,Arss]),
         trace('.\n',1),
         dbentry(L),
         !.
```

```
makprvals(Pred,Pars,T,X,Obj)
      :- var(Obj),
         X = Obj,
         !.

makprvals(_,[ars,_,_],_).

makprvals(Pred,[val,T,X],ObJ)
      :- typesatis(T,ansle)
                  -> csensym(ansle,X),
                     trace(' in direction %t',[X],1)
                  ;  csensym(Pred,X),
                     trace(' Let %t be the %t of %t',[X,Pred,ObJ],1).



prargs(Obj,[ars,T,X])
      :- diff(Obj,X),
         ( typesatis(T,parity) -> trace(' on the %t',[X],1) ;
           typesatis(T,point) -> trace(' at %t',[X],1) ;
           typesatis(T,point_of_ref) -> trace(' relative to %t',[X],1) ;
           typesatis(T,path) -> trace(' on the %t',[X],1) ;
           typesatis(T,period) -> trace(' durins %t',[X],1) ;
           typesatis(T,moment) -> trace(' at %t',[X],1)
         ),

prargs(_,_).

/*-------------------------------------------------------------------------*/


sensym(Pref,Q)
      :- var(Q),
         cderef(ss(Pref),1,X),
         N is X+1,
         ss(Pref) := N,
         concat(Pref,X,Q).



csensym(Pref,Q) :- sensym(Pref,Q), !.

csensym(_,_).



/* END OF BRICK  search control. */


/*=========================================================================*/


/* BRICK  normal form.     (@4)

                entry points     :  dbentry(+),
                                    access(+,?,?),
                                    update(+,?,?),
                                    dodse(+,?).

                internals        :  nfrec(2),reckind(1),defn(6),
                                    ifauans(2),merse_sp(3),merse_sep(4).

                uses             :  BRICK  io routines,
                                    BRICK  application routines,
                                    BRICK  multilist  routines,
```

```
                                    BRICK  error handler.
                                    BRICK  miscellaneous routines,
                                    BRICK  search control,
                                    BRICK  sameclass,
                                    BRICK  arsument list routines.

            db active       :  PROGRAM DEPENDENT.

            db passive      :  PROGRAM DEPENDENT.

            variables       :  NONE.

*/



  dbentry(P)
        :- unit(P),
           !,
           update(P,L,Up),
           nfrec(Up,L).

  dbentry(P)
        :- dodse(P) -> error(cover,[P],continue)
                     ;  asserta(P),
                        trace(' %t  asserted\n',[P],4).



  nfrec(edb,L) :- L, !.

  nfrec(edb,L) :- error(nfrec2,[L],continue).

  nfrec(dbinf,L)
        :- ?(L) -> error(nfrec3,[L],continue)
                ;  recorda(L),
                   trace('  %t  entered into data base.\n',[L],4),
                   reckind(L).



  reckind(L) :- functor(L,P,A), nokind(P,A), !.

  reckind(L)
        :- L =.. [Pred|Arss],
           arsstruct(Pred,N,Types,Fmap),
           numlist(1,N,Nlist),
           mlmaplist(ifval(Pred),[Nlist,Fmap,Types,Arss]),
           !.

  reckind(L)
        :- error(reckind,[L],continue).



  ifval(Pred,[N,val,Type,Ars])
        :- not integer(Ars),
           typesatis(Type,quantity),
           ( kind(Ars,Type,Pred,N)  \\
             asst(kind(Ars,Type,Pred,N)),
             trace(' Note: %t (of type %t) was used in a %t definition (%t)\n'
                        [Ars,Type,Pred,N],2)
           ).

  ifval(_,_).
```

```
nokind(roush,1).
nokind(solid,1).
nokind(isa,2).
nokind(kind,4).
nokind(quantity,3).
nokind(cued,1).
nokind(partition,2).
nokind(slope,2).
nokind(concavity,2).
nokind(constaccel,2).
nokind(probtype,2).
nokind(point_of,2).

/*-------------------------------------------------------------------------*/

update(P,L,Type)
      :- defn(P,Type,LL,Context),
         setupd(Type,LL,L),
         !,
         checklist(cc,Context).


setupd(dbinf,X,X).
setupd(edb,[X,Y],Y).



access(P,LL,Type)
      :- defn(P,Type,LL,Context),
         !,
         checklist(ncc,Context).




dodse(P)
      :- defn(P,Type,L,_),
         ( Type = edb  \\  diff(P,L) ),




defn( isa(T,X),edb,
               [ type(T,X), asst(isa(T,X)) ],
               []  ).

defn( initial(Per,Mom),dbinf,
               bndy(Per,Mom,left),
               []  ).

defn( final(Per,Mom),dbinf,
               bndy(Per,Mom,risht),
               []  ).

defn( vel(Obj,V,Dir,Time),dbinf,
               relvel(Obj,earth,V,Dir,Time),
               []  ).

defn( accel(Obj,A,Dir,Time),dbinf,
```

```
                [] ).

defn( initvel(Obj,V,Dir,Per),dbinf,
                relvel(Obj,earth,V,Dir,Start),
                [ initial(Per,Start) ] ).

defn( finvel(Obj,V,Dir,Per),dbinf,
                relvel(Obj,earth,V,Dir,Finish),
                [ final(Per,Finish) ] ).

defn( tangent(Path,Dir),dbinf,
                incline(Path,Dir,X),
                [ typical_point(Path,X) ] ).

defn( normal(Path,Dir),dbinf,
                angle(Path,Dir,X),
                [ typical_point(Path,X) ] ).

defn( fixed_contact(Ob1,Ob2,Per),edb,
                [ fixed_contact(Ob1,Ob2,Per), merge_sp(Ob1,Ob2,Per) ],
                [] ).

defn( sameplace(Ob1,Ob2,T),edb,
                [ ( sameclass(Ob1,Ob2,_,touch(T)) & diff(Ob1,Ob2) ),
                  merge_sp(Ob1,Ob2,T) ],
                [] ).

defn( concurrent(P1,P2,left),edb,
                [ sameclass(P1,P2,_,initses), _ ],
                [] ).

defn( concurrent(P1,P2,right),edb,
                [ sameclass(P1,P2,_,finses), _ ],
                [] ).

defn( factor(C,U1,U2),edb,
                [ sameclass(U1,U2,C,fac), _ ],
                [] ).

defn( separation(P1,P2,R,A,T),edb,
                [ sameclass(P1,P2,[R,A],sep(T)), merge_sep(P1,P2,[R,A],T) ],
                [] ).


defn(X,dbinf,X,[]).


/*-------------------------------------------------------------------------*/

asst(X)
      :- X -> error(asst,[X],continue)
            ; asserta(X),
              trace('  %t  unit-asserted.\n',[X],4).



merge_sp(X,Y,T)
      :- merge(X,Y,_,touch(T)),
         repeat( subtime(Subt,T) & merge(X,Y,_,touch(T)) ),
         trace('  %t is in the same place as %t during %t.\n',[X,Y,T],4).



merge_sep(X,Y,Rel,T)
```

```
      ;  merge(X,Y,Rel,sep(T)),
         repeat( subtime(Subt,T) & merge(X,Y,Rel,sep(T)) ),
         trace('  %t and %t are seperated by %t during %t.\n',[X,Y,Rel,T],4).


/* END OF BRICK  normal form */

/*=============================================================================*/

/* BRICK  sameclass.     (@5)

               entry points     :  sameclass(?,?,?,+).

               internals        :  rep(6),toroot(6),totips(6),
                                   treeacc(5),
                                   treeir(3),
                                   treeid(2).

               uses             :  NONE.

               db active        :  ina(3),inaf(3),slink(4).

               db passive       :  touchdd(3),partition(2).

               variables        :  NONE.

*/


  sameclass(X,Y,Rel,Ta)
        :- var(X),
           nonvar(Y),
           !,
           sameclass(Y,X,Rel,Ta).


  sameclass(X,Y,Rel,Ta)
        :- Ta =.. [Tree|Arss],
           treeid(Tree,Id),
           rep(X,R,Rel1,Tree,Id,Arss),
           rep(Y,R,Rel2,Tree,Id,Arss),
           treeir(Tree,sc,[Rel1,Rel2,Rel]).



  rep(Z,R,Re,Tree,Id,Arss)
        :- var(Z) -> totips(Z,R,Re,Tree,Id,Arss)
                  ;  toroot(Z,R,Re,Tree,Id,Arss).



  toroot(Z,R,Re,Tree,Id,Arss)
        :- treeacc(Tree,Z,W,Rel,Arss),
           !,
           toroot(W,R,Re2,Tree,Id,Arss),
           treeir(Tree,tt,[Rel,Re2,Re]).

  toroot(R,R,Id,_,Id,_).



  totips(R,R,Id,_,Id,_).
```

```
totips(Z,R,Re,Tree,Id,Arss)
      :- treeacc(Tree,W,R,Rel,Arss),
         totips(Z,W,Re2,Tree,Id,Arss),
         treeir(Tree,tt,[Rel,Re2,Re]).



merse(X,Y,Rel,P)
      :- P =.. [Tree|Arss],
         treeid(Tree,Id),
         toroot(X,XR,Xrel,Tree,Id,Arss),
         toroot(Y,YR,Yrel,Tree,Id,Arss),
         treeir(Tree,merse,[Xrel,Yrel,Rel,Q4]),
         ( XR = YR \\ treeup(Tree,XR,YR,Q4,Arss) ).



build(X,Y,Rel,P)
      :- P =.. [Tree|Arss],
         treeid(Tree,Id),
         toroot(X,XR,Xrel,Tree,Id,Arss),
         toroot(Y,YR,Yrel,Tree,Id,Arss),
         treeir(Tree,build,[Xrel,Yrel,Rel,Q5]),
         ( XR = Y \\ treeup(Tree,XR,Y,Q5,Arss) ).



/*-------------------------------------------------------------------------*/


treeid(touch,_).

treeacc(touch,X,Y,_,[T]) :- ?(touchdd(X,Y,T)).

treeup(touch,X,Y,_,[T]) :- recorda(touchdd(X,Y,T)).

treeir(touch,_,_).



treeid(initses,_).

treeacc(initses,X,Y,_,[]) :- ?(partition(Y,[X|_])).

treeir(initses,_,_).


treeid(finses,_).

treeacc(finses,X,Y,_,[]) :- ?(partition(Y,Plist)), last(X,Plist).

treeir(finses,_,_).



treeid(fac,1).

treeacc(fac,X,Y,Rel,[]) :- ( ina(Rel,X,Y) \\ nonvar(X) , inaf(Rel,X,Y) ).

treeup(fac,X,Y,Rel,[]) :- asserta(ina(Rel,X,Y)).

treeir(fac,sc,[R1,R2,R1/R2]).
treeir(fac,tt,[R1,R2,R1*R2]).
```

```
  treeir(fac,build,[XR,YR,RR,RR/XR]).


  treeid(sep,[0,Dir]).

  treeacc(sep,X,Y,Rel,[T]) :- slink(X,Y,Rel,T).

  treeup(sep,X,Y,Rel,[T]) :- asserta(slink(X,Y,Rel,T)).

  treeir(sep,sc,[R1,R2,REL]) :- vecadd(REL,R2,R1).
  treeir(sep,tt,[R1,R2,REL]) :- vecadd(R1,R2,REL).
  treeir(sep,merge,[XR,YR,RR,Q4]) :- vecadd(RR,YR,Z), vecadd(Q4,XR,Z).
  treeir(sep,build,[XR,YR,RR,Q5]) :- vecadd(Q5,XR,RR).


/* END OF BRICK  sameclass. */

/*=============================================================================*/


/* BRICK  types.     (@6)

                entry points     :  typev(?,?),
                                    type(+,+),
                                    oftype(?,?),
                                    typesatis(+,?).

                internals        :  typetreeu(2).

                uses             :  BRICK  miscellaneous routines.

                db active        :  NONE.

                db passive       :  NONE.

                variables        :  NONE.

*/


  typev(Type,X) :- var(X) \\ type(Type,X).


  type(Type,X)
        :- nonvar(X),
           !,
           isa(Atype,X),
           ( var(Type) -> typesatis(Atype,Type)
                       ;  typesatis(Atype,Type), !
           ),

  type(Type,X)
        :- typesatis(Atype,Type),
           isa(Atype,X).


  typesatis(T1,T2)
        :- var(T1) -> ts1(T1,T2)
                   ;  ts2(T1,T2).
```

```
  ts1(T,T).

  ts1(T1,T2)
        :- typetreeu(Des,T2),
           ts1(T1,Des).


  ts2(T,T).

  ts2(T1,T2)
        :- typetreeu(T1,Parent),
           ts2(Parent,T2).

/*-------------------------------------------------------------------------*/


  typetreeu(parity,entity).
  typetreeu(item,entity).
  typetreeu(quantity,entity).
  typetreeu(notation,entity).

  typetreeu(object,item).
  typetreeu(time,item).

  typetreeu(line,object).
  typetreeu(point_of_ref,object).

  typetreeu(string,line).
  typetreeu(path,line).
  typetreeu(rod,line).

  typetreeu(pp,point_of_ref).
  typetreeu(place,point_of_ref).

  typetreeu(period,time).
  typetreeu(moment,time).


  typetreeu(constant,quantity).
  typetreeu(scalar,quantity).
  typetreeu(angle,quantity).

  typetreeu(force,scalar).
  typetreeu(vel,scalar).
  typetreeu(accel,scalar).
  typetreeu(mass,scalar).
  typetreeu(length,scalar).
  typetreeu(duration,scalar).
  typetreeu(moment,scalar).


  typetreeu(number,notation).
  typetreeu(unit,notation).
  typetreeu(dim,notation).
  typetreeu(dim_system,notation).


/* END OF BRICK  types. */


/*=========================================================================*/


/* BRICK  argument list routines.     (@7)
```

```
                entry points    :  arsstruct(?,?,?,?).

                internals       :  NONE.

                uses            :  NONE.

                db active       :  NONE.

                db passive      :  NONE.

                variables       :  NONE.

*/


  arsstruct(duration,2,
                [period,duration],
                [ars,val] ).

  arsstruct(normal,2,
                [line,angle],
                [ars,val] ).

  arsstruct(tangent,2,
                [line,angle],
                [ars,val] ).

  arsstruct(angle,3,
                [line,angle,point],
                [ars,val,ars] ).

  arsstruct(incline,3,
                [line,angle,point],
                [ars,val,ars] ).


  arsstruct(distance,3,
                [particle,length,period],
                [ars,val,ars] ).

  arsstruct(separation,5,
                [point_of_ref,point_of_ref,length,angle,time],
                [ars,ars,val,val,ars] ).

  arsstruct(sround,2,
                [line,length],
                [ars,val] ).

  arsstruct(drop,3,
                [path,point,length],
                [ars,ars,val] ).

  arsstruct(typical_drop,3,
                [path,point,length],
                [ars,ars,val] ).

  arsstruct(radius,2,
                [line,length],
                [ars,val] ).

  arsstruct(constlength,2,
                [line,length],
                [ars,val] ).

  arsstruct(varlength,3,
```

```
                [ars,val,ars] ).


arsstruct(mass,3,
                [object,mass,time],
                [ars,val,ars] ).


arsstruct(tension,3,
                [string,force,time],
                [ars,val,ars] ).

arsstruct(reaction,5,
                [object,object,force,angle,time],
                [ars,ars,val,val,ars] ).


arsstruct(vel,4,
                [point_of_ref,vel,angle,time],
                [ars,val,val,ars] ).

arsstruct(relvel,5,
                [point_of_ref,point_of_ref,vel,angle,time],
                [ars,ars,val,val,ars] ).


arsstruct(accel,4,
                [point_of_ref,accel,angle,time],
                [ars,val,val,ars] ).

arsstruct(relaccel,5,
                [point_of_ref,point_of_ref,accel,angle,time],
                [ars,ars,val,val,ars] ).


arsstruct(coeff,2,
                [path,constant],
                [ars,val] ).

arsstruct(elastic,2,
                [string,constant],
                [ars,val] ).



arsstruct(bndy,3,
                [period,moment,parity],
                [ars,val,ars] ).

arsstruct(initial,2,
                [period,moment],
                [ars,val] ).

arsstruct(final,2,
                [period,moment],
                [ars,val] ).

arsstruct(isa,2,
                [type,entity],
                [ars,ars]  ).

arsstruct(typical_point,2,
                [line,point],
                [ars,val] ).
```

```
                [point_of_ref,place,moment],
                [ars,val,ars] ).

  arsstruct(end,3,
                [line,point,parity],
                [ars,val,ars] ).


  arsstruct(farend,3,
                [line,point,point],
                [ars,ars,val] ).

  arsstruct(farend,3,
                [line,point,point],
                [ars,val,ars] ).


  arsstruct(center_of_gravity,2,
                [object,point],
                [ars,val] ).


  arsstruct(quantity,3,
                [quantity,number,unit],
                [ars,ars,ars] ).

  arsstruct(stanunit,3,
                [dim,unit,dim_system],
                [ars,ars,ars] ).


/* END OF BRICK  argument list routines. */


/*=============================================================================*/

  :-end.
```

```
/*INFER*/
/*MOTION INFERENCE RULES*/
/*ALAN BUNDY 29/12/76*/



/*MAKING CONDITIONS*/
/*-----------------*/

/*SEE IF ITS TRUE*/
condition(L) :- L,!.

/*SEE IF ITS FALSE*/
condition(L) :- nott(L), !, fail.

/*OTHERWISE NOTE IT FOR LATER*/
condition(L) :- postulate(proviso(L)),
  trace('Storing proviso : %t.\n',[L],3).


/*NEGATION*/
nott(X>Y) :- Y>=X.
nott(X>=Y) :- Y>X.


/*MOTION ON A PATH*/
/*----------------*/

/*MOTION CHECK*/
motion(Part,Path,Start,Side,Per) :-
  inplace(Part,Path,Start,Side,Begin),
  cue(timesys(Per,Begin,End)),
  trace('Checking motion of %t\n\t\t on %t\n\t\t from %t\n\t\t on %t\n\t\t dur
          [Part,Path,Start,Side,Per],4),
  motion1(Part,Path,Start,Side,Per),
  cue(motion(Part,Path,Start,Side,Per)).


/*GENERAL MOTION ON PATH*/   /*LETS THROUGH TOO MANY POSIBILITIES*/
motion1(Part,Path,Start,Side,Per) :-
  not initial(Per,Begin),
  setstarted(Part,Path,Start,Side,Begin),
  nostopping(Part,Path,Start,Side,Per),
  notakeoff(Part,Path,Start,Side,Per),
  nofalloff(Part,Path,Start,Side,Per).


/*BREAK INTO SUBPATHS*/
motion1(Part,Path,Start,Side,Per) :-
  arrangepath(Path,Start,Npathlist),
  makeperiods(Npathlist,Per,Perlist),
  multimotion(Part,Npathlist,Start,Side,Perlist).

arrangepath(Path,Start,Npathlist) :-
  dc partition(Path,Pathlist),
  dc end(Path,Start,End),
  condrev(End,Pathlist,Npathlist).

makeperiods(Npathlist,Per,Perlist) :-
  maplist(makeone,Npathlist,Perlist),
  asserta(partition(Per,Perlist)),
```

```
  trace( divide at end : at zero/reasons... )

/*DEAL WITH EACH SUBPATH*/
multimotion(Part,[],Start,Side,[]).

multimotion(Part,[Path|Path1],Start,Side,[Per|Per1]) :-
  pc motion(Part,Path,Start,Side,Per),
  farend(Path,Start,Finish),
  multimotion(Part,Path1,Finish,Side,Per1).

/*MAKE ONE PERIOD                    how does declare work??*/
makeone(Path,Per) :- ccreate(period,Per).

getstarted(Part,Path,Start,Side,Begin) :-
  ncc vel(Part,zero,Dir,Begin), !,
  ( horizontal(Path,Start) ->
  nudge(Part,Begin) ; top(Path,Start) ).

/*HEADED IN RIGHT DIRECTION*/
getstarted(Part,Path,Start,Side,Begin) :-
  along(Path,Ang,Start),
  ncc vel(Part,V,Ang,Begin),
  condition(V>zero).


/*DOWNHILL RUN*/
nostopping(Part,Path,Start,Side,Per) :-
  dc end(Path,Start,End), opposite(End,Oend),
  dc slope(Path,Hend), diff(Oend,Hend), !.

/*MAKES IT TO THE TOP*/
nostopping(Part,Path,Start,Side,Per) :-
 ncc finvel(Part,V,Dir,Per), condition(real(V)).

/*BELOW PATH*/
notakeoff(Part,Path,Start,Side,Per) :-
  below(Path,Start,Side), !.

/*SLOPE DOES NOT DROP AWAY*/
notakeoff(Part,Path,Start,Side,Per) :-
  dc concavity(Path,Conc), diff(Conc,right), !.

/*INSUFFICIENT VEL TO TAKE OFF*/
notakeoff(Part,Path,Start,Side,Per) :-
  dc concavity(Path,right),
  ncc normal(Path,Dir),
  cc reaction(Path,Part,N,Dir,Per),
  condition(N>=zero).


/*SUPPORTED*/
nofalloff(Part,Path,Start,Side,Per) :-
  above(Path,Start,Side), !.

/*VERTICAL FALL*/
nofalloff(Part,Path,Start,Side,Per) :-
  dc slope(Path,Start), dc concavity(Path,stline),
  ncc incline(Path,270,Start), !.

/*STICKS ON*/
nofalloff(Part,Path,Start,Side,Per) :-
  dc concavity(Path,right),
  ncc normal(Path,Dir1),
  cc vel(Part,V,Dir2,Per),
  ncc radius(Path,R),
  condition((V^2)*sin(Dir1)>=R*g), !.
```

```
/*FREEFALL*/
nofalloff(Part,Path,Start,Side,Per) :-
  dc concavity(Path,Conc), diff(Conc,right),
  asserta(falls_off(Part,Path,Start,Side,Per)),
  !, fail.

/*PARTICLE IS IN PLACE AT START OF PATH*/
inplace(Part,Path1,Finish,Side,End) :-
  dc cued(motion(Part,Path2,Start,Side,Per)),
  farend(Path2,Start,Finish),
  ncc final(Per,End), !.

inplace(Part,Path,Start,Side2,Time) :-
  ncc at(Part,Start,Time),
  dc side(Part,Start,Side1,Time),
  dc end(Path,Start,End),
  condval(End,Side1,Side2).



/*YOU GET TO YOUR DESTINATION BY TRAVELING THERE*/
at(Part,Place2,Mom2) :-
  farend(Path,Place2,Place1),
  inplace(Part,Path,Place1,Side,Mom1),
  cue(timesys(Per,Mom1,Mom2)),
  pc motion(Part,Path,Place1,Side,Per).

/*DESCRIBING PATHS*/
/*----------------*/

/*PATH WITH MONOTONIC SLOPE*/
monopath(Path) :- dc slope(Path,left).
monopath(Path) :- dc slope(Path,right).
monopath(Path) :- dc slope(Path,hor).

/*POINT1 AND POINT2 ARE OPPOSITE ENDS OF PATH*/
farend(Path,Point1,Point2) :-
  dc end(Path,Point1,End1), opposite(End1,End2),
  dc end(Path,Point2,End2).

/*UPPER SIDE OF PATH*/
above(Path,Start,Side) :-
  monopath(Path), dc end(Path,Start,Side).

/*LOWER SIDE OF PATH*/
below(Path,Start,Side1) :-
  monopath(Path),
  dc end(Path,Start,Side2), opposite(Side1,Side2).

/*START IS THE TOP OF PATH*/
top(Path,Start) :- dc end(Path,Start,End), dc slope(Path,End).

/*USE TYPICAL POINT OF WHOLE CIRCLE*/
typical_point(Path,Point) :-
  bitof(Path,Circle), dc circle(Circle),
  cc typical_point(Circle,Point).



/*PARITY DEALING*/
/*--------------*/


/*PARTICLE IS INSIDE OR OUTSIDE OF CONCAVITY IN PATH*/
inside(Part,Path,Time) :- wh_side(Part,Path,Par,Par,Time).
```

```
outside(Part,Path,Time) :- wh_side(Part,Path,Par1,Par2,Time),
  opposite(Par1,Par2).

wh_side(Part,Path,Ans,Nc,Time) :-
  dc cued(motion(Part,Path,Start,Side,Time)),
  dc end(Path,Start,Par),
  twist(Par,Side,Ans),
  dc concavity(Path,Conv), norm(Conv,Nc).

/*CONDITIONAL ABOUT TURN*/
condturn2(Ph1,Ph2,Ans1,Ans2) :-
  dc concavity(Ph1,Conv1), norm(Conv1,Nc1),
  dc concavity(Ph2,Conv2), norm(Conv2,Nc2),
  condturn1(Nc1,Nc2,Ans1,Ans2).

condturn1(Par,Par,Ans,Ans) :- !.
condturn1(Par1,Par2,Ans1,Ans2) :-
  opposite(Par1,Par2), aboutturn(Ans1,Ans2).

condturn3(Path,Start,Dir2,Dir1) :-
  dc concavity(Path,Conv), norm(Conv,Nconv),
  dc end(Path,Start,Par), condturn1(Par,Nconv,Dir2,Dir1).

/*NORMALIZE PARITIES*/
norm(stline,left) :- !.
norm(Par,Par).

/*CONDITIONAL REVERSE*/
condrev(left,List,List).

condrev(risht,List,Rlist) :- rev(List,Rlist).

/*CONDITIONAL VALUE*/
condval(left,Side,Side).
condval(risht,Side1,Side2) :- opposite(Side1,Side2).

/*PARITY CHANGER*/
opposite(left,risht).
opposite(risht,left).

/*PARITY TWIST*/
twist(Par,Par,risht).
twist(Par1,Par2,left) :- opposite(Par1,Par2).

/*PART-WHOLE RELATIONSHIP*/
/*-----------------------*/

/*PARTS OF PERIOD*/

  subtime(Subp,P)
        :- typev(period,Subp),
           typev(period,P),
           bitof(Subp,P).

  subtime(M,P)
        :- typev(moment,M),
           typev(period,P),
           tipof(Tip,P),
           ncc bndy(Tip,M,Par).

/*PARTS OF A PATH*/

  subpath(Subp,P)
        :- typev(path,Subp),
           typev(path,P),
```

```
  subpath(Pt,P)
        :- typev(point,Pt),
           typev(path,P),
           tipof(Tip,P),
           point_of(Tip,Pt).


  point_of(Line,Point)
        :- dc end(Line,Point,Par) ;
           ncc typical_point(Line,Point) ;
           ncc c_of_gravity(Line,Point) ;
           ?( point_of(Line,Point) ).

/*PART OF TREE*/

  partof(Whole,Whole).

  partof(Part,Whole) :- bitof(Part,Whole).


  bitof(Part,Whole)
        :- dc partition(Whole,Parts),
           member(Part1,Parts),
           partof(Part,Part1).

/*TIP OF TREE*/

  tipof(Tip,Tree)
        :- dc partition(Tree,Subtrees),
           !,
           member(Subtree,Subtrees),
           tipof(Tip,Subtree).

  tipof(Tip,Tip).

/*LINE IS SMALLEST*/
minimal(Line) :- thnot(bitof(Subline,Line)).



/*LENGTH*/
/*------*/


/*DISTANCE TRAVELLED BY OBJ DURING PER IS D*/
distance(Obj,D,Per) :-
  dc cued(motion(Obj,Path,Start,Side,Per)),
  ncc varlength(Path,D,Per).


/*IF CONSTANT LENGTH IS KNOWN THEN IT IS ALWAYS LENGTH*/
varlength(X,D,T) :- ncc constlength(X,D).

/*LENGTHS OF CO-INCIDENT PATHS ARE EQUAL*/
varlength(Path1,D,Time) :-
  dc concavity(Path1,stline),
  farend(Path1,End1,End2),
  ncc sameplace(End1,End3,Time),
  ncc sameplace(End2,End4,Time),
  farend(Path2,End3,End4),
  diff(Path1,Path2),
  dc concavity(Path2,stline),
  ncc varlength(Path2,D,Time).
```

```
drop(Path,Start,-(D*tan(Ans))) :-
    dc concavity(Path,stline),
    ncc incline(Path,Ans,Start), dc sround(Path,D).

drop(Path,Start,Hsum) :-
    dc partition(Path,Pl), sumdrops(Pl,Start,Hsum).


/*VERTICAL DROP OF CIRCLE SEGMENT*/
drop(Path,Start,R*(sin(Dir1)-sin(Dir2))) :-
  partof(Path,Circle), dc circle(Circle),
  farend(Path,Start,Finish),
  ncc angle(Start,Dir1,Circle), ncc angle(Finish,Dir2,Circle),
  ncc radius(Circle,R).

/*TYPICAL DROP WITHIN CIRCLE SEGMENT*/
typical_drop(Path,Start,R*(sin(Dir1)-sin(Dir2))) :-
  partof(Path,Circle), dc circle(Circle),
  cc angle(Start,Dir1,Circle),
  ncc normal(Path,Dir2),
  ncc radius(Circle,R).


sumdrops([],Start,0).

sumdrops([P|Pl],Start,H+Sum) :-
    ncc drop(P,Start,H), !, farend(P,Start,Finish),
    sumdrops(Pl,Finish,Sum).

sumlength([],0,Time).

sumlength([P|Pl],D+Sum,Time) :-
    cc varlength(P,D,Time),
    !, sumlength(Pl,Sum,Time).


/*RADIUS OF CURVATURE OF CIRCLE SEGMENT*/
radius(Path,R) :-
  partof(Path,Circle), dc circle(Circle),
  cc radius(Circle,R).

/*ANGLES*/
/*------*/

/*INCLINATION OF HORIZONTAL LINE*/
incline(Line,0,Point) :-
  dc slope(Line,hor).

/*INCLINATION OF STRAIGHT LINE*/
incline(Line,Ans,Point1) :-
  dc concavity(Line,stline),
  point_of(Line,Point2), diff(Point1,Point2),
  ncc incline(Line,Ans,Point2).

/*CONSECUTIVE PATHS HAVE THE SAME INCLINATION
 (ADD SMOOTHNESS CONDITION) */

incline(Ph2,Ans1,Pt) :-
  type(path,Ph2),
  dc partition(Ph0,Ph1),
  nextto(Ph1,Ph2,Ph1),
  ncc incline(Ph1,Ans2,Pt),
  condturn2(Ph1,Ph2,Ans1,Ans2).

/*FIND ANGLE OF MINIMAL PATH*/
```

```
  bitof(Subpath,Path), ncc angle(Subpath,Ang,Point).

/*ANGLE IS AT RIGHT ANGLES TO INCLINE*/
angle(Path,Ang1,Point) :-
  ncc incline(Path,Ang2,Point), rightturn(Ang2,Ang1).

/*INCLINATION ALONG PATH FROM ONE END*/
along(Path,Ang,End) :-
  dc concavity(Path,Conc), norm(Conc,Nc),
  dc end(Path,End,Lr),
  ncc incline(Path,Inc,End),
  condturn1(Nc,Lr,Inc,Ang).

/*IS PATH HORIZONTAL AT POINT*/
horizontal(Path,Point) :- ncc incline(Path,0,Point).
horizontal(Path,Point) :- ncc incline(Path,180,Point).


/*ALWAYS USE TANGENT OF LARGEST PATH*/
tangent(Path,Dir) :-
  bitof(Path,Superpath), ncc tangent(Superpath,Dir).

/*TANGENT IS INCLINE AT TYPICAL POINT*/
tangent(Path,Dir) :-
  cc typical_point(Path,X),
  cc incline(Path,Dir,X).



/*ALWAYS USE NORMAL OF LARGEST PATH*/
normal(Path,Dir) :-
  bitof(Path,Superpath), ncc normal(Superpath,Dir).

/*NORMAL IS ANGLE AT TYPICAL POINT*/
normal(Path,Dir) :-
  cc typical_point(Path,X),
  cc angle(Path,Dir,X).


/*CONTACT*/
/*-------*/

/*POINTS ARE IN CONTACT IF THEY ARE IN THE SAME PLACE*/
contact(X,Y,T) :- ncc sameplace(X,Y,T).

/*AN OBJECT IS IN CONTACT WITH THE PATH IT MOVES ON*/
contact(X,Y,T) :-
  dc cued(motion(X,Y,Start,Side,T)), ncc solid(Y).

/*FIXED CONTACT CANNOT BE MOMENTARY*/
fixed_contact(X,Y,T) :- thnot(type(period,T)), !, fail.

/*FIXED CONTACT WITH EARTH MEANS FIXED*/
fixed_contact(earth,X,T) :- ncc fixed(X,T).

fixed_contact(X,earth,T) :- ncc fixed(X,T).

/*POINTS ARE IN FIXED CONTACT IF THEY ARE IN*/
/*THE SAME PLACE FOR A PERIOD*/
fixed_contact(X,Y,T) :- type(period,T),
                ncc sameplace(X,Y,T), diff(X,Y).



/*FIXEDNESS CANNOT BE MOMENTARY*/
```

```
/*THE EARTH IS FIXED*/
fixed(earth,Time).

/*OBJECTS IN MOTION ARE NOT FIXED*/
fixed(Obj,Time) :-
  dc cued(motion(Obj,Path,Start,Side,Time)), !, fail.

/*UNSUPPORTED PULLEY IS ASSUMED FIXED*/
fixed(Pull,Time) :-
  dc cued(pullsys_min(Sys,Pull,Str,Dir1,Dir2,Time)),
  forall(ncc(sameplace(Pull,Point,Time)) & diff(Point,Pull),
  dc partition(Str,[Lbit,Rbit]) & dc end(Lbit,Point,right)).

/*PATHS IN THE ROLLER COASTER WORLD ARE FIXED*/
fixed(Path,Time) :- type(path,Path), type(time,Time),
  ncc probtype(motion,Time).

/*PARTS ARE FIXED IF WHOLES ARE*/
fixed(Bit,Time) :- subpath(Bit,Path), ncc fixed(Path,Time).

/*OBJECTS IN SAME PLACE AS FIXED OBJECTS ARE FIXED*/
fixed(Obj1,T) :- type(period,T), ncc sameplace(Obj1,Obj2,T),
  ncc fixed(Obj2,T).

/*ARITHMETIC*/
/*-----------*/


/*ANGLES ARE AT RIGHT ANGLES*/
rtans(X,Y) :- Z is X-Y, 0 is Zmod180.

/*TURN ANGLE THROUGH 90 ANTI-CLOCKWISE*/
rightturn(Ans1,Ans2) :-
  eval(Ans1++90,Ans2).

/*TURN ANGLE THROUGH 180*/
aboutturn(Ans1,Ans2) :-
  eval(Ans1++180,Ans2).

cosfact(Dir,Dir,1) :- !.
cosfact(Dir1,Dir2,cos(Ans)) :-
  eval(Dir1--Dir2,Ans).

  vecadd([Mas1,Dir],[Mas2,Dir],[Mas,Dir])
        :- var(Mas1), eval(Mas-Mas2,Mas1)  ;
           var(Mas2), eval(Mas-Mas1,Mas2) ;
           eval(Mas1+Mas2,Mas),
           !.



/* MOMENTS */
/*---------*/


  summoments(Point1,Rod,Msum,Dir,Time)
        :- foreach( point_of(Rod,Point2) & diff(Point1,Point2),
                    moments(Point2,Point1,M,Dir,Time),
                    +,
                    M,
                    Msum
                  ).
```

```
  moments(Point2,Point1, F*D ,Dir1,Time)
        :- cc separation(Point1,Point2,D,Dir1,Time),
           rishtturn(Dir1,Dir2),
           sumforces(Point2,Dir2,Time,F).


/*FORCE*/
/*-----*/

sumforces(Part,Dir,Time,Gforce+Pressure+Others) :-
  gravitational(Part,Dir,Time,Gforce),
  reactional(Part,Dir,Time,Pressure),
  appliedforces(Part,Dir,Time,Others).

/*GRAVITATIONAL FORCE*/
gravitational(Part,Dir,Time,M*g*Costerm) :-
  cc mass(Part,M,Time), cosfact(270,Dir,Costerm).

/*REACTION FROM CONTACT*/
reactional(Part,Dir,Time,Forcesum) :-
  foreach(candidate(Part,Cand,Time),
  reac(Part,Cand,Force,Dir,Time),[+,0],Force,Forcesum).

/*SOURCE OF REACTIONAL FORCE*/
candidate(Part,earth,Time) :- fixed(Part,Time).

candidate(Part,Point-Line,Time) :-
  ncc sameplace(Part,Point,Time), diff(Part,Point),
  point_of(Line,Point), minimal(Line).

/*REACTION OF EARTH ON A FIXED OBJECT*/

reac(Part,earth,Costerm*R,Dir,Time) :- !,
  cc reaction(earth,Part,R,Dir1,Time),
  cosfact(Dir,Dir1,Costerm).

/*TENSION IN STRING*/

reac(Part,Point-Str,Costerm*T,Dir,Time) :-
  type(string,Str), !, cc tension(Str,T,Time),
  dc end(Str,Point,Par), cc incline(Str,Dir1,Point),
  condturn1(left,Par,Dir1,Dir2), cosfact(Dir,Dir2,Costerm).

/* REACTION OF A PATH */

reac(Part,Point-Path,Nforce+Friction,Dir,Time) :-
  ncc solid(Path), !,
  normal_reac(Part,Path,Point,Nforce,Dir,Time),
  ( roush(Path) & dc cued(motion(Part,Path,Start,Side,Time)) ->
    friction(Part,Path,Point,Friction,Dir,Time) ; Friction = zero ).

/*OTHERWISE ZERO */

reac(Part,Point-Path,zero,Dir,Time).


/*NORMAL REACTION OF PATH*/
normal_reac(Part,Path,Point,Costerm*R,Dir,Time) :-
  cc angle(Path,Dir2,Point), wh_side(Part,Path,Ans,Nc,Time),
  condturn1(Nc,Ans,Dir2,Dir1),
  cc reaction(Path,Part,R,Dir1,Time),
  cosfact(Dir,Dir1,Costerm).

/*SLIDING FRICTION*/
friction(Part,Path,Point,Costerm*Mu*R,Dir,Time) :-
```

```
  cc incline(Path,Dir2,Point),
  dc cued(motion(Part,Path,Start,Side,Time)),
  condturn3(Path,Start,Dir2,Dir3),
  cosfact(Dir,Dir3,Costerm).



/*ADDITIONAL APPLIED FORCES*/
appliedforces(Part,Dir,Time,Other) :-
  foreach(force(Part,F,Dir1,Time),cosfact(Dir,Dir1,Costerm),[+,0],
    Costerm*F,Other).

/*CONSTANT VELOCITY AND ACCELERATION*/
constvel(Obj,Time) :- dc relvel(Obj,earth,V,Dir,Time), isainvar(V).

constvel(Obj,Time) :- dc relaccel(Obj,earth,zero,Dir,Time).

constvel(Obj,Time) :-
  dc cued(line_motion(Obj,Path,Time)),
  foreach(ncc(sameplace(Obj,Point,Time))&diff(Obj,Point),
    reac(Obj,Point,Force,0,Time),[+,zero],Force,zero).


constrelvel(Obj,Earth,Time) :- constvel(Obj,Time).

/*CONSTANT ACCELERATION*/
constaccel(Obj,Time) :-  constvel(Obj,Time), !, fail.

constaccel(Obj,Time) :- dc relaccel(Obj,earth,A,Dir,Time), isainvar(A).

constaccel(Obj,Time) :- false,
  forall(dc force(Obj,F,Dir,Time),isainvar(F)).
/*DISABLED, NEEDS RETHINKING*/

/*VELOCITY*/
/*--------*/

/*RELATIVE VELOCITY BETWEEN TWO POINTS OF REFERENCE*/
relvel(P1,P2,zero,Dir,Time) :-
  ncc fixed_contact(P1,P2,Time).

relvel(Part1,Part2,Vel,Dir,Mom) :-
  subtime(Mom,Per), ncc constrelvel(Part1,Part2,Per),
  ncc relvel(Part1,Part2,Vel,Dir,Per).

relvel(Part1,Part2,Vel,Dir,Per) :-
  subtime(Mom,Per), ncc constrelvel(Part1,Part2,Per),
  ncc relvel(Part1,Part2,Vel,Dir,Mom).

relvel(End1,Midpt,Ra,Dir1,Time) :-
  dc cued(strinssys(Str,Lbit,Midpt,Rbit,Time)),
  farend(Str,End1,End2),
  farend(Bit1,End1,Midpt), alons(Bit1,Dir1,End1),
  farend(Bit2,Midpt,End2), alons(Bit2,Dir2,Midpt),
  ncc relvel(Midpt,End2,Ra,Dir2,Time).

relvel(P1,P2,A,Dir,Time) :-
  ncc fixed_contact(P2,P3,Time),
  ncc relvel(P1,P3,A,Dir,Time).

relvel(P1,P2,A,Dir,Time) :-
  ncc relvel(P2,P1,A,Dir1,Time),
  aboutturn(Dir1,Dir).
```

```
/*------------*/

/*ACCELERATION COMPONENT*/


/*ACCEL TOWARDS THE CENTRE OF A CURVE*/

accel_cmpnt(Part,-(V↑2)*Costerm/R,Dir2,Time) :-
  dc cued(motion(Part,Path,Start,Side,Time)),
  thnot( dc concavity(Path,stline) ),
  cc radius(Path,R),
  cc normal(Path,Dir3),
  cosfact(Dir2,Dir3,Costerm),
  cc vel(Part,V,Dir1,Time), !.


accel_cmpnt(Part,A*Costerm,Dir1,Time) :-
  cc accel(Part,A,Dir2,Time), cosfact(Dir1,Dir2,Costerm).


/*RELATIVE ACCELERATION BETWEEN TWO POINTS OF REFERENCE*/
relaccel(P1,P2,zero,Dir,Time) :-
  ncc fixed_contact(P1,P2,Time).

relaccel(Part1,Part2,zero,Dir,Per) :- ncc constrelvel(Part1,Part2,Per),
                                      type(period,Per).

relaccel(End1,Midpt,Ra,Dir1,Time) :-
  dc cued(strinssys(Str,Lbit,Midpt,Rbit,Time)),
  farend(Str,End1,End2),
  farend(Bit1,End1,Midpt), alons(Bit1,Dir1,End1),
  farend(Bit2,Midpt,End2), alons(Bit2,Dir2,Midpt),
  ncc relaccel(Midpt,End2,Ra,Dir2,Time).

relaccel(P1,P2,A,Dir,Time) :-
  ncc fixed_contact(P2,P3,Time),
  ncc relaccel(P1,P3,A,Dir,Time).

relaccel(P1,P2,A,Dir,Time) :-
  ncc relaccel(P2,P1,A,Dir1,Time),
  aboutturn(Dir1,Dir).



/* TIME */

  bndy(P,M,Par)
        :- nonvar(M),
           ?(bndy(P1,M,Par1)),
           adj(P,P1,Par,Par1).

  bndy(P,M,Par)
        :- var(M),
           adj(P,P1,Par,Par1),
           ?(bndy(P1,M,Par1)).



  adj(P1,P2,Par,Par) :- concurrent(P1,P2,Par).

  adj(P1,P2,Par1,Par2)
        :- opposite(Par1,Par2),
           consecutive(P1,P2,Par2).
```

```
consecutive(P1,P2,left) :- consec(P1,P2).

consecutive(P1,P2,risht) :- consec(P2,P1).


consec(P1,P2)
      :- consec1(Q1,Q2),
         concurrent(P1,Q1,risht),
         concurrent(P2,Q2,left).


consec(Q1,Q2)
      :- dc partition(E,Plist),
         type(period,E),
         nextto(Q1,Q2,Plist).



sumdurs([],0).

sumdurs([P|Ps],T+Sum)
      :- cc duration(P,T),
         !,
         sumdurs(Ps,Sum).



/* MISCELLANEOUS */


free(Path,Part,Per)
      :- forall( ncc(sameplace(Part,Point,Per)), point_of(Path,Point) ),
         ncc coeff(Path,zero),
         thnot(force(Part,F,Dir,Per)).



probtype(motion,Per)
      :- ncc probtype(roller-coaster,Per) ;
         ncc probtype(motion-in-a-straisht-line,Per).


coeff(Path,zero)
      :- type(path,Path),
         ncc probtype(motion,Time).

coeff(Path,Mu)
      :- bitof(Path,Suppath),
         ncc coeff(Suppath,Mu).



roush(Path) :- thnot( ncc(coeff(Path,zero)) ).


solid(X) :- bitof(X,Y), ncc solid(Y).



mass(Part,M,Per1)
      :- type(particle,Part),
         type(time,Per2),
         diff(Per1,Per2),
```

```
  place(X) :- type(point,X), ncc fixed(X,Time).


/*=============================================================================*/
  :-end.
```

```
/*   Various initialisations for MECHO.   */
```

init

```
const(s).

const(zero).


ina(2240,lbs,tons).
ina(60,mins,hrs).
ina(60,secs,mins).
ina(1760,yds,mls).
ina(3,ft,yds).
ina(12,ins,ft).

inaf(100,centi(X),X).
inaf(1000,milli(X),X).
inaf(1/1000,kilo(X),X).



:-  ccflag := on,
     tflag := 4.



isa(point_of_ref,earth).
```

```
/*     MECHO  10.1     */
```

Mecho

```
  :- consult(logic),
     consult(xtract),
     consult(infer),
     consult(init).

version
  :-  nl,nl,
      tab(8),write('================================'),nl,nl,
      tab(8),write('        MECHO       10.1'),nl,nl,
      tab(8),write('================================'),nl,nl,
      tab(8),write(' ( DEBUG  version 4 loaded )'),nl,nl.


  :- version.
```

PROLOG, MLC

```
.COPY PROG= 'A
.RUN NEWJOB[400,422]
*:-''NOLC''.
*?-CONSULT(PROG).
*?-SAVE(''PROG.IS'').
.CONT
*?-RUN.
*:-END.
.DEL PROG
```

/PROLOG <USVW.Q, INEQ, 500>

/PROLOG <USVW.Q, INTER, LOGIC, XTRACT / TEST. / PULLBS>

/PROLOG <USVW.Q, PEANO>.

```
/*CNVERT*/
/*MECHANICS UNIT CONVERSION ROUTINES*/
/*GATHERED TOGETHER BY ALAN BUNDY ON 8/9/76*/



/*UNIT CONVERSION*/



CONVERT(_FS,_FS1) :- STANDSYS(_SYS),
  RETRACT(CCFLAG(_SW)), ASSERTA(CCFLAG(ON)), ECONV(_FS,_FS1).

SCONV([],[]).

SCONV(_F._FS,_F1._FS1) :- ECONV(_F,_F1), SCONV(_FS,_FS1).

ECONV(TRUE,TRUE) :- !.

ECONV(_F,_F) :- ISCONST(_F), !.

ECONV(_F,_F) :- WORD(_F), KIND(_F,_PRED),
    TYPEINFO(_PRED,DIMLESS,1), !.

ECONV(_F,_F1) :- WORD(_F) , !, VCONV(_F,_F1).

ECONV(_F,_F1) :-  _F =.. _S._ARGS , SCONV(_ARGS,_ARGS1) ,
     _F1 =.. _S._ARGS1 .

VCONV(_F,_F1) :- VCONV1(_F,_F1),!.

VCONV(_F,_F1) :- VCONV2(_F,_F1),!,
    CASSERT2(VCONV1(_F,_F1)),TRACE(_F-GOES-TO-_F1,4).

VCONV2(_V,_M) :- UNIT(_V,_U) , STANDARD(_U,_U),
!, CC(MEASURE(_V,_M)).

VCONV2(_V,_M2) :- UNIT(_V,_U)
, STANDARD(_U,_U0) , CC(MEASURE(_V,_M)),
    MCONV(_M,_U,_M1,_U0), SIMPLIFY(_M1,_M2).

STANDARD(_U,_U0) :- STANDSYS(_SYS), DIMENSIONS(_U,_D),
    STANDUNIT(_U,_U0,_SYS).


DIMENSIONS(LRS,M) :- !.

DIMENSIONS(TONS,M) :- !.


DIMENSIONS(HRS,T) :- !.

DIMENSIONS(MINS,T) :- !.

DIMENSIONS(SECS,T) :- !.

DIMENSIONS(MLS,L) :- !.

DIMENSIONS(YDS,L) :- !.
```

```
DIMENSIONS(FT,L) :- !.

DIMENSIONS(INS,L) :- !.

DIMENSIONS(_U,_D) :- ISCONST(_U), !.

DIMENSIONS(_U,_D) :-  _U =.. _S._ARGS  ,
    LDIMENSIONS(_ARGS,_DARGS),  _D =.. _S._DARGS .


LDIMENSIONS([],[]).

LDIMENSIONS(_U._UL,_D._DL) :- DIMENSIONS(_U,_D) ,
    LDIMENSIONS(_UL,_DL).

MCONV(_M,_U,_C._M,_U1) :- FACTOR(_C,_U1,_U).

/*CONVERSION FACTORS BETWEEN UNITS*/


FACTOR(_C1/_C2,_U1,_U2) :-
  FINDREP(_C1,_U1,_R), FINDREP(_C2,_U2,_R).

FINDREP(_C*_C3,_U,_R) :-
  INA(_C,_U,_U3), !, FINDREP(_C3,_U1,_R).

FINDREP(1,_R,_R).

FMERGE(_C,_U3,_U2) :-
  FINDREP(_C3,_U3,_R1), FINDREP(_C2,_U2,_R2),
  COND(DIFF(_R1,_R2),ASSERTA(INA(_C*_C1/_C2,_R1,_R2)),TRUE).

INA(2240,LBS,TONS).

INA(60,MINS,HRS).

INA(60,SECS,MINS).

INA(1760,YDS,MLS).

INA(3,FT,YDS).

INA(12,INS,FT).

INA(100,CENTI(_X),_X).

INA(1000,MILLI(_X),_X).

INA(1/1000,KILO(_X),_X).

ISUNIT(_U) :- DIMENSIONS(_U,_D).


/*FIXING UNIT SYSTEM*/

STANDSYS(_SYS) :-
  NOLOOP ,SOUGHTS(_SI ), CONVLIST(UNIT,_SL,_UL1),
  SWORDSIN(_UL1,_PUL1), FIX(1.L.M.[],_PUL1,_REM1,_SYS),
  GIVENS(_GI ) ,CONVLIST(UNIT,_GI,_UL2),
  SWORDSIN(_UL2,_PUL2), FIX(_REM1,_PUL2,_REM2,_SYS),
  FIXREST(_REM2,_SYS),ASSERTA(STANDSYS(_SYS)),
```

```
  SYSFOUND.

NOLOOP :- SYSSOUGHT, !, FAIL.
NOLOOP :- ASSERTA(SYSSOUGHT).

SYSFOUND :- RETRACT(SYSSOUGHT).

FIX([],_UL,[],_SYS) .

FIX(_D1._DL1,_UL,_DL2,_SYS) :-
  SUBLIST(DIMENSIONS(_D1),_UL,_CANDL), DIFF(_CANDL,[]),
   !, VOTE(_CANDL,_WIN),
  CGENSYM(SYS,_SYS), ASSERTA(STANDUNIT(_D1,_WIN,_SYS)),
  FIX(_DL1,_UL,_DL2,_SYS).

FIX(_D1._DL1,_UL,_D1._DL2,_SYS) :-
  FIX(_DL1,_UL,_DL2,_SYS).

VOTE(_CANDL,_WIN) :-
  LISTTOSET(_CANDL,_CANDS),
  MAPLIST(SCOREFACH(_CANDL),_CANDS,_PAIRS),
  FAVOURITE(_PAIRS,_WIN._SCORE).

SCOREEACH(_U,_U.0,[]).
SCOREEACH(_U,_U._N1,_U._REST) :-
  !, SCOREEACH(_U,_U._N,_REST) ,_N1 IS _N+1.
SCOREEACH(_U,_U._N,_HD,_TL) :- SCOREEACH(_U,_U._N,_TL).

FAVOURITE((_U._S).[],_U._S) :- !.
FAVOURITE((_U1._S1)._PL,_U2._S2) :-
  FAVOURITE(_PL,_U2._S2), _S1(_S2 , !.
FAVOURITE((_U._S)._PL,_U._S).

FIXREST([],_SYS).

FIXREST(1.1._M.[],_SYS) :- !, DEFAULT(_SYS).

FIXREST(_D._DL,_SYS) :-
  DEFAULT(_SYS2) ,STANDUNIT(_D,_U,_SYS2),
  ASSERTA(STANDUNIT(_D,_U,_SYS)),FIXREST(_DL,_SYS).

DEFAULT(FLS).

STANDUNIT(1,SFOS,FLS) :- !.
STANDUNIT(I,FT,FLS) :- !.
STANDUNIT(M,LRS,FLS) :- !.

STANDUNIT(_U,_U,_SYS) :- ISCONST(_U) ,!.

STANDUNIT(_D,_U,_SYS) :- ISSYM(_D), !,FAIL.

STANDUNIT(_D,_U,_SYS) :- _D=..(_S._DARGS) ,
  MAPLIST(STANDUNIT(_SYS),_DARGS,_UARGS), _U=..(_S._UARGS) .



MEASURE(ZERO,0) :- !.

MEASURE(G,32) :- !.

CCMEASURE(_Q,_M) :- RETRACT(CCFLAG(_SW)),
```

```
  ASSERTA(CCFLAG(ON)), CC(MEASURE(_Q,_M)).


UNIT(ZERO,_U) :- !.

UNIT(G,FT/SECS:2) :- !.

UNIT(_Q,_U) :-
  STANDSYS(_SYS), KIND(_Q,_PREV),
    TYPEINFO(_PREV,_TYPE,_U), STANDUNIT(_U,_U,_SYS).
```

```
/*NEWVEL*/


/*VELOCITY RATIONALIZED*/
/*----------------------*/

/*ASSERTING NEW VEL RELATIONS*/
ASSVEL(←OBJ,←VEL,←DIR,←TIME) :-
  REP(←OBJ,←REP,TOUCH(←TIME)),
  RECPR(RELVEL(←REP,EARTH,←VEL,←DIR,←TIME),4),

ASSRELVEL(←OBJ1,←OBJ2,←VEL,←DIR,←TIME) :-
  REP(←OBJ1,←REP1,TOUCH(←TIME)), REP(←OBJ2,←REP2,TOUCH(←TIME)),
  RECPR(RELVEL(←REP1,←REP2,←VEL,←DIR,←TIME),4),

/*FINDING RELATIVE VELOCITYS*/
RELVEL(←OBJ,←OBJ,ZERO,←DIR,←TIME) :- !,

RELVEL(←OBJ1,←OBJ2,VEL,←DIR,TIME) :-
  REP(←OBJ1,REP1,TOUCH(←TIME)), REP(←OBJ2,←REP2,TOUCH(←TIME)),
  ?(RELVEL(←REP1,←REP2,←VEL,←DIR,←TIME)),

/*SINGLETON TREE FOR EARTH*/
REP(EARTH,EARTH,TOUCH(←TIME)),

/*ABOVE DOES NOT DEAL WITH: */
/*  COMMUTATIVE PROPERTY OF RELVEL*/
/*  SPECIAL STRINGSYS RULE*/
/*  TIME INHERITANCE*/
```

```
 TYPE LEVERS.AB
/*LEVERS*/

/*A BUNDY APRIL 1978*/

/*XTRACT*/

RELATES(MOMENTS,DMASS,LENGTH,FORCE,MOMENTSJ).

FINDINFO(MOMENTS,_OBJ-_ANG-_TIME,_POINT2-_TIME) :-
    ?(ROD(_ROD,_TIME)), CONNECTION(_OBJ,_ROD,_POINT1,_TIME),
    POINT_OF(_ROD,_POINT2), THNOT(SAMEPLACE(_POINT1,_POINT2,_TIME)).

ISFORM(_MSUM-ZERO,MOMENTS-(_POINT-_TIME)) :-
    POINT_OF(_ROD,_POINT), ?(ROD(_ROD,_TIME)), NCC(INCLINE(_ROD,_DIR,_PO
INT)),
    SUMMOMENTS(_POINT,_MSUM,_DIR,_TIME).

/*OBJ1&OBJ2 ARE CONNECTED VIA POINT1*/

CONNECTION(_OBJ1,_OBJ2,_POINT1,_TIME) :-
    FINDPOINT(_OBJ1,_POINT1), FINDPOINT(_OBJ2,_POINT2),
    SAMEPLACE(_POINT1,_POINT2,_TIME).

FINDPOINT(_OBJ,_POINT) :-
    COND(POINT_OF_REF(_OBJ), _POINT=_OBJ,       POINT_OF(_OBJ,_POINT)).


/*INFIX*/

/*TAKE MOMENTS ABOUT POINT1*/

SUMMOMENTS(_POINT1,_MSUM,_DIR,_TIME) :-
    SUMEACH(POINT_OF(_ROD,_POINT2) & DIFF(_POINT1,_POINT2),
            MOMENTS(_POINT2,_POINT1,_M,_DIR,_TIME),_M,_MSUM).

/*MOMENTS CONTRIBUTED BY POINT2*/

MOMENTS(_POINT2,_POINT1,_F*_D,_DIR1,_TIME) :-
    CC(SEPERATION(_POINT1,_POINT2,_D,_DIR1,_TIME)),
    RIGHTTURN(_DIR1,_DIR2), SUMFORCES(_POINT2,_DIR2,_TIME,_F).

POINT_OF(_LINE,_POINT) :- ?(POINT_OF(_LINE,_POINT)).
POINT_OF(_LINE,_POINT) :- ?(ROD(_LINE,_TIME)), ?(CENTRE_OF_GRAVITY(_LINE
,_POINT)).


/*DEFN OF SEPERATION BY SIM CLASSES*/

SEPERATION(_P1,_P2,_R,_A,_TIME) :-
    SROOT(_P1,_ROOT,_R1,_A1,_TIME),
    SROOT(_P2,_ROOT,_R2,_A2,_TIME),
    VECADD(_R,_A,_R2,_A2,_R1,_A1).

SROOT(_P,_ROOT,_R,_A, _TIME) :-
    SLINK(_P,_P1,_R1,_A1,_TIME),
    SROOT(_P1,_ROOT,_R2,_A2,_TIME),
    VECADD(_R1,_A1,_R2,_A2,_R,_A).

SROOT(_ROOT,_ROOT,ZERO,_A,_TIME).

SMERGE(_P1,_P2,_R,_A,_TIME) :-
```

```
    SROOT(_P1,_ROOT1,_R1,_A1,_TIME),
    SROOT(_P2,_ROOT2,_R2,_A2,_TIME),
    VECADD1(_R3,_A3,_R1,_A1,_R,_A),
    COND(DIFF(_ROOT1,_ROOT2),ASSERTA(SLINK(_ROOT1,_P2,_R3,_A3,_TIME)),
         TRUE).


VECADD1(_R1,_A,_R2,_A,_R1+_R2,_A) :- !.

VECADD1(_R2-_R1,_A,_R1,_A,_R2,_A) :- !.

VECADD1(_R1,_A1,_R2,_A2,_R3,_A3) :-
    TRACE(UNABLE-TO-ADD-VECTORS-[_R1,_A1,_R2,_A2,_R3,_A3],2).


/*LOGIC*/

SCHEMA(ROOT(_ROOT,_TIME),
    [CUT(LINESYS(_ROOT,_END1,_END2)),
      CC(INCLINE(_ROOT,_DIR,_END1)),
      CC(SEPERATION(_END1,_END2,_D,_DIR,_TIME)),
      CC(CENTRE-OF-GRAVITY(_ROOT,_CORG))  ],
    [CONCAVITY(_ROOT,STLINE),
      SEPERATION(_END1,_CORG,_D/2,_DIR,_TIME)],
    [] ).


/*NFORM WITHIN  LOGIC*/

NFREC(SEPERATION(_OBJ1,_OBJ2,_D,_ANG,_TIME),_N) :-
    !, SMERGE(_OBJ1,_OBJ2,_D,_ANG,_TIME).

NFREC(CONSTLENGTH(_LINE,_D),_N) :-
    ?(CONCAVITY(_LINE,STLINE)), !,
    CC(END(_LINE,_END1,LEFT)),
    CC(END(_LINE,_END2,RIGHT)),
    CC(INCLINE(_LINE,_DIR,_END1)),
    FORALL(TIME(_TIME), SMERGE(_END1,_END2,_D,_DIR,_TIME)).


VARLENGTH(_LINE,_D,_TIME) :-
    ?(CONCAVITY(_LINE,STLINE)),
    ?(END(_LINE,_END1,LEFT)),
    ?(END(_LINE,_END2,RIGHT)),
    SEPERATION(_END1,_END2,_D,_DIR,_TIME).


.
```

Usual conventions for variable names in parsing routines

A	<and>	list | nolist | sublist
F	<add flag>	add | test
H	<possession>	poss | noposs
Y	<type>	physobj | defmeas | ...
S	<selection>	one | other | last | all
G	<general object>	<atom> | <var>
D	<dimension>	mass | height | velocity ...
O	<object term>	Y : G
N	<name>	<letter> <> <subscript>
N	<number>	<integer> | <name> | two | three | ...
S	<plurality>	sing | plur
P	<part spec.>	end | ...
C	<class>	pulley | string | ...
A	<adj>	fine | constant | ...
U	<unit>	lb | lbs | gm | * | ...
M	<measpair>	[N, U]
I	<indefmeas>	D # M
W	<word>	<atom>
T	<time>	period <> <integer>
V	<verb>	pass | eat | ...
L	<literal>	<term>
P	<preposition>	on | in | ...
X	<sentence>	
D	<determinacy>	det | nondet
B		ass | noass

UTILITIES & REFERENCE HANDLING

SENTENCE DATASTRU

FIELDS

```
:- entry(refs,2).
:- entry(ment,2).
:- entry(add1,2).
:- entry(putconstr,2).
:- entry(constr,2).

:- program([util,distri,read,new,add,senj],[
setrnf(2), isrnf(2), closernf(1),
sethv(2), ishv(2), closehv(1),
setsubj(2), issubj(2),
setobj(3), addobj(3), isobj(2), isobj(3),
setmv(2), ismv(2), ismv1(2), ismood(2),
setplur(2), currview(2),
setpp(3), pp(3),
per(2), last(2), sname(2), level(2), islast(2),
per(2), findconstr(2), putconstr(2),
mentioned(2),
lossubj(2), losobj(2),
prsent(1), prass(1) ]).

:- mode roles(+,?).
:- mode data(+,?).
:- mode per(+,?).
:- mode ident(+,?).

:- mode role1(+,?).
:- mode role2(+,?).
:- mode role3(+,?).
:- mode role4(+,?).
:- mode role5(+,?).
:- mode role6(+,?).

:- mode data1(+,?).
:- mode data2(+,?).
:- mode data3(+,?).
:- mode data4(+,?).

:- mode ident1(+,?).
:- mode ident2(+,?).
:- mode ident3(+,?).

/* LOGICAL ROLES */
lossubj(G,X) :- ismood(actv,X), !, issubj(G,X).
lossubj(G,X) :- ismood(passv,X), !, pp(by,G,X).

losobj(G,X) :- ismood(actv,X), !, isobj(G,1,X).
losobj(G,X) :- ismood(passv,X), !, issubj(G,X).

/*  SENTENCE OUTPUT  */

prsent(sent(r(R,H,S,O,M,P),_,I,p(L,N,_))) :-
   nl, nl, print(final-parse-of-N-followins-from-L), nl, nl,
   print(roles-filled), nl, pnotvar(R), pnotvar(H), pnotvar(S), pnotvar(O),
   pnotvar(M), pslist(P), nl, fail.
prsent(_).

prass(sent(_,d(A,R,_,_),_,_)) :- nl, nl, print(add-assertions), pslist(A),
   nl, call(assertall(A)), nl, fail.
prass(_).

pnotvar(X) :- var(X), !.
pnotvar(X) :- write(X), nl.

pslist(X) :- var(X), !, nl.
```

```
/* ACCESSING SENTENCE FIELDS */

roles(sent(R,_,_,_),R).
data(sent(_,D,_,_),D).
per(sent(_,_,T,_),T).
ident(sent(_,_,_,I),I).

role1(r(X,_,_,_,_,_),X).
role2(r(_,X,_,_,_,_),X).
role3(r(_,_,X,_,_,_),X).
role4(r(_,_,_,X,_,_),X).
role5(r(_,_,_,_,X,_),X).
role6(r(_,_,_,_,_,X),X).

data1(d(X,_,_,_),X).
data2(d(_,X,_,_),X).
data3(d(_,_,X,_),X).
data4(d(_,_,_,X),X).

ident1(p(X,_,_),X).
ident2(p(_,X,_),X).
ident3(p(_,_,X),X).

rnf(X,R) :- roles(X,Rs), role1(Rs,R).
hv(X,H) :- roles(X,R), role2(R,H).
subj(X,S) :- roles(X,R), role3(R,S).
obj(X,O) :- roles(X,R), role4(R,O).
mv(X,M) :- roles(X,R), role5(R,M).
pps(X,P) :- roles(X,R), role6(R,P).
add1(X,A) :- data(X,D), data1(D,A).
refs(X,R) :- data(X,D), data2(D,R).
constr(X,C) :- data(X,D), data3(D,C).
ment(X,M) :- data(X,D), data4(D,M).
last(X,L) :- ident(X,I), ident1(I,L).
sname(X,N) :- ident(X,I), ident2(I,N).
level(X,L) :- ident(X,I), ident3(I,L).

/* PRIMITIVE FUNCTIONS ON FIELDS */

setrnf(S,X) :- rnf(X,rnf(S,F)).
isrnf(S,X) :- rnf(X,T), notvar(T), T=rnf(S,F), var(F).
closernf(X) :- rnf(X,rnf(_,closed)).

sethv(V,X) :- hv(X,hv(V,_)).
ishv(V,X) :- hv(X,W), notvar(W), W=hv(V,F), var(F).
closehv(X) :- hv(X,hv(_,closed)).

setsubj(O,X) :- subj(X,subj(O)).
issubj(S,X) :- subj(X,T), notvar(T), T=subj(S).

setobj(S,N,X) :- obj(X,obj(N#S,_)).
addobj(S,N,X) :- obj(X,obj(_,N#S)).
isobj(S,X) :- obj(X,O), notvar(O), O=obj(N#S,_), var(N).
isobj(S,N,X) :- obj(X,O), notvar(O), O=obj(A,B),
    (A=M#S;B=M#S), notvar(M), M=N.

setmv(I,X) :- mv(X,I).
ismv(V,X) :- mv(X,mainverb(V1,_)), notvar(V1), V=V1.
ismv1(I,X) :- mv(X,J), J=mainverb(V,I1), notvar(V), I=J.
ismood(M,X) :- mv(X,mainverb(_,[_,M1,_._])), notvar(M1), M1=M.

setplur(N,X) :- mv(X,mainverb(_,[N,_._])).

currview(W,X) :- mv(X,mainverb(_,[_,_,W1,_._])), notvar(W1), W=W1.
```

```
setpp(P,C,X) :- pps(X,Q), lastof(Q,[pp(P,C,_),_,_]).

pp(P,C,X) :- pps(X,Q), sin(pp(P,C,F),Q), var(F), F=closed.

islast(S,X) :- last(X,T), notvar(T), T=S.

findconstr(C,X) :- constr(X,L), sin(C,L).
putconstr(C,X) :- constr(X,L), lastof(L,[C,_,_]).

mentioned(G,X) :- ment(X,M), lastof(M,[G,_,_]).

:- end.
```

UTILITIES

```
:- entry(notvar,1).
:- entry(lastof,2).
:- entry(sin,2).
:- entry(print,1).
:- entry(append,3).
:- entry(nondet,1).
:- entry(diff,2).
:- entry(in,2).
:- entry(sround,1).
:- entry(trace,2).
:- entry(listlensth,2).
:- entry(sensym,2).
:- entry(snoc,2).

:- module(util,[
sin(2), notin(2),
lastof(2), snoc(2),
notvar(1), notlist(1), nondet(1),
reset(0),
sensym(2), sensymlist(2), csensym(3),
append(3), sapp(3),
listlensth(2), thnot(1),
in(2), sround(1), plur(2),
trace(2), tlim(1), print(1),
diff(2), diffall(1) ]).

:- mode sensymlist(+,+).
:- mode listlensth(+,-).
:- mode in(?,+).
:- mode trace(+,+).
:- mode print(+).

sin(_,X) :- var(X), !, fail.
sin(X,[X,..._]).
sin(X,[_,..Y]) :- sin(X,Y).

notin(X,L) :- sin(X,L), !, fail.
notin(_,_).

lastof(X,Y) :- var(X), !, X=Y.
lastof([_,..X],Y) :- lastof(X,Y).

snoc(X,Y) :- var(Y), !, Y=[X,..._].
snoc(X,[X,..._]) :- !.
snoc(X,[_,..L]) :- snoc(X,L).

notvar(X) :- var(X), !, fail.
notvar(_).

notlist(X) :- var(X), !.
notlist([]) :- !, fail .
notlist([_,..._]) :- !, fail.
notlist(_).

   nondet(X) :- var(X), !.
   nondet([]) :- !, fail.
   nondet([_,..X]) :- nondet(X).

reset :- eraseall(c(_,_)).

sensym(P,Q) :- recorded(c(P,N),Ptr), erase(Ptr), N1 is N+1,
   record(c(P,N1)), name(P,Na), append(Na,[N1],R), name(Q,R), !.
sensym(P,Q) :- record(c(P,49)), name(P,Na), append(Na,[49],R),
   name(Q,R), !.
```

```
append([X|Y],Z,[X|W]) :-  append(Y,Z,W), !.
append([],X,X).

sapp(A,B,C) :- nondet(A), !, append(B,A,C). /* *** */
sapp(A,B,C) :- append(A,B,C).

sensymlist(P,[S|Ss]) :- !, sensym(P,S), sensymlist(P,Ss).
sensymlist(_,[]) :- !.


listlength(0,[]) :- !.
listlength(N,[_|X]) :- integer(N), !, M is N-1, listlength(M,X).
listlength(_,[_|_]).

csensym(test,_,_) :- !.
csensym(_,_,G) :- sround(G), !.
csensym(_,C,G) :- sensymlist(C,G).

thnot(L) :- call(L), !, fail.
thnot(_).

in(X,L1-L2) :- L1==L2, !, fail.
in(X,[X|_]-_).
in(X,[_|L1]-L2) :- in(X,L1-L2).

sround(X) :- var(X), !, fail.
sround([X|_]) :- var(X), !, fail.
sround([_|X]) :- !, sround(X).
sround(_).

plur(you,plur).
plur([_],sing).
plur([_,_|_],plur).

/*  TRACING AND PRINTING  */

trace(M,N) :- ?tflag(N1), N1<N, !.
trace(M,N) :- print(M), ttyflush, repeat, set0(13), !.

tlim(N) :- eraseall(tflag(X)), record(tflag(N)), !.

print(X-Y) :- print(X), fail.
print(X-Y) :- !, put(32), write(Y).
print(X) :- put(42), put(32), write(X).

diff(X,X) :- !, fail.
diff(_,_).

diffall([]).
diffall([A|As]) :- diff1(A,As), diffall(As).

diff1(_,[]).
diff1(A,[A|_]) :- !, fail.
diff1(A,[_|As]) :- diff1(A,As).

:- end.
```

DISTRIBUTION of
PROPERTIES among OBJEC

```
:- module(distri,[
distrib(5), siveall(5) ]).

:- mode distrib(+,+,+,+,+).
:- mode dis1(+,+,+,+,+).
:- mode dis2(?,+,+,+,+).
:- mode dis12(?,+,+,+,+).
:- mode siveall(+,+,?,+,+).

distrib(A,[B],N,F,X) :- (nondet(A),!,act(N,A,B,F,X,nondet)) ;
    (!,dis1(A,B,N,F,X)).
distrib([A],B,N,F,X) :- !, dis2(A,B,N,F,X).
distrib(A,B,N,F,X) :- dis12(A,B,N,F,X).

    dis1([A,..As],B,N,F,X) :- act(N,A,B,F,X,det), dis1(As,B,N,F,X).
    dis1([],_,_,_,_).

    dis2(A,[B,..Bs],N,F,X) :- act(N,A,B,F,X,det), dis2(A,Bs,N,F,X).
    dis2(_,[],_,_,_).

    dis12([A,..As],[B,..Bs],N,F,X) :- act(N,A,B,F,X,det), dis12(As,Bs,N,F,X).
    dis12([],[],_,_,_).

siveall(A,K,R,F,X) :- nondet(A), !, act(K,A,R,F,X,nondet). /* *** */
siveall([],_,[],_,_) :- !.
siveall([A,..As],K,[R,..Rs],F,X) :- act(K,A,R,F,X,det), siveall(As,K,Rs,F,X).

act(K,A,B,F,X,nondet) :- !, call(typelt(A,E,X)), act(K,E,B,F,X,det).
act(makeprop(P),A,B,F,X,_) :- !, call(makeprop(P,A,F,X,B)).
act(makepart(P),A,B,F,X,_) :- !, call(makepart(P,A,F,X,B)).
act(makeadj,G,A,F,X,_) :- !, call(makeadj(A,G,F,X)).
act(L,A,B,F,X,_) :- L=..[Fn,..As], append(As,[A,B,F,X],A1), M=..[Fn,..A1],
    call(M).

:- end.
```

```
:- module(read,[
readsent(1), final(1),
vowel(1), conson(1), letter(1), digit(1) ]).

:- mode readsent(-).
:- mode restsent(+,-).
:- mode restword(-).
:- mode word(+,-).
:- mode tolc(+,-).

/*  READING IN SENTENCE  */

readsent(S) :- get(L), restsent(L,S), !.

restsent(L,[W]) :- final(L), !, name(W,[L]), repeat, get0(13).
restsent(L,[W,..Ws]) :- restword(Ls), word([L,..Ls],W), readsent(Ws).

restword([L,..Ls]) :- get0(L), diff(L,13), diff(L,32), !, restword(Ls).
restword([]).

final(46).
final(33).
final(63).

word(Ls,W) :- tolc(Ls,L1), name(W,L1).

tolc([L,..Ls],[M,..Ms]) :- ucletter(L), !, M is L+32, tolc(Ls,Ms).
tolc([L,..Ls],[L,..Ms]) :- tolc(Ls,Ms).
tolc([],[]).

vowel(97). vowel(101). vowel(105). vowel(111). vowel(117).

conson(X) :- vowel(X), !, fail.
conson(X) :- integer(X).

ucletter(X) :- X>64, X<91.
lcletter(X) :- X>96, X<123.

letter(X) :- (ucletter(X);lcletter(X)), !.

digit(X) :- X>47, X<58.

:- end.
```

SETTING UP REFERENCES

```
:- entry(defref,3).
:- entry(setref,4).
:- entry(initcands,4).
:- entry(satall,3).

:- module(new,[
defnp(3), dofoc(2), select(4) ]).

:- mode listref(+,+,+).
:- mode flatcands(+,-).
:- mode flatc(+,-,-).
:- mode initcands(-,+,+,-).
:- mode select(+,+,?,+).
:- mode diffall(?,+,-).

/* Setting up a new reference
   2 sorts arise -
      Those from definite noun phrases
      Those from pronouns
*/

defnp(G,C,X) :- initcands(R,[hasname(C,R),..._],X,Ca),
   trace(candidates-for-G-as-C-selected,3), value(Ca,G),
   defref(G,Ca,X).

defref(G,_,_) :- ground(G), !.
defref([G],Ca,X) :- !, setref(G,Ca,det,X).
defref(G,Ca,X) :- nondet(G), !, setref(G,Ca,nondet(T),X),
   setref(T,Ca,typ(G),X).
defref(G,Ca,X) :- listref(G,Ca,X), at1(diffall(G),X,_).

dofoc(G,X) :- ment(X,L), lastof(L,L1),
   initcands(G,[in(G,L-L1),ground(G),..._],X,Ca),
   focvalue(Ca,G), focref(G,Ca,X).

focref(G,_,_) :- ground(G), !.
focref([G],Ca,X) :- !, flatcands(Ca,C), setref(G,C,det,X).
focref(G,Ca,X) :- nondet(G), !, flatcands(Ca,C), defref(G,C,X),
   setof(Ca,S), findref(G,_,test(S,_,_),_,_,X).
focref(G,Ca,X) :- flatcands(Ca,C), listref(G,C,X), setof(Ca,S),
   at1(sin(G,S),X,_).

listref([G,..Gs],Ca,X) :- setref(G,Ca,det,X), listref(Gs,Ca,X).
listref([],_,_).

setref(G,Ca,T,X) :- refs(X,L),
   lastof(L,[ref(G,I,test(_,_,Co),Ca,T),..._]), sensym(ref,I),
   add1(X,A), lastof(A,Co).

/* Producing a flattened cand list when cands are lists */

flatcands([],[]).
flatcands([p(C,_),..Cs],R) :- flatc(C,R,R1), flatcands(Cs,R1).

flatc([],R,R).
flatc([A,..As],[p(A,_),..R1],R2) :- flatc(As,R1,R2).

/* Giving it an initial candidate list */

initcands(G,Li,X,C) :- add1(X,A), lastof(A,Co), record(c([])),
   satall(Li,Co,X), recorded(c(L),P), rec(P,L,G), fail.
initcands(_,_,_,L) :- recorded(c(L),P), erase(P).

satall(L,_,_) :- var(L), !.
```

```
satall(L,Co,X) :- L/[Co,X].

rec(P,L,R) :- cand(L,R), !.
rec(P,L,R) :- erase(P), record(c([p(R,_),..L])).

/* Selecting from a set which may be a reference */


select(all,S,S,_) :- !.
select(one,S,G,X) :- !, oneref(S,G,X), putconstr(one(S,G),X).
select(other,S,G,X) :- !, oneref(S,G,X), constr(X,C), prev(S,C,P),
   difrall(G,P,A), add(test,A,X), putconstr(other(S,G),X).
select(last,S,G,X) :- !, oneref(S,G,X), constr(X,C), prev(S,C,P),
   length(P,L), L1 is L+1, listlength(L1,S), difrall(G,P,A),
   add(test,A,X), putconstr(other(S,G),X).
select(some,S,G,X) :- !, someref(S,G,X).

/* Picking an object from a set - set candidates */

listcands(S,Ca,G,[],X) :- ground(S), !, setof(Ca,S).
listcands(S,Ca,G,triv(G,E),X) :- nondet(S), !,
   findref(S,_,_,C,nondet(E),X), copyca(C,Ca).
listcands(S,Ca,G,sin(G,S),X) :- initcands(G,sin(G,S),X,Ca).

/* Selecting one member */

oneref(S,[G],X) :- listcands(S,Ca,G,L,X), defref([G],Ca,X), addtest(L,X,_).

/* Selecting more than one member */

someref(S,[G1,G2,..Z],X) :- listcands(S,Ca,E,L,X), defref([G1,G2,..Z],Ca,X),
   nextfor([G1,G2,..Z],E,L,X).

nextfor(S,E,L,X) :- ground(S), !.
nextfor(S,E,L,X) :- typelt(S,E,X), addtest(L,X,_).

/* Finding previously mentioned choices */

   prev(_,C,_) :- var(C), !, fail.
   prev(G,[other(G,[G1]),..L],[G1,..Gs]) :- !, prev(G,L,Gs).
   prev(G,[one(G,[G1]),.._],[G1]) :- !.
   prev(G,[_,..L],P) :- prev(G,L,P).

   difrall(G,[G1,..Gs],[diff(G,G1),..Ls]) :- difrall(G,Gs,L), !.
   difrall(G,[],[]).


:- end.
```

```
:- entry(copyca,2).
:- entry(filter1,2).
:- entry(at1,3).
:- entry(add,3).
:- entry(addtest,3).

:- module(add,[
instantiate(2), add(2), add(3), test(2) ]).

:- mode trycand(+,+,?).
:- mode copyca(+,-).
:- mode someallin(+,+).
:- mode allin(+,+).
:- mode inmark(+,+).
:- mode transca(+,+).
:- mode fcand(+,+,-,+,+,-).
:- mode ripple(?,-,+,+,+).
:- mode addap(+,-).
:- mode addtest(+,+,?).
:- mode refsof(+,+,+,?).

/* Test to see if an object is a suitable candidate for a ref */

notcand(C,I,C,T,X) :- trycand(T,X,[I,..._]), !, fail.
notcand(_,_,_,_,_).

trycand(test(T,V,Co),X,L) :- satall(T,Co,X), check(V,X,L).

/* Having assigned possible values safely to refs in a particular
   list, we must check the validity of others which have become
   instantiated on the way */

check(V,_,_) :- var(V), !.        /* Have reached end of list */
check([I,..Is],X,L) :- sin(I,L), !, check(Is,X,L)
          . /* That R was in the list of those looked at */
check([I,..Is],X,L) :- findref(_,I,T,_,_,X), trycand(T,X,L),
   lastof(L,[I,..._]), check(Is,X,L).  /* That one wasn't */

/* When extra tests have been added, it is necessary to try and prune
   the existing candidate lists */

filter(I,X) :- findref(G,I,T,Ca,Y,X), trace(filtering-reference-for-G,4),
   (ground(G);fcand(Ca,I,G,T,X,F)), !, trace(filtering-for-G-over,4),
   masterfilter(F,Y,X), ripple(F,G,Ca,T,X), !.

/* Do repercussions when a nondet's typ has been reduced */

masterfilter(F,_,_) :- var(F), !.
masterfilter(_,typ(R),X) :- !, findref(R,_,test(T,_,_),Ca,_,X),
   ndfilter(T,Ca), ndvalue(Ca,R).
masterfilter(_,_,_).

ndfilter(T,_) :- var(T), !.
ndfilter(L,Ca) :- copyca(Ca,C), someallin(L,C), transca(C,Ca).

/* Deal with list candidates for nondet refs */

copyca([p(C,F),..Cs],[p(C,G),..Ds]) :- var(F), !, copyca(Cs,Ds).
copyca([_,..Cs],Ds) :- copyca(Cs,Ds).
copyca([],[]).

someallin([L,..Ls],Ca) :- (allin(L,Ca);true), !, someallin(Ls,Ca).
someallin([],_).
```

DEALING WITH

ADD & TEST ASSERTIO

```
allin([A,..As],Ca) :- inmark(A,Ca), allin(As,Ca).

inmark(A,[p(A,in),..._]) :- !.
inmark(A,[_,..Cs]) :- inmark(A,Cs).

transca(C,[p(_,F),..Cs]) :- notvar(F), !, transca(C,Cs).
transca([p(C,F),..C1],[p(C,out),..C2]) :- var(F), !, transca(C1,C2).
transca([_,..C1],[_,..C2]) :- transca(C1,C2).
transca([],[]).

/* Run throush a candidate list and set flag if something can be removed */

fcand([],_,_,_,_,_).
fcand([p(C,F),..Cs],I,G,T,X,out) :- var(F), notcand(C,I,G,T,X), !, F=out,
   trace(candidate-C-invalid,4), fcand(Cs,I,G,T,X,_).
fcand([p(C,F1),..Cs],I,G,T,X,F) :- var(F1), !, trace(candidate-C-valid,4),
   fcand(Cs,I,G,T,X,F).
fcand([_,..Cs],R,X,F) :- fcand(Cs,R,X,F).

/* Note that bindings made during these tests do not remain afterwards.
   Finally, if the flag is set, possible consequences must be followed up
*/

ripple(F,_,_,_,_) :- var(F), !, trace(no-repercussions,4).
ripple(_,G,Ca,test(_,V,_),X) :- value(Ca,[G]),
   trace(value-is-G-and-trying-repercussions,4),
   filter1(V,X).

/* Filter the set of refs affected */

filter1(V,_) :- var(V), !.
filter1([R,..Rs],X) :- filter(R,X), filter1(Rs,X), !.

/* Appending add assertions to the sentence add list */

add(L,X) :- add(add,L,X).

add(add,L,X) :- add1(X,A), lastof(A,M), addap(L,M), !.

addap([],_) :- !.
addap([L1,..Ls],[L1,..M]) :- !, addap(Ls,M), trace(add-assertion-L1-made,3).
addap(L,[L,..._]) :- trace(add-assertion-L-made,3).

/* Appending test lists to the relevent places
   and following up consequences                    */

test(L,X) :- add(test,L,X).

add(test,L,X) :- addtest(L,X,R), filter1(R,X), !.

addtest([],_,_) :- !.
addtest([L,..Ls],X,R) :- !, at1(L,X,R1), addtest(Ls,X,R), refapp(R1,*,R).
addtest(L,X,R) :- at1(L,X,R).

/* Appending a ref list onto another without duplications
     and avoiding one ref  */

refapp(L,_,_) :- var(L), !.
refapp([I,..Is],I1,I2) :- (I=I1;snoc(I,I2)), !, refapp(Is,I1,I2).

/* Getting a list of refs from a list of assertion arguments */

refsof([],_,_,_) :- !.
refsof([A,..As],L,X,R) :- var(A), findref(A,I,_,_,_,X), !, snoc(I,R),
   refsof(As,L,X,R).
```

```
    (sround(A) ; (setref(A,Ca,det,X),findref(A,I,_,_,_,X),snoc(I,R))),
    !, refsof(As,L,X,R).
refsof([A,..As],L,X,R) :- atomic(A), !, refsof(As,L,X,R).
refsof([T,..As],L,X,R) :- T=..[_,..Al], refsof(Al,L,X,R), refsof(As,L,X,R).

/* Deal with one test assertion, changing ref structures as needed */

at1(L,X,R) :- L=..[F,..A], refsof(A,L,X,R), incorr(R,L,X),
    trace(test-assertion-L-made,3).

incorr(R,L,X) :- var(R), !, add1(X,A), lastof(A,C), L/[C,X].
    /* If a test assertion contains no refs, test if true */
incorr(R,L,X) :- putin(R,L,R,X).

putin(I,_,_,_) :- var(I), !.
putin([I,..Is],L,I1,X) :- findref(_,I,test(T,V,_),_,_,X), lastof(T,[L,..._]),
    refspp(I1,I,V), putin(Is,L,I1,X).

/* FORCE A REFERENCE TO BECOME INSTANTIATED */

instantiate(R,X) :- notvar(R), !.
instantiate(R,X) :-findref(R,_,test(_,V,_),Ca,_,X), !, cand(Ca,R),
    filter1(V,X).

:- end.
```

```
:- entry(/,2).
:- entry(value,2).
:- entry(focvalue,2).
:- entry(ndvalue,2).
:- entry(findref,6).
:- entry(typelt,3).
:- entry(cand,2).
:- entry(setof,2).

:- module(sen,[
find(3), cleanuprefs(1), typelt(3) ]).

:- mode matchl(+,+,+).
:- mode copy(+,-).
:- mode cand(+,?).
:- mode value(+,+).
:- mode seek(+,?,-).
:- mode constlength(+,?).
:- mode ltall(+,+).

 /* HANDLING OF REFERENCES

Sentence datastructures have a reference field, accessed by, eg.
     refs(X,L).
L is a list of indeterminate length, whose entries are of the form
    ref(V,I,TT,Ca,Y).
Here, V is the variable standing for the object referred to,
 I is an unique atom identifier for the reference
 and TT is a term of the form
    test(T,V,Co).
In this, T is an indeterminate list of tests,
         V is a list of identifiers of 'refs' referred to in these
         Co is the context for the tests (that is, the set of
           sentence assertions to be consulted is the add list - Co),
         Ca is a determinate list of candidates (each being of the
           form p(C,F), where C is the name of a definite world
           object and F is a flag. The object is a candidate iff the
           flag is a variable).
         Y is the reference type (det/nondet(Typ)/typ(Nondet))
*/

/* Finding the 'ref' structure corresponding to a particular
   variable or identifier */

findref(R,I,T,C,Y,X) :- atom(I), !, refs(X,L),
   sin(ref(R,I,T,C,Y),L), !.
findref(R,I,T,C,Y,X) :- refs(X,L), sin(ref(R1,I,T,C,Y),L), R==R1, !.

/* Finding the typical element of a nondet ref */

typelt(S,E,X) :- refs(X,L), findref(S,_,_,_,nondet(E),X).

/* Augmenting deduction to make use of the sentence add list. The
   facility provided is slightly stronger than logical deduction.
   The only objects present in deduced assertions are definite
   objects (and variables interpreted in a universally quantified
   sense). Thus the system makes use of the 'rule':

      If p(x1,x2,..xn) holds and
        for i=1,2,..n  yi is xi or a candidate for xi
      Then p(y1,y2,..yn) holds.
*/

triv(O,T)/_ :- var(T), !, O=T.
```

GENERAL ROUTINE FOR REFERENCES

```
L/_ :- call(L).
L/[C,X] :- find(L,C,X).
L/C :- call(L/C).

find(L,C,X) :- L=..[F,..A1], copy(A1,A2), M=..[F,..A2], add1(X,A),
   in(M,A-C), match1(A2,A1,X).

match1([],[],_).
match1([M,..R],[N,..S],X) :- ground(M), !, M=N, match1(R,S,X).
match1([V,..R],[C,..S],X) :- findref(V,_,_,T,_,X),
   !, cand(T,C), match1(R,S,X).
match1([V,..R],[V,..S],X) :- match1(R,S,X).

/* Note that a ref with no candidate list is assumed initially to have
   no candidates */

cand([],_) :- !, fail.
cand([p(C,F),..._],C) :- var(F).
cand([_,..L],C) :- cand(L,C).

/* Converting valid cands to a list or vice versa */

setof([],[]).
setof([p(_,F),..L],C) :- notvar(F), !, setof(L,C).
setof([p(C,_),..L],[C,..Cs]) :- setof(L,Cs).

/* The value of a candidate list is the unique member if there is
   one and a variable if there is >1 member.
   Otherwise the predicate fails  */

value(Ca,[G]) :- !, seek(Ca,_,G).
value(Ca,G) :- nondet(G), !, ndvalue(Ca,G).
value(Ca,G) :- setof(Ca,C), length(G,L1), length(C,L2), compare(L1,L2,C,G).

seek([],V,_) :- var(V), !, fail.
seek([],V,V) :- !.
seek([p(_,F),..L],V,R) :- notvar(F), !, seek(L,V,R).
seek([p(C,_),..L],C,R) :- !, seek(L,C,R).
seek(_,_,_).

focvalue(Ca,[G]) :- !, value(Ca,[G]).
focvalue(Ca,G) :- nondet(G), !, value(Ca,[G]),
   (ground(G);setlength(Ca,G)), !.
focvalue(Ca,G) :- value(Ca,[G]).

setlength(Ca,G) :- constlength(Ca,L), !, listlength(L,G).
setlength(Ca,G) :- minlength(Ca,L), !, ndlength(G,L).

constlength([p(C,_),..Cs],L) :- length(C,L), constlength(Cs,L).
constlength([],_).

minlength(Ca,L) :- cand(Ca,C), length(C,L), ltall(Ca,L).

ltall([p(C,_),..Cs],L) :- length(C,L1), L=<L1, ltall(Cs,L).
ltall([],_).

ndvalue(Ca,R) :- setof(Ca,C), ndlength(R,L1), length(C,L2),
   compare(L1,L2,C,R).

compare(L1,L2,_,_) :- L1<L2, !.
compare(L,L,R,R) :- !.

ndlength(X,0) :- var(X), !.
ndlength([_,..L],N) :- ndlength(L,N1), N is N1+1.
```

```
copy([_,.A],[_,.B]) :- copy(A,B).

/* CLEANING UP REFERENCES */

cleanuprefs(X) :- refs(X,L), cleardets(L,X), clearndets(L,X).

cleardets(L,_) :- var(L), !.
cleardets([ref(R,_,test(_,V,_),Ca,det),.Rs],X) :- var(R), cand(Ca,R), !,
    filter1(V,X), cleardets(Rs,X).
cleardets([_,.Rs],X) :- cleardets(Rs,X).

clearndets(L,_) :- var(L), !.
clearndets([ref(R,_,_,Ca,nondet(E)),.Rs],X) :- nondet(R), !, setof(Ca,R),
    !, findref(E,_,test(_,V,_),_,_,X), filter1(V,X), clearndets(Rs,X).
clearndets([_,.Rs],X) :- clearndets(Rs,X).

:- end.
```

TOP-LEVEL PARSING ROUTINES

```
/*  VERB GROUP  */

verbsr(mainverb(V,[N,M,T,A]),X,W1,W2) :- ishv(A1,X), closehv(X), !,
   vg(V,T,A,M,N,[A1,..W1],W2).
verbsr(mainverb(V,[N,M,T,A]),X,W1,W2) :- vg(V,T,A,M,N,W1,W2).

vg(V,fut,A,M,N) --> [will], inf(V,A,M).
vg(V,pres,A,M,sing) --> [is], papr(V,A,M).
vg(V,pres,A,M,plur) --> [are], papr(V,A,M).
vg(V,past,A,M,sing) --> [was], papr(V,A,M).
vg(V,past,A,M,plur) --> [were], papr(V,A,M).
vg(V,pres,compl,M,sing) --> [has], pa(V,M).
vg(V,pres,compl,M,plur) --> [have], pa(V,M).
vg(V,past,compl,M,N) --> [had], pa(V,M).
vg(V,pres,inst,actv,sing) --> [does,V], {verb(V)}.
vg(V,pres,inst,actv,plur) --> [do,V], {verb(V)}.
vg(V,past,inst,actv,N) --> [did,V], {verb(V)}.
vg(V,pres,inst,actv,N) --> [P], {pres(P,V,N)}.
vg(V,past,inst,actv,N) --> [P], {past(P,V)}.

papr(V,cing,passv) --> [being,P], {pastpart(P,V)}.
papr(V,inst,passv) --> [P], {pastpart(P,V)}.
papr(V,cing,actv) --> [P], {prespart(P,V)}.

pa(V,passv) --> [been,P], {pastpart(P,V)}.
pa(V,actv) --> [P], {pastpart(P,V)}.

inf(V,compl,M) --> [have], pa(V,M).
inf(V,A,M) --> [be], papr(V,A,M).
inf(V,inst,actv) --> [V], {verb(V)}.

:- end.
```

```
/*  MAIN PARSING FUNCTIONS  */

parse(O,A,F,X) --> continue(O,A,F,X,noass).
constituent(O,A,F,X) --> continue(O,A,F,X,ass).

/* BOTTOM-UP ANALYSIS */

continue(O,A,F,X,B)                          --> ([a];[an]), {presubst(B,O,X)},
   indefsp(O,A,F,X).
continue(O,A,F,X,B)                          --> [the], {presubst(B,O,X)},
   defsp(O,A,F,X).
continue(physobj:[G],nolist,_,X,B)           --> [the,former], !,
   {presubst(B,physobj:[G],X), forlat(X,[G,_])}.
continue(physobj:[G],nolist,_,X,B)           --> [the,latter], !,
   {presubst(B,physobj:[G],X), forlat(X,[_,G])}.
continue(defmeas:[G],nolist,F,X,B)           --> [what],
   {prernf(B,defmeas:[G],X), sensym(unk,G),
   add(add,sousht(G),X)}.
continue(O,A,F,X,B,[W,.,W1],W2)              :- (W=which;W=what),
   prernf(B,O,X), indefsp(O,A,F,X,[which,.,W1],W2).
continue(measpair:[M],nolist,F,X,B)          --> [how,much],
   {prernf(B,measpair:[M],X), sensym(meas,M),
   add(add,sousht(M),X)}.
continue(physobj:[G],nolist,F,X,B)           --> [it], {presubst(B,physobj:[G],X),
   dofoc([G],X),
   mentioned([G],X)}.
continue(physobj:Gs,nolist,F,X,B)--> ([they];[them]),
   {Gs=[_,_,.,_], presubst(B,physobj:Gs,X), dofoc(Gs,X),
   mentioned(Gs,X)}.
continue(O,A,F,X,B)                          --> [its],
   {presubst(B,O,X), dofoc([G],X), mentioned([G],X)},
    postphysobj(poss,[G],O,A,test,F,X).
continue(O,A,F,X,B)                          --> [their],
   {presubst(B,O,X), Gs=[_,_,.,_], dofoc(Gs,X),
   mentioned(Gs,X)}, postphysobj(poss,Gs,O,A,test,F,X).
continue(indefmeas:Js,A,F,X,B)  --> [D1],
   {dim(D1,D,_), presubst(B,indefmeas:Js,X)}, qual(Ms,F,X),
   {sivesl(Ms,breakdown(D),Is,F,X)}, and(indefmeas:Is,indefmeas:Js,A,F,X).
continue(indefmeas:I,A,F,X,B)   --> pmadjs(F,G,D,X), {presubst(B,indefmeas:I,
   qual(Ms,F,X),
   {sivesl(Ms,breakdown(D),Is,F,X)}, and(indefmeas:Is,indefmeas:I,A,F,X).
continue(ajp:A,nolist,F,X,B)    --> [A],
   {(cadj(A);padj(A)), presubst(B,ajp:A,X)}.
continue(madv:I,nolist,F,X,B)   --> [with], findsubst(indefmeas:I,F,X).
continue(timeadv,nolist,F,X,B)  --> timeadv(X).
continue(pp(P,physobj:G),nolist,F,X,B)--> [P],
   {prep(P,_,_), prepp(B,P,physobj:G,X)},
   findsubst(physobj:G,F,X).
continue(binder:st,nolist,F,X,_)             --> [so,that].
continue(O,A,F,X,B,[Nu,.,L],M)  :- num(Nu,N), presubst(B,O,X),
   indefsp(O,A,F,X,[N,.,L],M).
continue(O,A,F,X,B)                          --> [Nu,U],
   {nnum(Nu,N,F,X), unit(U,U1_)},
   afternu(O,N,U1,A,F,X,B).
continue(Y:[N],nolist,F,X,B)                 --> [N],
   {name(N), presubst(B,Y:[N],X), checkname(N,Y,test,X)}.
continue(prt:P,nolist,F,X,B)    --> [P], {prt(P)}.
continue(aux:A,nolist,F,X,B)    --> [A], {aux(A), preaux(B,A,X)}.
continue(verbsr:I,nolist,F,X,B) --> verbsr(I,X), {premv(B,I,X)}.
continue(physobj:G,A,F,X,B)     --> [C], {class(C,C1,[plur,noposs]),
   presubst(B,physobj:G,X), plur(G1,plur), startref(G1,C1,F,X),
   mentioned(G1,X)},
   postphysobj(noposs,G1,physobj:G,A,F,F,X).
```

```
continue(binder!'implicit,comma',_,_,_,_) --> [].

forlat(X,G) :- findconstr(forlat(G),X), !.
forlat(X,G) :- dofoc(G,X), putconstr(forlat(G),X). /* ******!!!!! */

findsubst(O,F,X) --> parse(O,_,F,X)! {isrnf(O,X),closernf(X),
    trace(rnf-O-absorbed,2)}.

afternu(pp(P,[G,[N,U]]),N,U,nolist,F,X,B) --> [P],
    {prep(P,_,_), prepp(B,P,[G,[N,U]],X), G=physobJ!G1},
    findsubst(G,F,X).
afternu(measpair![[N,U]],N,U,nolist,F,X,B) --> {presubst(B,measpair![[N,U]],X
    [].

and(Y!G1,Y!G3,list,F,X) --> [and], parse(Y!G2,nolist,F,X),
    {sapp(G1,G2,G3), mentphys(Y,G3,X)}.
and(Y!G1,Y!G3,list,F,X) --> parse(Y!G2,sublist,F,X), {sapp(G1,G2,G3),
    mentphys(Y,G3,X)}.
and(O,O,nolist,_,_) --> [], !.
and(Y!G1,Y!G3,sublist,F,X) --> [and], parse(Y!G2,nolist,F,X),
    {sapp(G1,G2,G3)}.
and(Y!G1,Y!G3,sublist,F,X) --> parse(Y!G2,sublist,F,X), {sapp(G1,G2,G3)}.

/* Mentionins conJoined physical obJects */

mentphys(physobJ,G,X) :- !, mentioned(G,X).
mentphys(_,_,_).

:- end.
```

```
/* THIS FILE REPLACES A SEMANTIC COMPONENT */

/* Main interface functions */

makeprop(D,G,F,X,M) :- per(X,T), L=..[D,G,M,T], dofn(F,D,X,[M],L).
makepart(P,G,F,X,M) :- plits(P,G,M,L), dofn(F,P,X,M,L).
makeadJ(A,G,F,X) :- padJ(A), !, padJ(G,A,F,X).
makeadJ(A,G,F,X) :- cadJ(A), !, cadJ(G,A,F,X).

/* Replacements for semantic routines */

padJ(G,fine,F,X) :- makeprop(mass,G,F,X,zero).
padJ(G,roush,F,X) :- sensym(mu,Mu), add(F,[coeff(G,Mu),given(Mu)],X).
padJ(G,smooth,F,X) :- add(F,coeff(G,zero),X).
padJ(G,fixed,F,X) :- add(F,fixed(G),X).
padJ(G,stationary,F,X) :- makeprop(velocity,G,F,X,zero).
padJ(G,light,F,X) :- makeprop(mass,G,F,X,zero).

cadJ(G,constant,F,X) :- add(F,invar(G),X).
cadJ(zero,zero,_,_).

dofn(test,_,X,_,L) :- !, add(test,L,X).
dofn(add,_,X,_,L) :- add(test,L,X), !.
dofn(add,D,X,M,L) :- (sround(M);sensymlist(D,M)), !, add(add,L,X).

plits(end,G,[G1,G2],[end(G,G1,left),end(G,G2,right)]).

/* Dummy verb definitions */

hans(_,X) :- prsent(X).
with(_,X) :- prsent(X).
over(_,X) :- prsent(X).
remain(_,X) :- prsent(X).
carry(_,X) :- prsent(X).
place(_,X) :- prsent(X).
at(_,X) :- prsent(X).

connect(_,X) :- prsent(X).
pass(_,X) :- prsent(X).
find(_,X) :- prsent(X).

:- end.
```

```
/*  REGULAR VERBS  */

pastpart(P,V) :- name(P,X), (append(Z,"ed",X);append(Z,"en",X)), verbdb(Z,V).

prespart(P,V) :- name(P,X), append(Z,"ing",X), verbdb(Z,V).

past(P,V) :- name(P,X), append(Z,"ed",X), verbdb(Z,V).

pres(V,V,plur) :- verb(V), !.  /*  Assuming no 1st and 2nd person singular */
pres(P,V,sing) :- name(P,X), append(Z,"s",X), verbe(Z,V).

   verbdb(X,V) :- name(V,X), verb(V), !.
   verbdb(Z,V) :- append(Y,[C,C],Z), conson(C), append(Y,[C],X), name(V,X),
      verb(V).
   verbdb(Z,V) :- append(Z,"e",Y), name(V,Y), verb(V).

   verbe(X,V) :- name(V,X), verb(V), !.
   verbe(Z,V) :- append(Y,"e",Z), name(V,Y), verb(V).

/*  REGULAR NOUNS  */

class(C,C,[sing,noposs]) :- class(C), !.
class(C1,C2,[S,P]) :- name(C1,X1), ending(X1,X2,S,P), name(C2,X2), class(C2).

   ending("'s",[],sing,poss) :- !.
   ending("s",[],plur,noposs) :- !.
   ending("s'",[],plur,poss) .
   ending("'", [], sing,poss) :- !.
   ending([L,.Ls],[L,.Ms],S,P) :- ending(Ls,Ms,S,P).

/*  DIMENSIONS  */

dim(D,D,sing) :- dim(D,_).
dim(D1,D2,plur) :- name(D1,X1), append(X2,"s",X1), name(D2,X2), dim(D2,_).

/*  VERB INFORMATION  */

aux(is). aux(are). aux(do). aux(does). aux(have). aux(has).
aux(had). aux(was). aux(were). aux(did). aux(will).

verb(attach). verb(connect). verb(hang). verb(pass).
verb(place). verb(remain). verb(rest). verb(show).
verb(suspend). verb(find).

pres(is,be,sing).
pres(are,be,plur).
pres(has,have,sing).
past(was,be).
past(were,be).
past(had,have).
past(hung,hang).
pastpart(hung,hang).
pastpart(shown,show).

/*  'NOUN' INFORMATION  */

class(pulley). class(string). class(weight). class(mass).
class(table). class(particle). class(man). class(rope).
class(peg).

class(men,man,[plur,noposs]).
class('men''s',man,[plur,poss]).
class(masses,mass,[plur,noposs]).
```

```
dim(weight,[1,0,0]).
dim(mass,[1,0,0]).
dim(tension,[1,1,-2]).
dim(acceleration,[0,1,-2]).
dim(length,[0,1,0]).
dim(masses,mass,plur).

possdim(_,[1,0,0],mass).

/* Part descriptions */

part(end,one) --> [end].
part(end,all) --> [ends].

/* NAMES - These may be of the form
      <letter>, <letter><digit> or <letter><prime(s)>  */

name(N) :- name(N,[L,,,R]), letter(L), subscript(R).

subscript([]). subscript("'"). subscript("''").
subscript([R]) :- digit(R).

/*  'ADJECTIVE' INFORMATION   */

dadJ(heavy,mass).

padJ(fine).  padJ(rough).  padJ(smooth). padJ(fixed).
padJ(stationary). padJ(light).

cadJ(constant). cadJ(zero).

/*  MISCELLANEOUS  */

num(N,N) :- integer(N), !.
num(two,2).
num(three,3).

/* More general number, including suitable names */

nnum(Nu,N,_,_) :- num(Nu,N), !.
nnum(N,N,F,X) :- name(N), checkname(N,number,F,X).

unit(lbs,[1,0,0]).
unit(pounds,[1,0,0]).
unit(stones,[1,0,0]).
unit(gms,[1,0,0]).

unit(U,U1,Dim) :- name(U,N), append(N,"s",N1), name(U1,N1), unit(U1,Dim).

relpron(which).
relpron(who).
relpron(whom).

binder('.'). binder(','). binder(';'). binder(st).
binder(if).

/*  PREPOSITION INFORMATION

   Prepositions are marked:
      s - if can modify noun groups (locative)
      d - if cannot modify noun groups (directional)

   The marker 'loc' or 'noloc' determines whether
   it is conveying locational information          */
```

```
prep(at,s,loc).
prep(in,s,loc). prep(into,d,loc). prep(with,s,noloc).
prep(over,s,loc). prep(on,s,loc). prep(to,d,loc).
prep(from,d,loc). prep(by,d,noloc).

/*  PREPOSITIONS FOR VERBS  */

prepfor(attach,loc).
prepfor(connect,to).
prepfor(hang,loc).
prepfor(pass,over). prepfor(pass,under).
prepfor(place,loc).

prepfor(V,P) :- prep(P,_,loc), prepfor(V,loc).

/* 'prepfor' represents a crude analysis of which prepositions
   are expected by which verbs                                  */

prepfor(_,by).

:- end.
```

```
/* AFTER 'the'  */

defsp(defmeas:M,A,F,X) --> [D1], {dim(D1,D,S), plur(G,S)}, ofobJ(G,F,X),
   {giveall(G,makeprop(D),M1,add,X)}, and(defmeas:M1,defmeas:M,A,F,X).
defsp(physobJ:G,A,F,X) --> part(P,S), ofobJ([G1],F,X),
   postpart(P,S,[G1],G,A,F,X).
defsp(O,A,F,X) --> physobJ(O,test,F,X).

   ofobJ(G,F,X) --> ([of];[on];[in]), parse(physobJ:G,_,F,X).
   ofobJ(_,_,_) --> [].

/* AFTER 'a''an'  */

indefsp(indefmeas:M,A,F,X) --> pmadJs(F,M1,D,X),
   optqual(D,F,M1,X), and(indefmeas:M1,indefmeas:M,A,F,X).
indefsp(physobJ:G,A,F,X) --> part(P,S), ofobJ([G1],F,X),
   postpart(P,S,[G1],G,A,F,X).
indefsp(physobJ:G,A,F,X) --> physobJ(physobJ:G,F,F,X).

/* AFTER 'one' */

onesp(physobJ:G,A,F,X) --> part(P,_), ofobJ([G1],F,X),
   postpart(P,one,[G1],G,A,F,X).
onesp(physobJ:G,A,F,X) --> scanclass(C,_,H,W,F,X), {plur(G1,plur),
   startref(G1,C,test,X), ppadJs(W,F,G1,C,X), select(one,G1,G2,X)},
   postphysobJ(H,G2,physobJ:G,A,test,F,X).

/* NOUN PHRASE STARTING WITH PHYSOBJ DECSRIPTION */

physobJ(O,F,F1,X) --> num(G), scanclass(C,G,H,W,F,X), names(G,F,X),
   {startref(G,C,F,X), mentioned(G,X), ppadJs(W,G,C,F,X)},
   postphysobJ(H,G,O,A,F,F1,X).

   num(G) --> [Nu], {num(Nu,N), listlength(N,G)}.
   num(_) --> [].

   names([N,.,Ns],F,X) --> [N], {name(N), checkname(N,physobJ,F,X)},
      morenames(Ns,F,X).
   names(_,_,_) --> [].

   morenames([N],F,X) --> [and,N], {name(N), checkname(N,physobJ,F,X)}.
   morenames([N,.,Ns],F,X) --> [N], {name(N), checkname(N,physbbJ,F,X)},
      morenames(Ns,F,X).

   startref(G,C,add,X) :- !, (sround(G);sensymlist(C,G)),
      distrib(G,[C],nmeans,add,X).
   startref(G,_,_,_) :- sround(G), !.
   startref(G,C,_,X) :- defnp(G,C,X).

   optqual(D,F,G,X) --> qual(Ms,F,X), {giveall(Ms,breakdown(D),G,F,X)}.
   optqual(_,_,_,_) --> [].

   qual(Ms,F,X) --> ([equal,to];[of];[]), mp1(Ms,F,X).

/*  AFTER A PHYSICAL OBJECT  */

postphysobJ(poss,G,physobJ:G2,A,F,F1,X) --> part(P,S),
   postpart(P,S,G,G2,A,F1,X).
postphysobJ(poss,G,defmeas:M2,A,F,F1,X) --> [D1], {dim(D1,D,_),
   giveall(G,makeprop(D),M1,add,X)}, and(defmeas:M1,defmeas:M2,A,F1,X).
postphysobJ(noposs,G,physobJ:G1,A,F,F1,X) --> ppadJs(F,physobJ:G,X),
   and(physobJ:G,physobJ:G1,A,F1,X).
```

```
postpart(P,S,[G],G3,A,F,X) --> {makepart(P,G,F,X,G1), select(S,G1,G2,X),
   mentioned(G2,X)}, and(physobj:G2,physobj:G3,A,F,X).

/*  ADJECTIVES & MODIFIERS

   pm - measurements  ) Before noun
   pp - physobjs      )

   ap - physobjs      ) After noun
                                        */

/* Before physical objects */

scanclass(C1,G,H,[],F,X) --> [C], {class(C,C1,[S,H]), plur(G,S)}.
scanclass(C,G,H,[padJ(A),,,As],F,X) --> [A], {padJ(A)},
   scanclass(C,G,H,As,F,X).
scanclass(C,G,H,[dadJ(N,U,T,D),,,As],F,X) --> [N1,U1], {nnum(N1,N,F,X),
   unit(U,U1,T)}, ((A],{dadJ(A,D)}):[]), scanclass(C,G,H,As,F,X).

padjs([],_,_,_,_) :- !.
padjs([padJ(A),,,As],G,C,F,X) :- !, distrib(G,[A],makeadj,F,X),
   padjs(As,G,C,F,X).
padjs([dadJ(N,U,T,D),,,As],G,C,F,X) :- (atom(D):possdim(C,T,D)), !,
   distrib(G,[[N,U]],adddim(D),F,X).

/* After the noun */

apadjs(P,S,X) --> postmod(P,S,X,_,I), mmods(I,P,S,X).
apadjs(_,_,_) --> [].

   mmods(relc,_,_,_) --> [], !.
   mmods(I,P,S,X) --> apadjs(P,S,X).

postmod(P,O,X,A,I) --> mp1(Ms,P,X), anu(P,O,X,A,I,Ms).

   anu(P,Y:G,X,A,dmeas,Ms) --> [in,D1], {dim(D1,D,_),
      distrib(G,Ms,adddim(D),P,X)},
      list(P,Y:G,A,X,dmeas).
   anu(P,Y:G,X,A,dmeas,Ms) --> [Ad], {dadJ(Ad,D),
      distrib(G,Ms,adddim(D),P,X)},
      list(P,Y:G,A,X,dmeas).
   anu(P,O,X,nolist,relc,[[N,U]]) --> [Q,R], {prep(Q,_,_), relpron(R),
      matchsent(X,T), setpp(Q,[O,[N,U]],T)}, subsentence(P,X,T).
   anu(P,O,X,nolist,pp,[[N,U]]) --> [Q], {prep(Q,s,_)},
      parse(physobj:T,_,P,X),
      {dopp(Q,O,physobj:T,P,X)}.

mp1(L,F,X) --> mp1(L,M,_,F,X), (moremp(M,F,X):{M=[]}).

   moremp(M,F,X) --> [and], !, mp1(M,[],_,F,X).
   moremp(M,F,X) --> mp1(M,M1,_,F,X), moremp(M1,F,X).

   mp1([[N,U],,,M1],M2,U,F,X) --> [Nu], {number(Nu,N,F,X,G)},
      (mp2(M1,M2,U,F,X,G):{M1=M2}), punit(G,U).

   mp2([[N,U],,,M],M,U,F,X,G) --> [and,Nu], !, {number(Nu,N,F,X,G)}.
   mp2([[N,U],,,M1],M2,U,F,X,G) --> [Nu], {number(Nu,N,F,X,G)},
      mp2(M1,M2,U,F,X,G).

   number(Nu,N,_,_,need) :- num(Nu,N), !.
   number(N,N,F,X,_) :- name(N), checkname(N,number,F,X), !.

   punit(_,U1) --> [U], {unit(U,U1,_)}.
   punit(G,*) --> [], {var(G)}.
```

```
postmod(P,Z:S,X,nolist,imeas) --> [with], parse(indefmeas:M,_,P,X),
   {distrib(S,M,nfod,P,X)}.

postmod(P,Z:S,X,nolist,mp) --> [of],
   ([D1],{dim(D1,D,_)}){possdim(_,S,D)}), !, mp1(Ms,P,X),
   {distrib(S,Ms,adddim(D),P,X)}.
postmod(P,Z:S,X,nolist,relc) --> [Q,R], {prep(Q,_,_), relpron(R),
   matchsent(X,T), setpp(Q,S,T)}, subsentence(P,X,T).
postmod(P,O,X,nolist,pp) --> [Q], {prep(Q,s,_)},
   parse(physobJ:T,_,P,X), {dopp(Q,O,physobJ:T,P,X)}.
postmod(P,S,X,nolist,relc) --> ([who];[which];[whom]),
   {matchsent(X,T), setrnf(S,T)}, subsentence(P,X,T).
postmod(P,S,X,nolist,relc) --> [that], {matchsent(X,T), setrnf(S,T)},
   subsentence(test,X,T).
postmod(P,Z:S,X,nolist,relc) --> [whose],
   postphysobJ(poss,S,T,nolist,P,_,X),
   {matchsent(X,U), setrnf(T,U)}, subsentence(P,X,U).
postmod(P,Z:S,X,nolist,relc) --> [T], {prespart(T,V), matchsent(X,U),
   setsubJ(Z:S,U),
   setmv(mainverb(V,[N,actv,W,cins]),U),
   currview(W,X)}, subsentence(P,X,U).
postmod(P,Z:S,X,nolist,relc) --> [T], {pastpart(T,V), matchsent(X,U),
   setsubJ(Z:S,U),
   setmv(mainverb(V,[N,passv,W,inst]),U),
   currview(W,X)}, subsentence(P,X,U).

list(F,O,list,X,I) --> [and], postmod(F,O,X,nolist,I).
list(F,O,list,X,I) --> postmod(F,O,X,list,I).
list(F,O,nolist,X,I) --> [].

/* Before measurements */

pmadJs(F,I,D,X) --> scandim(G,D,W), {pma1(F,G,X,W,[]), objtoi(G,D,I)}.

   scandim(G,D,[]) --> [D1], {dim(D1,D,S), plur(G,S)}.
   scandim(G,D,[A,.,Ws]) --> [A], {cadJ(A)}, scandim(G,D,Ws).

pma1(_,_,_,[],[]) :- !.
pma1(F,G,X) --> [A], {per(X,T), distrib(G,[A],makeadJ,F,X)},
      pma1(F,G,X).

objtoi(G,_,_) :- var(G), !.
objtoi([],_,[]).
objtoi([G,.,Gs],D,[D#G,.,Is]) :- objtoi(Gs,D,Is).

/*  DISTRIBUTION OF PROPERTIES  */

breakdown(D,[N,U],D#M,F,X) :- !, unitass(M,U,L),
   (atom(D);(unit(U,T),possdim(_,T,D))), !,
   add(F,[measure(M,N),.,L],X), makesiven(M,F,X).
breakdown(D,M,D#M,_,_).

   unitass(M,*,[]) :- !.
   unitass(M,U,[unit(M,U)]).

nfod(G,D#M,F,X) :- makeprop(D,G,F,X,M).

adddim(D,G,Mp,F,X) :- makeprop(D,G,F,X,M), breakdown(D,Mp,D#M,F,X).

makesiven(_,test,_) :- !. /* Redundent test */
makesiven(M,add,X) :- !, putconstr(s(M,_),X).

makesousht(G,X) :- findconstr(s(G,out),X), !.
makesousht(G,X) :- putconstr(s(G,out),X).
```

```
nmeans(G,C,F,X) :- ideal(C,I), add(F,[isa(I,G),hasname(C,G)],X).

ideal(mass,particle) :- !.
ideal(weisht,particle) :- !.
ideal(C,C).

/* Checking of types of names */

checkname(N,Y,add,X) :- findconstr(n(N,Y1),X), !, Y=Y1.
checkname(N,Y,add,X) :- !, putconstr(n(N,Y),X).
checkname(N,Y,test,X) :- findconstr(n(N,Y),X).

:- end.
```

```
:- op(300,fx,'#').
:- op(300,xfy,[#,:]).
:- tlim(4).

/*  INITIAL COMMAND  */

continue :- recorded(s(N,C,M,T),P), !, readsent(S), nl,
   newsentence(sent(r(_,_,_,_,_,_),d(_,_,C,M),T,p(N,N1,0)),S),
   erase(P), record(s(N1,C,M,T)).

start :- gensym(period,T), X=sent(r(_,_,_,_,_,_),d(_,_,C,M),T,p(start,N,0)),
   readsent(S), nl,
   add(period(T),X), newsentence(X,S), record(s(N,C,M,T)).

finish :- recorded(s(_,C,_,_),P), !, display('% equation solving information'
   nl, mksivens(C), nl, display('end of paragraph'), nl.
finish :- display('no input'), nl.

newsentence(X,S) :- sentence(add,X,S,[]), cleanuprefs(X), doassertions(X), !.

/*  SENTENCE PARSING  */

sentence(F,X) --> {gensym(sentence,N), trace(starting-N,2), sname(X,N)},
   readcons(F,X).

readcons(F,X) --> constituent(O,_,F,X), {trace(constituent-O-found,2)},
   action(O,F,X).

action(binder:B,F,X) --> !, binder(B,F,X).
action(O,F,X) --> readcons(F,X).

   binder(B,F,X) --> {name(B,[L]), final(L)}, !, final(F,X).
   binder(';',F,X) --> !, semicol(F,X).
   binder(',',F,X) --> !, comma(F,X).
   binder(if,F,X) --> {matchsent(X,Y)}, subsentence(F,X,Y), sentence(F,X).
   binder('implicit.comma',F,X) --> {level(X,N), diff(N,0),
      endofsent(F,X)}, !.
   binder('implicit.comma',F,X) --> !, {trace('backtracking...',2), fail}.
   binder(_,F,X) --> {endofsent(F,X), matchsent(X,Y)}, succsentence(F,X,Y).

   final(F,X) --> {level(X,0), !, endofsent(F,X)}.
   final(F,X), ['.'] --> {endofsent(F,X)}.

   semicol(F,X) --> {level(X,0), !, endofsent(F,X), matchsent(X,Y)},
      succsentence(F,X,Y).
   semicol(F,X), [';'] --> {endofsent(F,X)}.

   comma(F,X) --> {level(X,0), !, beform(X,P), sethv(Y,P),
      endofsent(F,X), matchsent(X,Y)}, succsentence(F,X,Y).
   comma(F,X) --> {endofsent(F,X)}.

   beform(X,bepres) :- currview(pres,X), !.
   beform(X,bepast) :- currview(past,X).

/*  INTER-SENTENCE RELATIONSHIPS  */

subsentence(F,X,Y) --> {level(X,N), N1 is N+1,
   level(Y,N1), sname(X,M), last(Y,M),
   trace(doing-embedded-sentence,2)}, sentence(F,Y).
succsentence(F,X,Y) --> {level(X,N), level(Y,N), sname(X,M), last(Y,M),
   trace(doing-successor-sentence,2)}, sentence(F,Y).

matchsent(sent(_,D,T,_),sent(r(_,_,_,_,_,_),D,T,p(_,_,_))).
```

```
setsubj(X) :- issubj(_,X), !.
setsubj(X) :- isrnf(O,X), !, closernf(X), setsubj(O,X), trace(subject-is-O,2).
setsubj(X) :- thnot(pp(_,_,X)), !, ismvi(mainverb(_,[plur,_,pres,inst]),X),
   setsubj(you,X),
   trace(imperative-sentence,2).
setsubj(_).   /* SUBJECT NOT YET AVAILABLE */

presubst(noass,_,_) :- !.
presubst(_,O,X) :- ismv(_,X), !, saftermv(O,X).
presubst(_,O,X) :- ishv(_,X), !, saferhv(O,X).
presubst(_,_,X) :- issubj(_,X), !, fail.
presubst(_,O,X) :- setsubj(O,X), trace(lookins-for-subject,2).

saftermv(O,X) :- isobj(O1,X), !, setobj(O1,2,X), addobj(O,1,X),
   trace(object2-is-O1-',-lookins-for-object1,2).
saftermv(O,X) :- issubj(_,X), !, setobj(O,_,X), trace(lookins-for-object,2).
saftermv(O,X) :- setsubj(O,X), trace(lookins-for-subject,2).

saferhv(O,X) :- issubj(_,X), !, verbsr(I,X,[],[]), setmv(I,X),
   trace(main-verb-is-I,2), setobj(O,_,X), trace(lookins-for-object,2).
saferhv(O,X) :- setsubj(O,X), trace(lookins-for-subject,2).

prepp(noass,_,_,_) :- !.
prepp(_,P,O,X) :- dprep(P,X), setpp(P,O,X),
   trace(parsins-prepositional-phrase,2).

preaux(noass,_,_) :- !.
preaux(_,_,X) :- (ishv(_,X);ismv(_,X)), !, fail.
preaux(_,A,X) :- sethv(A,X), trace(holdins-aux-A,2).

premv(noass,_,_) :- !.
premv(_,_,X) :- ismv(_,X), !, fail.
premv(_,I,X) :- setmv(I,X), setsubj(X), trace(lookins-for-mainverb,2).

prernf(noass,_,_) :- !.
prernf(_,_,X) :- isrnf(_,X), !, fail.
prernf(_,O,X) :- setrnf(O,X), trace(lookins-for-np-to-hold,2).

/* Acceptability of prepositions */

dprep(P,X) :- ismv(V,X), !, prepfor(V,P).
dprep(_,_).

/*  FINAL CLEARUP OF ROLES  */

clearuproles(X) :- setmv(X), setobjs(X).

setmv(X) :- ismv(_,X), !.
setmv(X) :- ishv(V,X), verbsr(I,X,[],[]), setmv(I,X),
   trace(main-verb-is-I,2), setsubj(X).

setobjs(X) :- isrnf(G,X), !, closernf(X), placernf(G,X).
setobjs(X) :- isobj(G,X), !, setobj(G,1,X).
setobjs(_).

placernf(G,X) :- isobj(G1,X), !, setobj(G1,2,X), addobj(G,1,X).
placernf(G,X) :- setobj(G,1,X).

/* END OF SENTENCE */

endofsent(F,X) :- per(X,T), (sround(T);sensym(period,T)), !, clearuproles(X),
   ismv(V,X), L=..[V,F,X], ttflas(N), (N<2;prsent(X)), !, L.

/* DEALING WITH GIVENS & UNKNOWNS */
```

```
mksivens(C) :- var(C), !.
mksivens([s(M,F)|_,C]) :- !, sivorsou(F,M), mksivens(C).
mksivens([_|_,C]) :- mksivens(C).

sivorsou(F,M) :- var(F), !, write(siven(M)), nl, assert(siven(M)).
sivorsou(_,M) :- write(sousht(M)), nl, assert(sousht(M)).

/* Prepositional phrases */

dopp(P,S,O,F,X) :- trace(finishing-prepositional-phrase,2), matchsent(X,Y),
    setsubj(S,Y), setobj(O,1,Y), setmv(mainverb(P,[_,actv,_,cins]),Y),
    endofsent(F,Y).

/* Making final assertions */

assertall(A) :- var(A), !.
assertall([A|_,As]) :- assert(A), assertall(As).

:- end.
```

Isa hierarchy Unit assertions : isa (X, Y) — Y is an X

For particular and general objects

isa1 — Transitive closure of isa

Parts Unit assertions : part (X, Y) — X names a part of Y

For general objects only

An object's parts include those of its possible shapes and descriptions.

Shape shape (X, Y) — X has the shape Y

General objects only

Description description (X, Y) — X can be described as Y

General objects only

Properties hasprop (X, Y) — X has property Y

Unit assertions for general objects

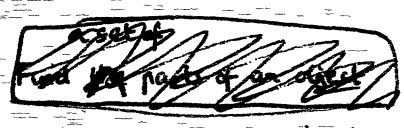
Inference rule (using isa1) for particular

~~Satisfies (X, Y) :- (X=Y); (shape (Y, X))~~

satisfies (X, Y) :- X=Y ; shape (Y, X) ; isa1 (X, Y) ; hasprop (Y, X) .

Makepart test haspart

Testpart or addpart



Find a particular set of parts of an object

Makeprop Test if exists

Otherwise test hasprop, then add

Find a particular property of an object

Makeadj Convert to <property, value>

Makeprop

Apply an adjective to an object

Verbs — Constraints (things to be "satisfied")

are associated with ⟨verb, case⟩ and ⟨verbtype, case⟩ pairs.

General procedure (?): — Get sentence constituents by using sentence roles

Ascertain cases & check constraints

Produce assertions for meaning

Check systems

System check — Verb or verb type gives a tag — typecontact

Look for objects with this tag which

have systemparts

or satisfy 'isa system' — gets "pulley"

? —
- Finds existing system members
- and adds new ones
- If succeeds in filling all slots, cue system

Worked example p18 Snell & Kelmes

```
:- 'NOLC'.

/*TRAIN.PRB*/
/*TRAIN PROBLEM*/
/*ALAN BUNDY APRIL 1978*/

STEADY :- CUE(TIMESYS(EPISODE,DEPARTURE,ARRIVAL)),
  CUE(TIMESYS(PERIOD1,DEPARTURE,CHANGE1)),
  CUE(TIMESYS(PERIOD2,CHANGE1,CHANGE2)),
  CUE(TIMESYS(PERIOD3,CHANGE2,ARRIVAL)),
  CHECKLIST(PASSERTA,[
    PARTITION(EPISODE,[PERIOD1,PERIOD2,PERIOD3]),
    DURATION(PERIOD1,TQ1), DISTANCE(TRAIN,DQ1,PERIOD1),
    DURATION(PERIOD2,TQ2), DISTANCE(TRAIN,DQ2,PERIOD2),
    DURATION(PERIOD3,TQ3), DISTANCE(TRAIN,DQ3,PERIOD3),
    DURATION(EPISODE,TQ0), DISTANCE(TRAIN,DQ0,EPISODE),
    VEL(TRAIN,ZERO,0,DEPARTURE),ACCEL(TRAIN,AQ1,0,PERIOD1),
    VEL(TRAIN,ZERO,0,ARRIVAL), ACCEL(TRAIN,ZERO,0,PERIOD2),
    VEL(TRAIN,VQ,0,PERIOD2), ACCEL(TRAIN,AQ3,0,PERIOD3),
    MEASURE(TQ1,T1), UNIT(TQ1,MINS),
    MEASURE(TQ2,T2), UNIT(TQ2,MINS),
    MEASURE(TQ3,T3), UNIT(TQ3,MINS),
    MEASURE(TQ0,T0), UNIT(TQ0,MINS),
    MEASURE(DQ1,D1), UNIT(DQ1,MLS),
    MEASURE(DQ2,D2), UNIT(DQ2,MLS),
    MEASURE(DQ3,D3), UNIT(DQ3,MLS),
    MEASURE(DQ0,D0), UNIT(DQ0,MLS),
    MEASURE(AQ1,2*(-1)), UNIT(AQ1,FT_SEC*(-2)),
    MEASURE(AQ3,(-2)), UNIT(AQ3,FT_SEC*(-2)),
    MEASURE(VQ,45), UNIT(VQ,MLS_HRS*(-1)),
    CONCAVITY(TRACK,STLINE), SLOPE(TRACK,HOR),
    CONCAVITY(SEG1,STLINE),SLOPE(SEG1,HOR),
    CONCAVITY(SEG2,STLINE), SLOPE(SEG2,HOR),
    CONCAVITY(SEG3,STLINE), SLOPE(SEG3,HOR)  ]),
  CUE(LINE_MOTION(TRAIN,TRACK,EPISODE)),
  CUE(LINE_MOTION(TRAIN,SEG1,PERIOD1)),
  CUE(LINE_MOTION(TRAIN,SEG2,PERIOD2)),
  CUE(LINE_MOTION(TRAIN,SEG3,PERIOD3)),
  ASSERTA(SOUGHT(TQ0)).

/

:- END.

:- 'NOLC'.

/*BLOC.PRB*/
/*PROBLEM 3, STAGE 4*/
/*THE KLEEKS SLIDING BLOCK PROBLEM*/
/*ALAN BUNDY SEPT 1976*/

/*CHANGE TO PATHSYS?*/
PATHINFO(_NAME,_END,_RENT,_SLOPE,_CONV) :-
```

7. check

partition(track, [seg1, seg2, seg3]),
conflength(track, dq0)
conflength(seg1, dq1)
conflength(seg2, dq2)
conflength(seg3, dq3)

```
  CHECKLIST(PASSERTA,[ PATH(_NAME), POINT(_JEND),
    END(_NAME,_JEND,LEFT), END(_NAME,_REND,RIGHT),
    (CONCAVITY(_NAME,_CONV) ), (SLOPE(_NAME,_SLOPE) )]).

PROBINFO :-
  CHECKLIST(PASSERTA,[ PROBTYPE(ROLLER-COASTER,_T), PARTICLE(M),
    PATH(S0), PARTITION(S0,LS1,S2,S3]),
    END(S0,CA,LEFT), END(S0,CB,RIGHT),
    VEL(M,ZERO,DIRA,MA), AT(M,CA,MA),
     SIDE(M,CA,LEFT,MA)] ).


MISCINFO :-
 CHECKLIST(PASSERTA, [
    (DROP(S1,CA,H1) ), (DROP(S2,CB,H2) ),
    GROUND(S3,L) , (INCLINE(S3,T,CC) ),
    (MEASURE(H1,2) ), (MEASURE(H2,-1) ),
    (MEASURE(L,2) ), (MEASURE(T,30) ),
    POINT(CD) ] ) .


STEADY :-
  TRACE(PROBLEM-DEFINED-BY,4), NL,
  PROBINFO, PATHINFO(S1,CA,CB,LEFT,LEFT),
  PATHINFO(S2,CB,CC,RIGHT,LEFT), PATHINFO(S3,CC,CD,RIGHT,STLINE),
  MISCINFO.


GOAL :- DA(AT(M,CD),_MOM),SOLVEINFO(_X,_VAL)).



:- END.



:- 'NOLC'.
```

/& Pulley.pro #1

```
/*G LUGER NOV 197/ PROBLEM 1*/

STEADY :-    CHECKLIST(PASSERTA,[
               PERIOD(PERIOD1),
               ISA(PARTICLE,P1),
               ISA(PARTICLE,P2)   ]),

          CHECKLIST(PASSERTA,[
               MASS(P1,BQ,PERIOD1),
               MASS(P2,CQ,PERIOD1),
               ACCEL(P1,A1,90,PERIOD1),
               MEASURE(BQ,B), MEASURE(CQ,C)   ]),

          CHECKLIST(ASSERTA,[
               GIVEN(BQ), GIVEN(CQ),
                SOUGHT(A1)  ]),

             CUE(PULLSYS_STAN(SYS,PULL,STR,P1,P2,PERIOD1)).
```

```
:- END.


:- 'NOLC'.
/*PROBLEM 5, STAGE 4*/
/*DE KLEERS GREAT DOME PROBLEM*/
/*ALAN BUNDY 30/12/76*/


STEADY1 :-
  CHECKLIST(PASSERTA,[
    PATH(S), POINT(TOP), POINT(BOTTOM),
    END(S,TOP,LEFT), END(S,BOTTOM,RIGHT),
    CONCAVITY(S,RIGHT), SLOPE(S,LEFT)  ]).

STEADY2 :-
  CHECKLIST(PASSERTA,[
    VEL(M,ZERO,0,DEPART), AT(M,TOP,DEPART),
    SIDE(M,TOP,LEFT,DEPART),
    PROBTYPE(ROLLER-COASTER,_T), PARTICLE(M),
    PARTITION(C,[S,..REST]), CIRCLE(C), RADIUS(C,R),
    ANGLE(TOP,90,C), ANGLE(BOTTOM,0,C),
    NORMAL(S,DIR), NUDGE(M,DEPART) ]).


STEADY3 :-
  CHECKLIST(ASSERTA,[
    GIVEN(DIR),GIVEN(R) ]).

STEADY :- STEADY1,STEADY2,STEADY3.

GOAL :- QA(MOTION(M,S,TOP,LEFT,_PER),MIN(DIR,_MINVAL)).




:- END.



:- 'NOLC'.
/*PROBLEM 4, STAGE 4*/
/*DE KLEERS LOOP THE LOOP PROBLEM*/
/*ALAN BUNDY 30/12/76*/

PATHINFO(_NAME,_LEND,_REND,_SLOPE,_CONV) :-
  CHECKLIST(PASSERTA,[ PATH(_NAME), POINT(_LEND),
    END(_NAME,_LEND,LEFT), END(_NAME,_REND,RIGHT),
    (CONCAVITY(_NAME,_CONV) ), (SLOPE(_NAME,_SLOPE) )]).

PROBINFO :-
  CHECKLIST(PASSERTA,[ PROBTYPE(ROLLER-COASTER,_T), PARTICLE(M),
    PATH(SO), PARTITION(SO,[S1,S2,S3,S4,S5]),
    END(SO,CA,LEFT), END(SO,CB,RIGHT),
    VEL(M,ZERO,DIRA,MA), AT(M,CA,MA),
    SIDE(M,CA,LEFT,MA)] ).
```

```
MISCINFO :-
  CHECKLIST(PASSERTA,[
    PARTITION(CIRCLE,[S2,S3,S4,S5]),
    CIRCLE(CIRCLE), RADIUS(CIRCLE,R),
    ANGLE(CB,270,CIRCLE), ANGLE(CC,0,CIRCLE),
    ANGLE(CD,90,CIRCLE),ANGLE(CE,180,CIRCLE),
    DROP(S1,CA,H) ]),
  CHECKLIST(ASSERTA,[GIVEN(H), GIVEN(R) ]).


STEADY :-
  TRACE(PROBLEM-DEFINED-BY,4), NL,
  PROBINFO,
  PATHINFO(S1,CA,CB,LEFT,LEFT),
  PATHINFO(S2,CB,CC,RIGHT,LEFT),
  PATHINFO(S3,CD,CC,LEFT,RIGHT),
  PATHINFO(S4,CE,CD,RIGHT,RIGHT),
  PATHINFO(S5,CE,CB,LEFT,LEFT),
  MISCINFO.

GOAL :-  DA(MOTION(M,SO,CA,LEFT,_PER),MIN(H,_MINVAL)).




:- END.
```

Solve (S_2) Solve (circle)