```
;; COAST.SUB - All Alan's Roller Coaster stuff
;;
coast.sub        ;; This File
coast.           ;  Program load file
top.
energy.
rules.
schema.
predic.
paths.
angle.
parity.
mk2.
typ2.
hack.
aik.
dome.prb         ;; Problem files
bloc.prb         ;;
loop.prb         ;;
drop.prb         ;;
fly.prb          ;;
ramp.prb         ;;
tdome.prb        ;;
tloop.prb        ;;
tfly.prb         ;;
tramp.prb        ;;
dome.sol         ;; Solution files (traces)
bloc.sol         ;;
loop.sol         ;;
drop.sol         ;;
fly.sol          ;;
ramp.sol         ;;
tdome.sol        ;;
tloop.sol        ;;
tfly.sol         ;;
tramp.sol        ;;
```

```
/* coast
Consult files for Roller Coaster Problems
Alan Bundy 7.4.81 */

:-[
        top,                % Top level stuff
        predic,             % Motion Prediction
        paths,              % Inference Rules  for paths
        parity,             % Switching parity this way and that
        energy,             % Conservation of Energy Formulae
        schema,             % Motion schema
        hack,               % Make cc create types for isa
        mk2,                % Meta-level knowledge
        qik,                % Quick restore and save
        rules               % Other inference rules, inc defaults
],

load([
        typ2                % Type definitions
]),
```

```
/* top.
Roller Coaster problems
Alan Bundy 7.4.81 */

/* Top Level: Question Answering */

  qa(Qual,Quan)
        :- Qual,
           findall(X,proviso(X),Condlist),
           trace('%t ok provided :%1',[Qual,Condlist],2),
           setup(Condlist,Xs,Gs),
           set_types(Xs,[],Xtypes),
           set_types(Gs,Xtypes,Types),
           solve(Xs,Gs,Types,Es,Xs1),
           crunch(Es,Xs1,Ans),
           subst(Solns,Condlist,Ncondlist),
           trace('New conditions are :%1',[Ncondlist],2),
           apply(Quan,[Ncondlist]).

/* Find soughts and givens */

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



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

/*SEE IF ITS TRUE*/
condition(L) :-
  evaluate(L,true), !,
  trace('condition: %t holds\n', [L],3).

/*SEE IF ITS FALSE*/
condition(L) :-
  evaluate(L,false), !,
  trace('condition: %t does not hold\n', [L],3),
  fail.
```

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

/* Retractable Assert */
postulate(Ass) :-                          % assert at top first time round
        asserta(Ass).

postulate(Ass) :-                          % retract and fail on backup
        retract(Ass), !, fail.


/* Evaluate condition */

evaluate(Cond,true) :- ncc Cond, !.              % condition is true
```

```
evaluate(Cond,false) :- negate(Cond,NotCond), ncc NotCond, !.

negate(positive(X), non_pos(X)).
negate(non_neg(X),negative(X)).
negate(real(X),complex(X)).
```

```
/* predict.
Prediction for Roller Coaster Problems
Alan Bundy 7.4.81 */


/*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 duri
        [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) :-
  simple_curve(Path), !,
  dcc initial(Per,Begin),
  getstarted(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).

/* Path is a simple curve */
simple_curve(Path) :-
        dc concavity(Path,_),
        dc slope(Path,_), !.

'* Put partition of path in order */
  rangepath(Path,Start,Npathlist) :-
  dc partition(Path,Pathlist),
  dc end(Path,Start,End),
  condrev(End,Pathlist,Npathlist).

/* make a new subperiod for each subpath */
makeperiods(Npathlist,Per,Perlist) :-
  maplist(makeone,Npathlist,Perlist),
  dbentry(partition(Per,Perlist)),
  trace('Divide %t into : %l',[Per,Perlist],2).

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

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

/*MAKE ONE PERIOD */
```

```
makeone(Path,Per) :- ccreate(period,Per).

/* Gets started on motion */
/*-------------------------*/

/* Stationary at start */
setstarted(Part,Path,Start,Side,Begin) :-
  ncc vel(Part,zero,Dir,Begin), !,
  ( horizontal(Path,Start) ->
  dc nudge(Part,Begin) ; top(Path,Start) ).

/*HEADED IN RIGHT DIRECTION*/
setstarted(Part,Path,Start,Side,Begin) :- !,
  alongs(Path,Start,Start,Dir),
  cc vel(Part,V,Dir,Begin),
  condition(positive(V)).


/* Particle does not run out of steam */
/*------------------------------------*/

/*DOWNHILL OR HORIZONTAL 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 farend(Path,Start,Finish),
 alongs(Path,Start,Finish,Dir),
 cc finvel(Part,V,Dir,Per),
 condition(real(V)).

/* Particle does not take off */
/*----------------------------*/

/* Particle is threaded on */
notakeoff(Part,Path,Start,threaded,Per) :- !.

  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), !,
  cc typical_point(Path,TypPt),
  towards(Path,Part,Start,Side,TypPt,Per,Dir),
  cc reaction(Path,Part,N,Dir,Per),
  condition(non_neg(N)).


/* Particle does not fall off */
/*----------------------------*/

/* It is threaded on */
```

```
nofalloff(Part,Path,Start,threaded,Per) :- !.

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

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

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

/*FREEFALL*/              %Put in failure cases on other conditions?
nofalloff(Part,Path,Start,Side,Per) :-
  dc concavity(Path,Conc), diff(Conc,right),
  trace('%t falls off %t during %t\n',[Part,Path,Per],1),
  assert(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) :-  % Side2 is looking from Start
  ncc at(Part,Start,Time), !,
  dc side(Part,Start,Side1,Time),       % Side1 is looking from left end
  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)), !,                % duplication
  pc motion(Part,Path,Place1,Side,Per).
```

```
/* paths.
Describing Paths for Roller Coaster Problems
Alan Bundy 7.4.81 */


/*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,Par1), opposite(Par1,Par2),
  dc end(Path,Point2,Par2).

/*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,Par), dc slope(Path,Par).

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

```
/* ANGLE.
Inference Rules for Angles and Inclinations
Alan Bundy 9.4.81 */


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

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

/* incline is at rightangle to angle */
incline(Path,Ang1,Point) :-
  ncc angle(Path,Ang2,Point), leftturn(Ang2,Ang1).

/* Another path shares the point */
incline(Ph2,Ang2,Pt) :-
  type(path,Ph2),
  dc point_of(_,Ph1,Pt),
  diff(Ph1,Ph2),                        % add smooth transition condition!!
  ncc incline(Ph1,Ang1,Pt),
  condturn2(Pt,Ph1,Ph2,Ang1,Ang2).      % angle may flip 180 degrees

/*FIND ANGLE OF MINIMAL PATH*/
angle(Path,Ang,Point) :-
  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 At Pt ALONG PATH FROM ONE END*/

/* For simple curves */
along(Path,Start,Pt,Dir) :-
  dc concavity(Path,Conc), norm(Conc,Nc),
  dc end(Path,Start,Par),
  cc incline(Path,Dir,Pt),
  condturn1(Nc,Par,Dir,Ang).

/* For non-simple curves */
along(Path,Start,Pt,Dir) :-
        dc point_of(path,Path1,Pt),             %ugly hack
        diff(Path,Path1),
        dc cued(motion(Part,Path1,Start1,Side1,Per1)),
        along(Path1,Start1,Pt,Dir).

/* Dir2 is normal from Surface to Part at Pt at Time */
towards(Surface,Part,Start,SSide,Pt,Time,Dir2) :-
        cc angle(Surface,Dir1,Pt),
        dc end(Surface,Start,Par),
        condval(Par,SSide,LSide),
        dc concavity(Surface,Conv), norm(Conv,Nc),
        opposite(Nc,Nc1),
        condturn1(Nc1,LSide,Dir1,Dir2).
```

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


/* Short term (?) hack - Angles can be eliminated from equations
using stationary values method, and do not need solving for */

eliminable(Quan) :- kind(Quan,angle,_,_).
```

```
/* rules.
Inference rules for Roller Coaster problems
Alan Bundy 7.4.81 */

/* Path is free of particle to travel on,
i.e. no friction or extraneous forces */

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

/* This is a motion type problem */

  probtype(motion)
        :- ncc probtype(roller_coaster) ;
           ncc probtype(motion_in_a_straisht_line).



/* Default: paths in motion problems are frictionless */
  coeff(Path,zero)
        :- type(path,Path),
           ncc probtype(motion).

/* The friction of paths is inherited by their subpaths */
  coeff(Path,Mu)
        :- bitof(Path,Suppath),
           ncc coeff(Suppath,Mu).


/* Default: a path is assumed roush if it is not known to be smooth */
  roush(Path) :- thnot( ncc(coeff(Path,zero)) ).



/*VERTICAL DROP OF PATH FROM START TO FINISH*/
/*-----------------------------------------*/

/* drop of horizontal straisht line is zero */
 drop(Start,Finish,zero) :-
        farend(Path,Start,Finish),
        dc concavity(Path,stline),
        horizontal(Path,Start).

/* drop of straisht line can be calculated from sround */
drop(Start,Finish,-(D*tan(Ans))) :-
    farend(Path,Start,Finish),
    dc concavity(Path,stline),
    ncc incline(Path,Ans,Start), dc sround(Path,D).

/* drop is anti-commutative */
/* drop(Start,Finish,-H) :-
        ncc drop(Finish,Start,H).
*/
/* drop of path is sum of drops of subpath */
drop(Start,Finish,Hsum) :-
        farend(Path,Start,Finish),
        dc partition(Path,Pl),
        sumdrops(Pl,Start,Hsum).
```

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

/* Sum of Drops across partition */
sumdrops([],Start,0).

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

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

```
/* parity.
Parity switching for Roller Coaster Problems
Alan Bundy 7.4.81 */


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


/*PARTICLE IS INSIDE OR OUTSIDE OF CONCAVITY IN PATH*/ % Are these used?
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).

/* Which side of path is particle on? */
wh_side(Part,Path,Ans,Nc,Time) :-
  dc cued(motion(Part,Path,Start,Side,Time)),
  dc end(Path,Start,Par),
  condval(Par,Side,Ans),
  dc concavity(Path,Conv), norm(Conv,Nc).

   The inclinations of two paths at a shared point are either
identical or at 180 */

/* Ph1 is a subpath of Ph2 */
condturn2(Pt,Ph1,Ph2,Ans,Ans) :-
        dc partition(Ph2,PhList),
        member(Ph1,PhList), !.

/* Ph2 is a subpath of Ph1 */
condturn2(Pt,Ph1,Ph2,Ans,Ans) :-
        dc partition(Ph1,PhList),
        member(Ph2,PhList), !.

/* Ph1 and Ph2 join (smoothly?) at Pt */
condturn2(Pt,Ph1,Ph2,Ans1,Ans2) :- !,
  dc end(Ph1,Pt,Par1),
  dc end(Ph2,Pt,Par2),
  condval(Par,Par1,Par2),                % Par is right iff Par1 & 2 differ
  dc concavity(Ph1,Conv1), norm(Conv1,Nc1),
  dc concavity(Ph2,Conv2), norm(Conv2,Nc2),
  condval(Nc,Nc1,Nc2),                   % Nc is right iff Nc1 & 2 differ
  condturn3(Par,Nc,Ans1,Ans2).           % not clear this is right

/* Angles are turned iff parities are right right */
condturn3(right,right,Ans1,Ans2) :- !,
        aboutturn(Ans1,Ans2).
condturn3(Par1,Par2,Ans,Ans) :- !.


/* Angles are turned iff parities are different */
condturn1(Par,Par,Ans,Ans) :- !.
condturn1(Par1,Par2,Ans1,Ans2) :-
  opposite(Par1,Par2), aboutturn(Ans1,Ans2).

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

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

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

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

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

```
/* energy.
Conservation of Energy Formula for Roller Coaster Problems
Alan Bundy 8.4.81 */

/* Preference ratins of formula */
preference(consvenersy,3) :- probtype(roller_coaster).

/* What quantities does formula relate? */
relates(consvenersy,[lensth,vel]).

/* Prepare situation of formula */
/*-------------------------------*/

/* when V is initial velocity */
prepare(consvenersy,V,vel,relvel(ObJ1,ObJ2,V,Dir,Initial),
                          situation(ObJ,Start,Finish,Initial,Final)) :-
        ncc moment(Initial),
        ncc initial(Period,Initial),
        cc final(Period,Final),
        perm2(ObJ1,ObJ2,ObJ,earth),
        dc cued(motion(ObJ,Path,Start,Side,Period)),
        dc fixed_contact(ObJ,Finish,Final),
        ncc free(Path,ObJ,Period).


/* when V is final velocity */
prepare(consvenersy,V,vel,relvel(ObJ1,ObJ2,V,Dir,Final),
                          situation(ObJ,Start,Finish,Initial,Final)) :-
        ncc moment(Final),
        ncc final(Period,Final),
        cc initial(Period,Initial),
        perm2(ObJ1,ObJ2,ObJ,earth),
        dc cued(motion(ObJ,Path,Start,Side,Period)),
        dc fixed_contact(ObJ,Finish,Final),
        ncc free(Path,ObJ,Period).


/* when V is typical velocity */
prepare(consvenersy,V,vel,relvel(ObJ1,ObJ2,V,Dir,Period),
                          situation(ObJ,Start,TypPt,Initial,Period)) :-
        ncc period(Period),
        cc initial(Period,Initial),
        perm2(ObJ1,ObJ2,ObJ,earth),
        dc cued(motion(ObJ,Path,Start,Side,Period)),
        cc typical_point(Path,TypPt),
        ncc free(Path,ObJ,Period).


/*  The Formula */

isform(consvenersy,situation(ObJ,Start,Finish,Initial,Final),
                                        s*H=((V^2)/2)-((U^2)/2))
        :- ncc drop(Start,Finish,H),
           cc vel(ObJ,V,Dir1,Final),
           cc vel(ObJ,U,Dir2,Initial).
```

```
/* SCHEMA : Motion schemas

                                                    Updated: 8 April 81
*/

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

                [ probtype(motion_in_a_straight_line),
                  concavity(Path,stline),
                  slope(Path,hor)  ],

                []  ),


  schema(motion(Part,Path,Start,Side,Time),
                [ cc farend(Path,Finish,Start),
                  cc final(Time,End),
                  along(Path,Start,Finish,Dir),
                  cc vel(Part,V,Dir,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 constccel(Part,Time)  ),
                  ( positive(V)
                        :-?~~loc(Path,Start),~~?
                           ncc probtype(roller_coaster)  )
                                                          ]  ),
```

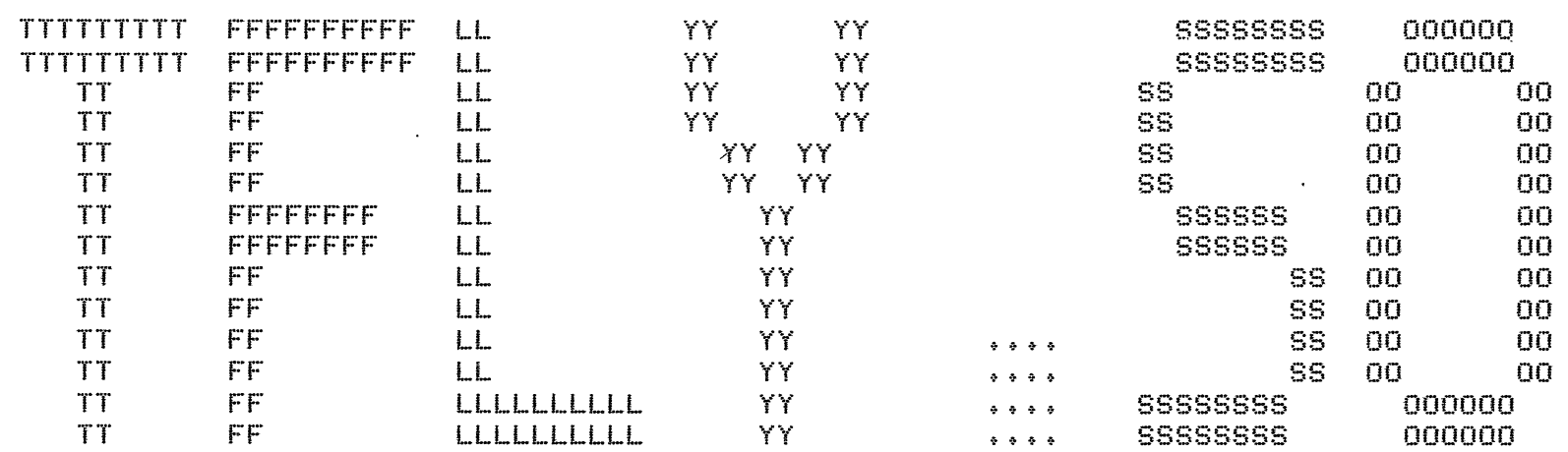
```
/* HACK - Instant hack */

:- asserta((

defn( Term, edb,
                [ type(Type,Ars), ccreate(Type,Ars) ],
                [])
     :- functor(Term,Type,1),
        istype(Type),
        !,
        ars(1,Term,Ars)
)),


:- asserta((

  defn( isa(T,X),edb,
                [ type(T,X), ccreate(T,X) ],
                []   ) :- !
)),
```

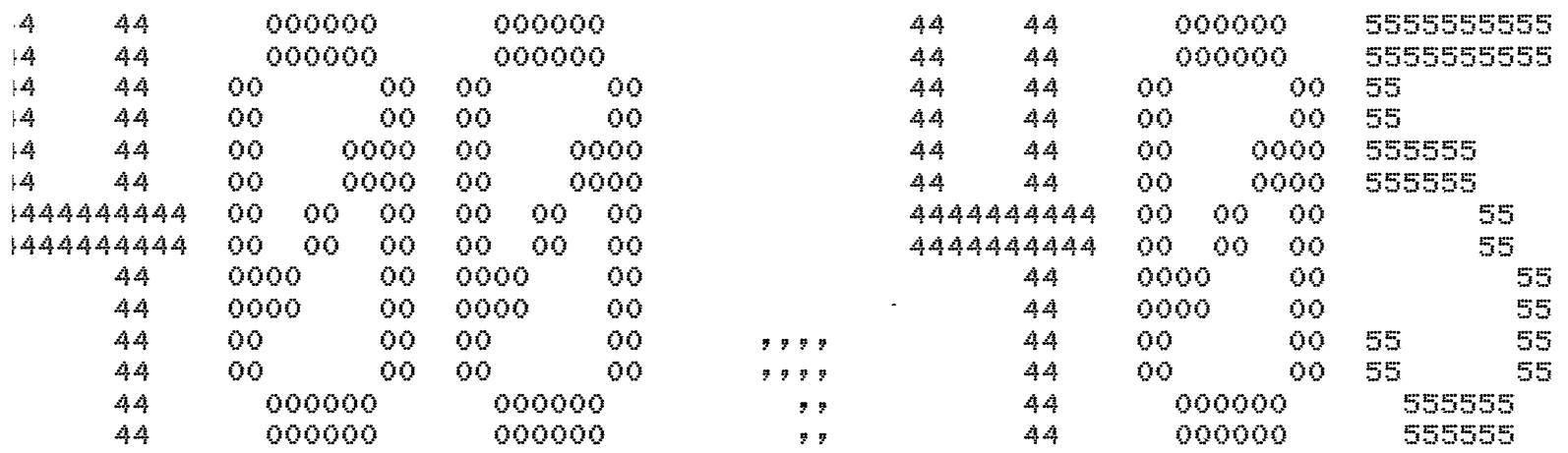
```
/* MK2 : Meta level knowledge for the Roller Coaster Problems

                                                    Updated: 7 April 81
*/


:- meta_knowledge ).
%--------------------


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

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

arsstruct(height,2,
                  [object,length],
                  [ars,val]).

arsstruct(side,4,
                  [particle,point,parity,time],
                  [ars,ars,val,ars]).

arsstruct(nudge,2,
                  [particle,moment],
                  [ars,ars]).
```

```
/* QIK */

s :- save('scra:save'),

r :- restore(save).
```

```
/* TYP2 : Part of type hierarchy for Roller Coaster problems */


{ types }.
%----------

circle -> path.
```

```
/* tfly.prb */
/* A threaded version of fly.prb */
/* A particle starts to move up a wire, on which it is threaded,
with velocity va. What is the minimum value of va such that it
will reach the top of the wire? */
/* Alan Bundy 1.5.81 */

        cue pathsys(wire,bottom,top,right,left).
        incline(wire,dir,bottom).
        isa(particle,p).
        at(p,bottom,depart).
        vel(p,va,dir,depart).
        positive(va).
        drop(bottom,top,ha).
        side(p,bottom,threaded,depart).

        probtype(roller_coaster).
        given(va).
        given(ha).
        given(dir).

        goal :- aa(at(p,top,Mom) , min(va,Ans)).
```





START User BUNDY HPS [400,405] Job TFLY Seq. 3255 Date 05-May-81 15:20:29
File: DSKA:TFLY.SOL<005>[400,405,COAST] Created: 05-May-81 15:19:13 Printed: 05-Ma
QUEUE Switches: /FILE:ASCII /COPIES:1 /SPACING:1 /LIMIT:53 /FORMS:NORMAL

```
ues
| ?- input(tfly).
Pullins in schema pathsys(wire,bottom,top,risht,left)
Pullins in schema linesys(path,wire,[bottom,top])
 Let wire be a new path
 Let bottom be a new point
 Let top be a new point
 Note: dir (of type anSle) was used in a incline definition (2)
 Let p be a new particle
 Note: va (of type vel) was used in a relvel definition (3)
 Note: dir (of type anSle) was used in a relvel definition (4)

K* ERROR Cannot record kind for positive(va)
   ( continue after error )
 Note: ha (of type lenSth) was used in a drop definition (3)

tfly problem read into data base.


ues
| ?- saol.

no
| ?- soal.
Pullins in schema timesys(_69,depart,_33)
 Let period1 be a new period
 Let depart be the bndy of period1 on the left.
 Let bndy1 be the bndy of period1 on the risht.
 Let depart be a new moment
 Let bndy1 be a new moment
'ullins in schema timesys(period1,depart,_4371)

(* ERROR Nfrec already recorded: cued(timesys(period1,depart,bndy1))
   ( continue after error )
:ondition: positive(va) holds
 in direction anSle1 at top.
 Note: anSle1 (of type anSle) was used in a incline definition (2)
 Let relvel1 be the relvel of p in direction anSle1 relative to earth.
 Note: relvel1 (of type vel) was used in a relvel definition (3)
 Note: anSle1 (of type anSle) was used in a relvel definition (4)
itorins proviso : real(relvel1).
'ullins in schema motion(p,wire,bottom,threaded,period1)
 Let typical_point1 be a new typical_point
  p is in the same place as top durins bndy1.
  p is in the same place as bottom durins depart.
  p is in the same place as typical_point1 durins period1.
:t(p,top,bndy1) ok provided :
   real(relvel1)

ttemptins to solve for [relvel1] in terms of [va,ha,dir]


 am now tryins to solve for relvel1 without introducins any unknowns.

Applicable formulae : [consvenersy,relvel,constvel,constaccel-1,constaccel-2,const
(try  consvenersy)
Tryins to apply stratesy(consvenersy,situation(p,bottom,top,depart,bndy1))

Equation-1 : s*ha=relvel1^2/2-va^2/2
```

formed by applying : strategy(conservenergy,situation(p,bottom,top,depart,bndy1))

This equation solves for relvel1.

[No unknowns] Do you accept this equation ? yes.

So now I must solve for []
given [relvel1,va,ha,dir]

Equations extracted :
s*ha=relvel1^2/2-va^2/2

yes
?- core 83456 (54272 lo-ses + 29184 hi-ses)
heap 49152 = 47516 in use + 1636 free
global 1187 = 16 in use + 1171 free
local 1024 = 16 in use + 1008 free
trail 511 = 0 in use + 511 free
0.06 sec. for 1 GCs gaining 507 words
0.27 sec. for 17 local shifts and 23 trail shifts
4.95 sec. runtime

```
/* tdome.prb */
/* A threaded version of dome.prb */
/* Alan Bundy 1.5.81 */

        circle(circle).
        radius(circle,ra).
        partition(circle,[wire,quad2,quad3,quad4]).
        solid(wire).
        cue pathsys(wire,top,bottom,left,right).
        angle(wire,90,top).
        angle(wire,0,bottom).
        normal(wire,dir).
        isa(particle,p).
        mass(p,ma,period1).
        at(p,top,depart).
        vel(p,zero,0,depart).
        side(p,top,threaded,depart).
        nudge(p,depart).


        probtype(roller_coaster).
        given(ra).
        given(ma).
        given(dir).

        goal :- ea(motion(p,wire,top,left,period1) , min(dir,ANS)).
```

```
| ?- input(tdome).
 Let circle be a new circle
 Note: rq (of type length) was used in a radius definition (2)
Pulling in schema pathsys(wire,top,bottom,left,right)
Pulling in schema linesys(path,wire,[top,bottom])
 Let wire be a new path
 Let top be a new point
 Let bottom be a new point
 Let typical_point1 be a new typical_point
 Note: dir (of type angle) was used in a angle definition (2)
 Let p be a new particle
 Note: mq (of type mass) was used in a mass definition (2)
 Note: zero (of type vel) was used in a relvel definition (3)

tdome problem read into data base.


yes
| ?- goal.
Pulling in schema timesys(period1,depart,_67)
 Let period1 be a new period
 Let depart be the bndy of period1 on the left.
 Let bndy1 be the bndy of period1 on the right.
 Let depart be a new moment
 Let bndy1 be a new moment
Pulling in schema motion(p,wire,top,threaded,period1)
 Let relvel1 be the relvel of p in direction 90 relative to earth.
 Note: relvel1 (of type vel) was used in a relvel definition (3)
  p is in the same place as bottom during bndy1.
  p is in the same place as top during depart.
  p is in the same place as typical_point1 during period1.
motion(p,wire,top,threaded,period1) ok provided :

Attempting to solve for [] in terms of [rq,mq,dir]


Equations extracted :


yes
| ?- core     83456  (54272 lo-ses + 29184 hi-ses)
heap     49152 =  47548 in use +   1604 free
global    1187 =     16 in use +   1171 free
local     1024 =     16 in use +   1008 free
trail      511 =      0 in use +    511 free
    0.06 sec. for 1 GCs gaining 515 words
    0.13 sec. for 11 local shifts and 19 trail shifts
    3.93 sec. runtime
```

```
/* tramp.prb */
/* A threaded version of ramp.prb */
/* A wire consists of a convex, circular slope joined smoothly
to a horizontal top piece. A particle, threaded on the wire is projected up
it with a velocity va. What is the least value of va
such that the particle will reach the end of the wire?
/* Alan Bundy 1.5.81 */

        isa(path,wire),
        partition(wire,[s1,s2]),
        cue pathsys(s1,pt1,pt2,left,right),
        cue pathsys(s2,pt2,pt3,hor,stline),
        cue linesys(path,wire,[pt1,pt2,pt3]),
        circle(circ),
        radius(circ,ra),
        partition(circ,[s1,s3]),
        solid(s1),
        angle(s1,90,pt2),
        incline(s1,dir,pt1),
        mass(p,ma,initial),
        isa(particle,p),
        at(p,pt1,initial),
        vel(p,va,dir,initial),
        positive(va),
        side(p,pt1,threaded,initial),

        probtype(roller_coaster),
        given(ra),
        given(ma),
        given(dir),
        given(va),

        goal :- ga( at(p,pt3,Mom) , min(va,ANS) ),
```

```
yes
| ?- input(tramp).
 Let wire be a new path
Pullins in schema pathsys(s1,pt1,pt2,left,right)
Pullins in schema linesys(path,s1,[pt1,pt2])
 Let s1 be a new path
 Let pt1 be a new point
 Let pt2 be a new point
Pullins in schema pathsys(s2,pt2,pt3,hor,stline)
Pullins in schema linesys(path,s2,[pt2,pt3])
 Let s2 be a new path
 Let pt3 be a new point
Pullins in schema linesys(path,wire,[pt1,pt2,pt3])
 Let circ be a new circle
 Note: ra (of type length) was used in a radius definition (2)
 Note: dir (of type angle) was used in a incline definition (2)
 Note: ma (of type mass) was used in a mass definition (2)
 Let p be a new particle
 Note: va (of type vel) was used in a relvel definition (3)
 Note: dir (of type angle) was used in a relvel definition (4)

** ERROR Cannot record kind for positive(va)
    ( continue after error )

tramp problem read into data base.


yes
| ?- goal.
Pullins in schema timesys(_69,initial,_33)
 Let period1 be a new period
 Let initial be the bndy of period1 on the left.
 Let bndy1 be the bndy of period1 on the right.
 Let initial be a new moment
 Let bndy1 be a new moment
Pullins in schema timesys(period1,initial,_4371)

** ERROR Nfrec already recorded: cued(timesys(period1,initial,bndy1))
    ( continue after error )
 Let period2 be a new period
 Let period3 be a new period
 divide period1 into :
     period2
     period3
Pullins in schema timesys(period2,initial,_7325)
 Let initial be the bndy of period2 on the left.
 Let bndy2 be the bndy of period2 on the right.
 Let bndy2 be a new moment
condition: positive(va) holds
Pullins in schema motion(p,s1,pt1,threaded,period2)
 Let relvel1 be the relvel of p in direction 180 relative to earth.
 Note: relvel1 (of type vel) was used in a relvel definition (3)
 Let typical_point1 be a new typical_point
  p is in the same place as pt2 during bndy2.
  p is in the same place as pt1 during initial.
  p is in the same place as typical_point1 during period2.
Pullins in schema timesys(period3,bndy2,_19845)
 Let bndy3 be the bndy of period3 on the right.
 Let bndy3 be a new moment
```

```
 Let relvel2 be the relvel of p in direction 0 relative to earth.
 Note: relvel2 (of type vel) was used in a relvel definition (3)
Storing proviso : positive(relvel2).
Pulling in schema motion(p,s2,pt2,threaded,period3)
 Let relvel3 be the relvel of p in direction 0 relative to earth.
 Note: relvel3 (of type vel) was used in a relvel definition (3)
 Let typical_point2 be a new typical_point
  p is in the same place as pt3 during bndy3.
  p is in the same place as pt2 during bndy2.
  p is in the same place as typical_point2 during period3.
Pulling in schema motion(p,wire,pt1,threaded,period1)
 Let relvel4 be the relvel of p in direction 0 relative to earth.
 Note: relvel4 (of type vel) was used in a relvel definition (3)
 Let typical_point3 be a new typical_point
  p is in the same place as pt3 during bndy1.
  p is in the same place as pt1 during initial.
  p is in the same place as typical_point3 during period1.
at(p,pt3,bndy1) ok provided :
    positive(relvel2)

Attempting to solve for [relvel2] in terms of [ra,ma,dir,va]


I am now trying to solve for relvel2 without introducing any unknowns.

 Applicable formulae : [consvenersy-1,consvenersy-2,relvel,constvel,constaccel-1,
 (try  consvenersy-1)
 Trying to apply strategy(consvenersy-1,situation(p,s2,pt2,period3))

 Equation-1 : s*zero=relvel3^2/2-relvel2^2/2
 formed by applying : strategy(consvenersy-1,situation(p,s2,pt2,period3))

 Equation-1 rejected.

 Trying to apply strategy(consvenersy-1,situation(p,s1,pt1,period2))

 Equation-2 : s*(ra*(sin(dir--90)-sin(90)))=relvel2^2/2-va^2/2
 formed by applying : strategy(consvenersy-1,situation(p,s1,pt1,period2))

 This equation solves for relvel2.

[ No unknowns ]   Do you accept this equation ? yes.

 So now I must solve for []
                  given [relvel2,ra,ma,dir,va]


Equations extracted :
    s*(ra*(sin(dir--90)-sin(90)))=relvel2^2/2-va^2/2


yes
| ?- core     84480  (55296 lo-ses + 29184 hi-ses)
heap     50176 =  48309 in use +   1867 free
global    1187 =     16 in use +   1171 free
local     1024 =     16 in use +   1008 free
trail      511 =      0 in use +    511 free
    0.80 sec. for 2 GCs gaining 46474 words
    1.17 sec. for 29 local shifts and 40 trail shifts
   17.56 sec. runtime
```

```
/* tloop.prb */
/* A threaded version of loop.prb */
/* Alan Bundy 1.5.81 */

        partition(s0,[s1,s2,s3,s4,s5]).
        radius(circle1,rq).
        partition(circle1,[s2,s3,s4,s5]).
        cue linesys(path,s0,[pt1,pt2,pt3,pt4,pt5,pt2]).
        cue linesys(circle,circle1,[pt2,pt3,pt4,pt5,pt2]).
        cue pathsys(s1,pt1,pt2,left,left).
        cue pathsys(s2,pt2,pt3,right,left).
        cue pathsys(s3,pt4,pt3,left,right).
        cue pathsys(s4,pt5,pt4,right,right).
        cue pathsys(s5,pt5,pt2,left,left).
        angle(circle1,270,pt2).
        angle(circle1,0,pt3).
        angle(circle1,90,pt4).
        angle(circle1,180,pt5).
        drop(pt1,pt2,hq).
        isa(particle,p).
        at(p,pt1,initial).
        incline(s1,dir1,pt1).
        vel(p,zero,dir1,initial).
        side(p,pt1,threaded,initial).

        probtype(roller_coaster).
        given(rq).
        given(hq).

        goal :- qa( motion(p,s0,pt1,threaded,Per) , min(hq,ANS) ).
```

```
TTTTTTTTTT  LL            000000      000000    PPPPPPPP                  SSSSSSS
TTTTTTTTTT  LL            000000      000000    PPPPPPPP                  SSSSSSS
    TT      LL          00    00    00    00    PP    PP                SS
    TT      LL          00    00    00    00    PP    PP                SS
    TT      LL          00    00    00    00    PP    PP                SS
    TT      LL          00    00    00    00    PP    PP                SS
    TT      LL          00    00    00    00    PPPPPPPP                  SSSSS
    TT      LL          00    00    00    00    PPPPPPPP                  SSSSS
    TT      LL          00    00    00    00    PP                            SS
    TT      LL          00    00    00    00    PP                            SS
    TT      LL          00    00    00    00    PP                ....        SS
    TT      LL          00    00    00    00    PP                ....        SS
    TT      LLLLLLLLLL    000000      000000    PP                ....    SSSSSSS
    TT      LLLLLLLLLL    000000      000000    PP                ....    SSSSSSS


44    44      000000      000000                44    44      000000    555555555
44    44      000000      000000                44    44      000000    555555555
44    44    00      00    00      00            44    44    00      00  55
44    44    00      00    00      00            44    44    00      00  55
44    44    00    0000    00    0000            44    44    00    0000  555555
44    44    00    0000    00    0000            44    44    00    0000  555555
4444444444  00  00  00    00  00  00            4444444444  00  00  00        55
4444444444  00  00  00    00  00  00            4444444444  00  00  00        55
      44    0000    00    0000    00                  44    0000    00          5
      44    0000    00    0000    00                  44    0000    00          5
      44    00      00    00      00      ,,,,        44    00      00  55      5
      44    00      00    00      00      ,,,,        44    00      00  55      5
      44      000000      000000          ,,          44      000000      555555
      44      000000      000000          ,,          44      000000      555555


BBBB  U   U N   N DDDD  Y   Y                  H   H PPPP   SSSS
B   B U   U N   N D   D Y   Y                  H   H P   P S
B   B U   U NN  N D   D  Y Y                   H   H P   P S
BBBB  U   U N N N D   D   Y                    HHHHH PPPP   SSS
B   B U   U N  NN D   D   Y                    H   H P         S
B   B U   U N   N D   D   Y                    H   H P         S
BBBB  UUUUU N   N DDDD    Y                    H   H P     SSSS


 *START* User BUNDY    HPS [400,405]  Job  TLOOP Seq, 3222 Date 05-May-81 13:11:0
File: DSKA:TLOOP.SOL<005>[400,405,COAST] Created: 05-May-81 13:07:39 Printed: 05-
QUEUE Switches:  /FILE:ASCII /COPIES:1 /SPACING:1 /LIMIT:58 /FORMS:NORMAL
```

```
yes
| ?- input(tloop).
 Note: ra (of type length) was used in a radius definition (2)
Pullins in schema linesys(path,s0,[pt1,pt2,pt3,pt4,pt5,pt2])
 Let s0 be a new path
 Let pt1 be a new point
 Let pt2 be a new point
 Let pt3 be a new point
 Let pt4 be a new point
 Let pt5 be a new point

** ERROR Nfrec already recorded: point_of(path,s0,pt2)
    ( continue after error )
Pullins in schema linesys(circle,circle1,[pt2,pt3,pt4,pt5,pt2])
 Let circle1 be a new circle

** ERROR Nfrec already recorded: point_of(circle,circle1,pt2)
    ( continue after error )
Pullins in schema pathsys(s1,pt1,pt2,left,left)
Pullins in schema linesys(path,s1,[pt1,pt2])
 Let s1 be a new path
Pullins in schema pathsys(s2,pt2,pt3,right,left)
Pullins in schema linesys(path,s2,[pt2,pt3])
 Let s2 be a new path
Pullins in schema pathsys(s3,pt4,pt3,left,right)
Pullins in schema linesys(path,s3,[pt4,pt3])
 Let s3 be a new path
Pullins in schema pathsys(s4,pt5,pt4,right,right)
Pullins in schema linesys(path,s4,[pt5,pt4])
 Let s4 be a new path
Pullins in schema pathsys(s5,pt5,pt2,left,left)
Pullins in schema linesys(path,s5,[pt5,pt2])
 Let s5 be a new path
 Note: ha (of type length) was used in a drop definition (3)
 Let p be a new particle
 Note: dir1 (of type angle) was used in a incline definition (2)
 Note: zero (of type vel) was used in a relvel definition (3)
 Note: dir1 (of type angle) was used in a relvel definition (4)

tloop problem read into data base.


yes
| ?- goal.
Pullins in schema timesys(_33,initial,_68)
 Let period1 be a new period
 Let initial be the bndy of period1 on the left.
 Let bndy1 be the bndy of period1 on the right.
 Let initial be a new moment
 Let bndy1 be a new moment
 Let period2 be a new period
 Let period3 be a new period
 Let period4 be a new period
 Let period5 be a new period
 Let period6 be a new period
 ivide period1 into :
    period2
    period3
    period4
```

```
    period5
    period6
Pullins in schema timesys(period2,initial,_5413)
 Let initial be the bndy of period2 on the left.
 Let bndy2 be the bndy of period2 on the right.
 Let bndy2 be a new moment
Pullins in schema motion(p,s1,pt1,threaded,period2)
 Let relvel1 be the relvel of p in direction 0 relative to earth.
 Note: relvel1 (of type vel) was used in a relvel definition (3)
 Let typical_point1 be a new typical_point
  p is in the same place as pt2 during bndy2.
  p is in the same place as pt1 during initial.
  p is in the same place as typical_point1 during period2.
Pullins in schema timesys(period3,bndy2,_19848)
 Let bndy3 be the bndy of period3 on the right.
 Let bndy3 be a new moment
condition: positive(relvel1) holds
 Let relvel2 be the relvel of p in direction 90 relative to earth.
 Note: relvel2 (of type vel) was used in a relvel definition (3)
Storing proviso : real(relvel2).
Pullins in schema motion(p,s2,pt2,threaded,period3)
 Let typical_point2 be a new typical_point
  p is in the same place as pt3 during bndy3.
  p is in the same place as pt2 during bndy2.
  p is in the same place as typical_point2 during period3.
Pullins in schema timesys(period4,bndy3,_42757)
 Let bndy4 be the bndy of period4 on the right.
 Let bndy4 be a new moment
Storing proviso : positive(relvel2).
 Let relvel3 be the relvel of p in direction 180 relative to earth.
 Note: relvel3 (of type vel) was used in a relvel definition (3)
Storing proviso : real(relvel3).
Pullins in schema motion(p,s3,pt3,threaded,period4)
 Let typical_point3 be a new typical_point
  p is in the same place as pt4 during bndy4.
  p is in the same place as pt3 during bndy3.
  p is in the same place as typical_point3 during period4.
Pullins in schema timesys(period5,bndy4,_20261)
 Let bndy5 be the bndy of period5 on the right.
 Let bndy5 be a new moment
Storing proviso : positive(relvel3).
Pullins in schema motion(p,s4,pt4,threaded,period5)
 Let relvel4 be the relvel of p in direction 270 relative to earth.
 Note: relvel4 (of type vel) was used in a relvel definition (3)
 Let typical_point4 be a new typical_point
  p is in the same place as pt5 during bndy5.
  p is in the same place as pt4 during bndy4.
  p is in the same place as typical_point4 during period5.
Pullins in schema timesys(period6,bndy5,_36071)
 Let bndy6 be the bndy of period6 on the right.
 Let bndy6 be a new moment
condition: positive(relvel4) holds
Pullins in schema motion(p,s5,pt5,threaded,period6)
 Let relvel5 be the relvel of p in direction 0 relative to earth.
 Note: relvel5 (of type vel) was used in a relvel definition (3)
 Let typical_point5 be a new typical_point
  p is in the same place as pt2 during bndy6.
  p is in the same place as pt5 during bndy5.
  p is in the same place as typical_point5 during period6.
Pullins in schema motion(p,s0,pt1,threaded,period1)
```

```
 Let relvel6 be the relvel of p in direction 0 relative to earth.
 Note: relvel6 (of type vel) was used in a relvel definition (3)
 Let typical_point6 be a new typical_point
  p is in the same place as pt2 during bndy1.
  p is in the same place as pt1 during initial.
  p is in the same place as typical_point6 during period1.
motion(p,s0,pt1,increased,period1) ok provided :
    positive(relvel3)
    real(relvel3)
    positive(relvel2)
    real(relvel2)

Attempting to solve for [relvel3,relvel2] in terms of [r0,h0]


I am now trying to solve for relvel3 without introducing any unknowns.

 Applicable formulae : [consvenersy-1,consvenersy-2,relvel,constvel,constaccel-1
 (try  consvenersy-1)
 Trying to apply strategy(consvenersy-1,situation(p,s4,pt4,period5))

 Equation-1 : s*(r0*(sin(90)-sin(180)))=relvel4^2/2-relvel3^2/2
 formed by applying : strategy(consvenersy-1,situation(p,s4,pt4,period5))

 Equation-1 rejected.

 Trying to apply strategy(consvenersy-1,situation(p,s3,pt3,period4))

 Equation-2 : s*(r0*(sin(0)-sin(90)))=relvel3^2/2-relvel2^2/2
 formed by applying : strategy(consvenersy-1,situation(p,s3,pt3,period4))

 This equation solves for relvel3.

[ No unknowns ]   Do you accept this equation ? yes.

 So now I must solve for [relvel2]
                   given [relvel3,r0,h0]

I am now trying to solve for relvel2 without introducing any unknowns.

 Applicable formulae : [consvenersy-1,consvenersy-2,relvel,constvel,constaccel-1
 (try  consvenersy-1)
 Trying to apply strategy(consvenersy-1,situation(p,s2,pt2,period3))

 Equation-3 : s*(r0*(sin(270)-sin(0)))=relvel2^2/2-relvel1^2/2
 formed by applying : strategy(consvenersy-1,situation(p,s2,pt2,period3))

 Equation-3 rejected.

 (try  consvenersy-2)
 Trying to apply strategy(consvenersy-2,situation(p,s3,pt3,period4))
 Trying to apply strategy(consvenersy-2,situation(p,s2,pt2,period3))
 (try  relvel)
 Trying to apply strategy(relvel,situation(p,earth,pt5,bndy3))
 Trying to apply strategy(relvel,situation(p,earth,pt4,bndy3))
 Trying to apply strategy(relvel,situation(p,earth,pt3,bndy3))
 Trying to apply strategy(relvel,situation(p,earth,pt2,bndy3))
 Trying to apply strategy(relvel,situation(p,earth,pt1,bndy3))
 (try  constvel)
 (try  constaccel-1)
```

```
 (try  constaccel-2)
 (try  constaccel-3)

No luck - I will now accept unknowns in solving for relvel2.

 Applicable formulae : [consvenersy-1,consvenersy-2,relvel,constvel,constaccel-1,co
 (try  consvenersy-1)
 Trying to apply strategy(consvenersy-1,situation(p,s2,pt2,period3))

 Equation-4 : s*(ra*(sin(270)-sin(0)))=relvel2^2/2-relvel1^2/2
 formed by applying : strategy(consvenersy-1,situation(p,s2,pt2,period3))

 This equation solves for relvel2 but introduces [relvel1].

[ Unknowns allowed ]   Do you accept this equation ? yes.

 So now I must solve for [relvel1]
                   given [relvel2,relvel3,ra,ha]

I am now trying to solve for relvel1 without introducing any unknowns.

 Applicable formulae : [consvenersy-1,consvenersy-2,relvel,constvel,constaccel-1,c
 (try  consvenersy-1)
 Trying to apply strategy(consvenersy-1,situation(p,s1,pt1,period2))

 Equation-5 : s*ha=relvel1^2/2-zero^2/2
 formed by applying : strategy(consvenersy-1,situation(p,s1,pt1,period2))

 This equation solves for relvel1.

[ No unknowns ]   Do you accept this equation ? yes.

 So now I must solve for []
                   given [relvel1,relvel2,relvel3,ra,ha]


Equations extracted :
    s*(ra*(sin(0)-sin(90)))=relvel3^2/2-relvel2^2/2
    s*(ra*(sin(270)-sin(0)))=relvel2^2/2-relvel1^2/2
    s*ha=relvel1^2/2-zero^2/2


yes
| ?- core     87552  (58368 lo-ses + 29184 hi-ses)
heap     53248 =  50797 in use +   2451 free
global    1187 =     16 in use +   1171 free
local     1024 =     16 in use +   1008 free
trail      511 =      0 in use +    511 free
    2.14 sec. for 3 GCs gaining 82967 words
    1.46 sec. for 29 local shifts and 49 trail shifts
   65.02 sec. runtime
```

```
/* fly.prb */
/* A particle starts to move up the underside of an overhang,
with velocity va. Will it reach the top of the overhang?
What will it do? */
/* Alan Bundy 7.4.81 */

        cue pathsys(overhang,bottom,top,right,left).
        incline(overhang,dir,bottom).
        isa(particle,p).
        at(p,bottom,depart).
        vel(p,va,dir,depart).
        positive(va.)
        side(p,bottom,right,depart).

        probtype(roller_coaster).
        given(va).
        given(dir).

        goal :- qa(at(p,top,Mom) , true).
```

```
yes
| ?- input(fly).
Pullins in schema pathsys(overhang,bottom,top,right,left)
Pullins in schema linesys(path,overhang,[bottom,top])
 Let overhang be a new path
 Let bottom be a new point
 Let top be a new point
 Note: dir (of type angle) was used in a incline definition (2)
 Let p be a new particle
 Note: va (of type vel) was used in a relvel definition (3)
 Note: dir (of type angle) was used in a relvel definition (4)

fly problem read into data base.


yes
| ?- goal.
Pullins in schema timesys(_68,depart,_33)
 Let period1 be a new period
 Let depart be the bndy of period1 on the left.
 Let bndy1 be the bndy of period1 on the right.
 Let depart be a new moment
 Let bndy1 be a new moment
Pullins in schema timesys(period1,depart,_4370)

** ERROR Nfrec already recorded: cued(timesys(period1,depart,bndy1))
    ( continue after error )
Storing proviso : positive(va).
 in direction angle1 at top.
 Note: angle1 (of type angle) was used in a incline definition (2)
 Let relvel1 be the relvel of p in direction angle1 relative to earth.
 Note: relvel1 (of type vel) was used in a relvel definition (3)
 Note: angle1 (of type angle) was used in a relvel definition (4)
  oring proviso : real(relvel1).
p falls off overhang during period1

no
| ?- core     83456  (54272 lo-ses + 29184 hi-ses)
heap     49152 =  47310 in use +   1842 free
global    1187 =     16 in use +   1171 free
local     1024 =     16 in use +   1008 free
trail      511 =      0 in use +    511 free
    0.01 sec. for 1 GCs gaining 361 words
    0.24 sec. for 16 local shifts and 22 trail shifts
    6.20 sec. runtime
```

```
/* drop.prb */
/* A particle is dropped from the top of a vertical wall.
Will it reach the bottom? */
/* Alan Bundy 7.4.81 */

        cue pathsys(wall,top,bottom,left,stline).
        incline(wall,270,top).
        isa(particle,p).
        at(p,top,depart).
        vel(p,zero,270,depart).
        side(p,top,left,depart).

        probtype(roller_coaster).


        goal :- qa(at(p,bottom,Mom) , true).
```

```
/* ramp.prb */
/* A ramp consists of a convex, circular slope joined smoothly
to a horizontal top piece. A particle is projected up
the ramp with a velocity va. What is the least value of va
such that the particle will reach the end of the ramp?
/* Alan Bundy 7.4.81 */

        isa(path,ramp),
        partition(ramp,[s1,s2]),
        cue pathsys(s1,pt1,pt2,left,right),
        cue pathsys(s2,pt2,pt3,hor,stline),
        cue linesys(path,ramp,[pt1,pt2,pt3]),
        circle(circ),
        radius(circ,ra),
        partition(circ,[s1,s3]),
        solid(s1),
        angle(s1,90,pt2),
        incline(s1,dir,pt1),
        mass(p,ma,initial),
        isa(particle,p),
        at(p,pt1,initial),
        vel(p,va,dir,initial),
        positive(va),
        side(p,pt1,left,initial),

        probtype(roller_coaster),
        given(ra),
        given(ma),
        given(dir),
        given(va),

        goal :- as( at(p,pt3,Mom) , min(va,ANS) ),
```

```
:RRRRRRR       AAAAAA     MM       MM   PPPPPPPP                  SSSSSSSS    000000
:RRRRRRR       AAAAAA     MM       MM   PPPPPPPP                  SSSSSSSS    000000
:R       RR   AA       AA   MMMM   MMMM   PP       PP            SS          00      00
:R       RR   AA       AA   MMMM   MMMM   PP       PP            SS          00      00
:R       RR   AA       AA   MM   MM   MM   PP       PP            SS          00      00
:R       RR   AA       AA   MM   MM   MM   PP       PP            SS          00      00
:RRRRRRR      AA       AA   MM       MM   PPPPPPPP                SSSSSS      00      00
:RRRRRRR      AA       AA   MM       MM   PPPPPPPP                SSSSSS      00      00
:R   RR       AAAAAAAAAA   MM       MM   PP                            SS    00      00
:R   RR       AAAAAAAAAA   MM       MM   PP                            SS    00      00
:R     RR     AA       AA   MM       MM   PP                 ....       SS    00      00
:R     RR     AA       AA   MM       MM   PP                 ....       SS    00      00
:R       RR   AA       AA   MM       MM   PP                 ....  SSSSSSSS      000000
:R       RR   AA       AA   MM       MM   PP                 ....  SSSSSSSS      000000


)4     44       000000       000000                  44     44       000000    5555555555
)4     44       000000       000000                  44     44       000000    5555555555
)4     44     00       00   00       00              44     44     00       00  55
)4     44     00       00   00       00              44     44     00       00  55
)4     44     00     0000   00     0000              44     44     00     0000  555555
)4     44     00     0000   00     0000              44     44     00     0000  555555
)4444444444  00   00   00   00   00   00            4444444444  00   00   00        55
)4444444444  00   00   00   00   00   00            4444444444  00   00   00        55
       44     0000     00   0000     00                    44     0000     00          55
       44     0000     00   0000     00                    44     0000     00          55
       44     00       00   00       00       ,,,,         44     00       00  55      55
       44     00       00   00       00       ,,,,         44     00       00  55      55
       44       000000       000000           ,,           44       000000      555555
       44       000000       000000           ,,           44       000000      555555


:BBB  U    U N    N DDDD   Y    Y                  H    H PPPP   SSSS
:    B U    U N    N D   D Y    Y                  H    H P   P S
:    B U    U NN   N D   D  Y Y                   H    H P   P S
:BBB  U    U N N  N D   D    Y                    HHHHH PPPP   SSS
:    B U    U N  NN D   D    Y                    H    H P         S
:    B U    U N    N D   D    Y                    H    H P         S
:BBB  UUUUU N    N DDDD     Y                     H    H P     SSSS


 *START* User BUNDY     HPS [400,405]  Job  RAMP Seq, 8122 Date 30-Apr-81 16:37:37
File: DSKA:RAMP,SOL<005>[400,405,COAST] Created: 30-Apr-81 16:33:56 Printed: 30-Ap
QUEUE Switches:  /FILE:ASCII /COPIES:1 /SPACING:1 /LIMIT:57 /FORMS:NORMAL
```

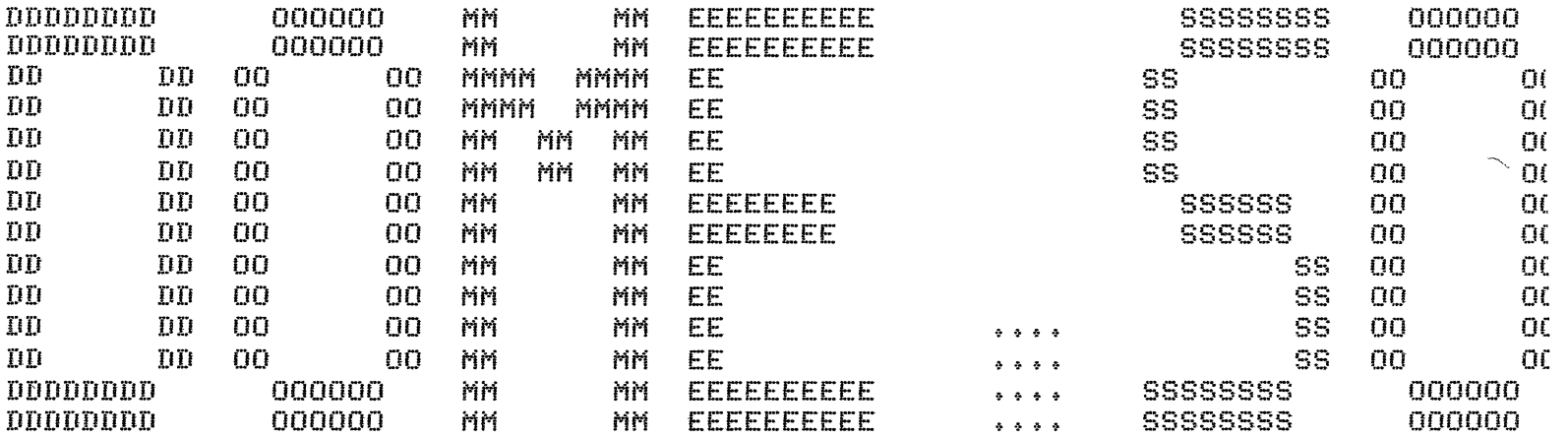
```
es
  ?- input(ramp).
 Let ramp be a new path
ullins in schema pathsys(s1,pt1,pt2,left,risht)
ullins in schema linesys(path,s1,[pt1,pt2])
 Let s1 be a new path
 Let pt1 be a new point
 Let pt2 be a new point
ullins in schema pathsys(s2,pt2,pt3,hor,stline)
ullins in schema linesys(path,s2,[pt2,pt3])
 Let s2 be a new path
 Let pt3 be a new point
ullins in schema linesys(path,ramp,[pt1,pt2,pt3])
 Let circ be a new circle
 Note: ra (of type lensth) was used in a radius definition (2)
 Note: dir (of type angle) was used in a incline definition (2)
 Note: ma (of type mass) was used in a mass definition (2)
 Let p be a new particle
 Note: va (of type vel) was used in a relvel definition (3)
 Note: dir (of type angle) was used in a relvel definition (4)

amp problem read into data base.


es
  ?- goal.
ullins in schema timesys(_69,initial,_33)
 Let period1 be a new period
 Let initial be the bndy of period1 on the left.
 Let bndy1 be the bndy of period1 on the risht.
 Let initial be a new moment
 Let bndy1 be a new moment
ullins in schema timesys(period1,initial,_4371)

% ERROR Nfrec already recorded: cued(timesys(period1,initial,bndy1))
   ( continue after error )
 Let period2 be a new period
 Let period3 be a new period
ivide period1 into :
    period2
    period3
ullins in schema timesys(period2,initial,_7325)
 Let initial be the bndy of period2 on the left.
 Let bndy2 be the bndy of period2 on the risht.
 Let bndy2 be a new moment
storins proviso : positive(va).
 Let typical_point1 be a new typical_point
 in direction angle1 at typical_point1.
 Note: angle1 (of type angle) was used in a angle definition (2)
 Let reaction1 be the reaction of s1 in direction angle1.
 Note: reaction1 (of type force) was used in a reaction definition (3)
 Note: angle1 (of type angle) was used in a reaction definition (4)
storins proviso : non_nes(reaction1).
ullins in schema motion(p,s1,pt1,left,period2)
 Let relvel1 be the relvel of p in direction 180 relative to earth.
 Note: relvel1 (of type vel) was used in a relvel definition (3)
  p is in the same place as pt2 durins bndy2.
  p is in the same place as pt1 durins initial.
  p is in the same place as typical_point1 durins period2.
```

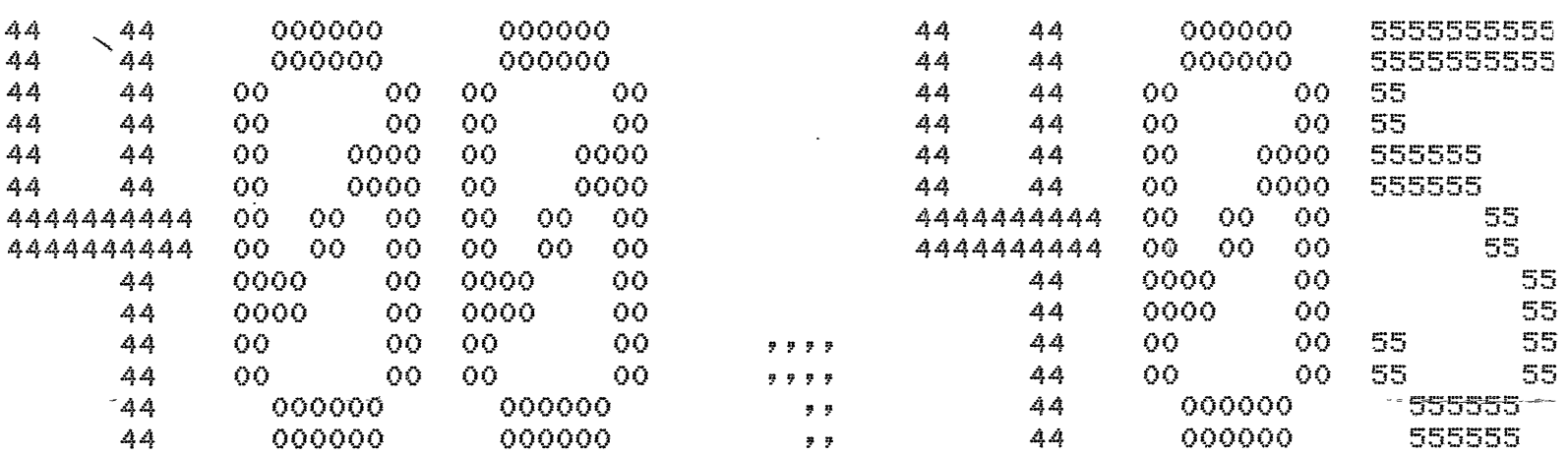
```
Pullins in schema timesys(period3,bndy2,_23841)
 Let bndy3 be the bndy of period3 on the risht.
 Let bndy3 be a new moment
 Let relvel2 be the relvel of p in direction 0 relative to earth.
 Note: relvel2 (of type vel) was used in a relvel definition (3)
Storins proviso : positive(relvel2).
Pullins in schema motion(p,s2,pt2,left,period3)
 Let relvel3 be the relvel of p in direction 0 relative to earth.
 Note: relvel3 (of type vel) was used in a relvel definition (3)
 Let typical_point2 be a new typical_point
  p is in the same place as pt3 durins bndy3.
  p is in the same place as pt2 durins bndy2.
  p is in the same place as typical_point2 durins period3.
Pullins in schema motion(p,ramp,pt1,left,period1)
 Let relvel4 be the relvel of p in direction 0 relative to earth.
 Note: relvel4 (of type vel) was used in a relvel definition (3)
 Let typical_point3 be a new typical_point
  p is in the same place as pt3 durins bndy1.
  p is in the same place as pt1 durins initial.
  p is in the same place as typical_point3 durins period1.
at(p,pt3,bndy1) ok provided :
    positive(relvel2)
    non_nes(reaction1)
    positive(va)

Attemptins to solve for [relvel2,reaction1] in terms of [ra,ma,dir,va]


I am now tryins to solve for relvel2 without introducins any unknowns.

 Applicable formulae : [consvenersy-1,consvenersy-2,relvel,constvel,constaccel-1,c
 (try  consvenersy-1)
 Tryins to apply strates(consvenersy-1,situation(p,s2,pt2,period3))

 Equation-1 : s*zero=relvel3^2/2-relvel2^2/2
 formed by applyins : strates(consvenersy-1,situation(p,s2,pt2,period3))

 Equation-1 rejected.

 Tryins to apply strates(consvenersy-1,situation(p,s1,pt1,period2))

 Equation-2 : s*(ra*(sin(dir--90)-sin(90)))=relvel2^2/2-va^2/2
 formed by applyins : strates(consvenersy-1,situation(p,s1,pt1,period2))

 This equation solves for relvel2.

  No unknowns ]   Do you accept this equation ? yes.

 So now I must solve for [reaction1]
                    siven [relvel2,ra,ma,dir,va]

  am now tryins to solve for reaction1 without introducins any unknowns.

 Applicable formulae : [moments,resolve]
 (try  moments)
 (try  resolve)
 Tryins to apply strates(resolve,situation(particle,p,[typical_point1,p],ansle1,pe

 o luck - I will now accept unknowns in solvins for reaction1.
```

```
Applicable formulae : [moments,resolve]
(try  moments)
(try  resolve)
Trying to apply strategy(resolve,situation(particle,p,[typical_point1,p],angle1,p
Let relvel5 be the relvel of p in direction angle2 relative to earth.
Note: relvel5 (of type vel) was used in a relvel definition (3)
Note: angle2 (of type angle) was used in a relvel definition (4)

Equation-3 : reaction1*1+0+(ma*g*cos(angle1--270)+0)=ma*((-relvel5^2)*1/ra)
formed by applying : strategy(resolve,situation(particle,p,[typical_point1,p],ang

This equation solves for reaction1 but introduces [angle1,relvel5].

: Unknowns allowed ]   Do you accept this equation ? yes.

So now I must solve for [angle1,relvel5]
                  given [reaction1,relvel2,ra,ma,dir,va]

: am now trying to solve for angle1 without introducing any unknowns.

Applicable formulae : []

Applicable formulae : []

lo luck - I will now accept unknowns in solving for angle1.

Applicable formulae : []

Applicable formulae : []

: shall assume that angle1 can be eliminated!

: am now trying to solve for relvel5 without introducing any unknowns.

Applicable formulae : [consvenersy-1,consvenersy-2,relvel,constvel,constaccel-1,c
(try  consvenersy-1)
(try  consvenersy-2)
Trying to apply strategy(consvenersy-2,situation(p,s1,pt1,period2))

Equation-4 : g*(ra*(sin(dir--90)-sin(angle1)))=relvel5^2/2-va^2/2
formed by applying : strategy(consvenersy-2,situation(p,s1,pt1,period2))

Equation-4 rejected.

(try  relvel)
Trying to apply strategy(relvel,situation(p,earth,pt3,period2))
Trying to apply strategy(relvel,situation(p,earth,pt2,period2))
Trying to apply strategy(relvel,situation(p,earth,pt1,period2))
(try  constvel)
(try  constaccel-1)
(try  constaccel-2)
(try  constaccel-3)

lo luck - I will now accept unknowns in solving for relvel5.

Applicable formulae : [consvenersy-1,consvenersy-2,relvel,constvel,constaccel-1,c
(try  consvenersy-1)
(try  consvenersy-2)
Trying to apply strategy(consvenersy-2,situation(p,s1,pt1,period2))
```

```
Equation-5 : s*(ra*(sin(dir--90)-sin(angle1)))=relvel5^2/2-va^2/2
formed by applying : strategy(consvenergy-2,situation(p,s1,pt1,period2))

This equation solves for relvel5 but introduces [angle1].

[ Unknowns allowed ]   Do you accept this equation ? yes.

So now I must solve for [angle1]
                  given [relvel5,reaction1,relvel2,ra,ma,dir,va]

I am now trying to solve for angle1 without introducing any unknowns.

Applicable formulae : []

Applicable formulae : []

No luck - I will now accept unknowns in solving for angle1.

Applicable formulae : []

Applicable formulae : []

I shall assume that angle1 can be eliminated!

Equations extracted :
   s*(ra*(sin(dir--90)-sin(90)))=relvel2^2/2-va^2/2
   reaction1*1+0+(ma*s*cos(angle1--270)+0)=ma*((-relvel5^2)*1/ra)
   s*(ra*(sin(dir--90)-sin(angle1)))=relvel5^2/2-va^2/2

yes
  ?- core      84992  (55808 lo-ses + 29184 hi-ses)
heap      50688 =  48498 in use +   2190 free
global     1187 =     16 in use +   1171 free
local      1024 =     16 in use +   1008 free
trail       511 =      0 in use +    511 free
    0.65 sec. for 2 GCs gaining 44469 words
    1.25 sec. for 30 local shifts and 45 trail shifts
   40.57 sec. runtime
```

```
/* dome.prb */
/* de Kleers Great Dome problem */
/* Alan Bundy 7.4.81 */

        circle(circle).
        radius(circle,rq).
        partition(circle,[dome,quad2,quad3,quad4]).
        solid(dome).
        cue pathsys(dome,top,bottom,left,risht).
        ansle(dome,90,top).
        ansle(dome,0,bottom).
        normal(dome,dir).
        isa(particle,p).
        mass(p,mq,period1).
        at(p,top,depart).
        vel(p,zero,0,depart).
        side(p,top,left,depart).
        nudse(p,depart).


        probtype(roller_coaster).
        siven(rq).
        siven(mq).
        siven(dir).

        soal :- qa(motion(p,dome,top,left,period1) , min(dir,ANS)).
```



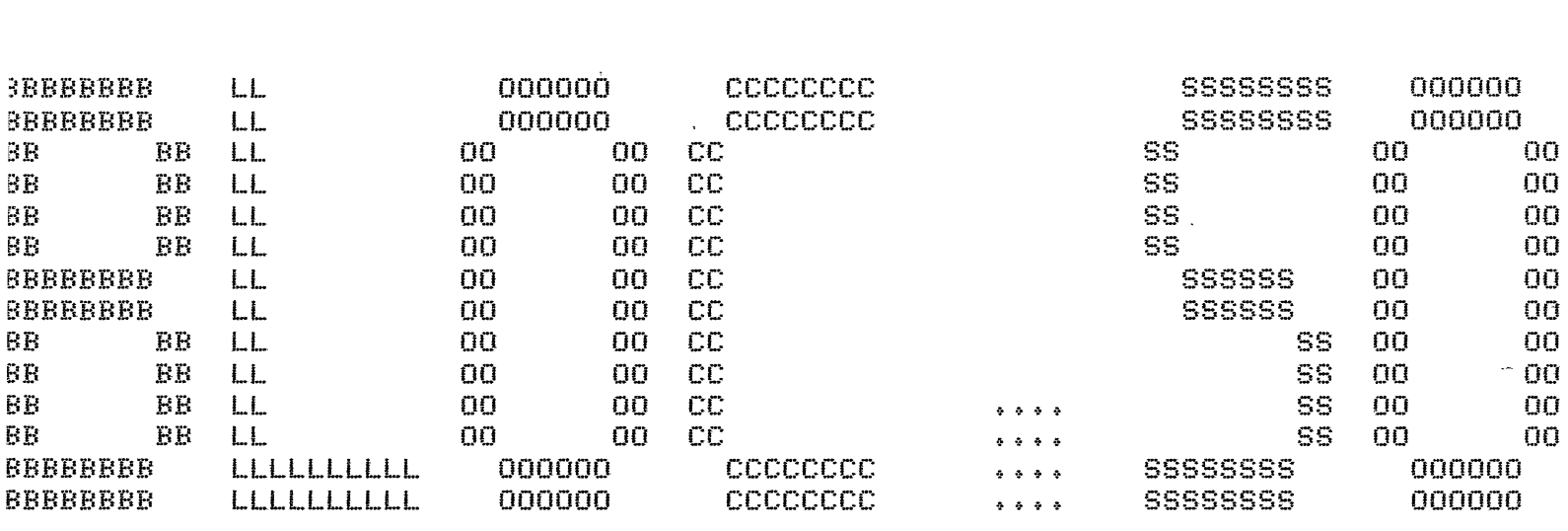


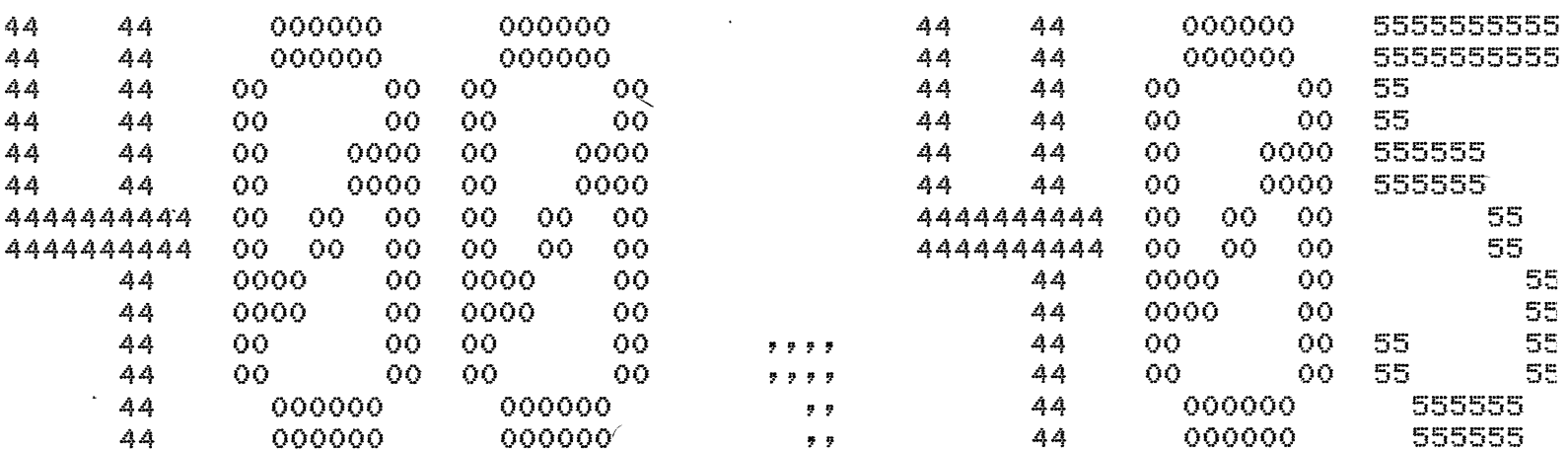
START User BUNDY HPS [400,405] Job DOME Seq. 8121 Date 30-Apr-81 16:31:34
File: DSKA:DOME.SOL<005>[400,405,COAST] Created: 30-Apr-81 15:25:04 Printed: 30-Apr
QUEUE Switches: /FILE:ASCII /COPIES:1 /SPACING:1 /LIMIT:54 /FORMS:NORMAL

```
es
  ?- input(dome).
 Let circle be a new circle
 Note: ra (of type length) was used in a radius definition (2)
ullins in schema pathsys(dome,top,bottom,left,right)
ullins in schema linesys(path,dome,[top,bottom])
 Let dome be a new path
 Let top be a new point
 Let bottom be a new point
 Let typical_point1 be a new typical_point
 Note: dir (of type angle) was used in a angle definition (2)
 Let p be a new particle
 Note: ma (of type mass) was used in a mass definition (2)
 Note: zero (of type vel) was used in a relvel definition (3)

ome problem read into data base.


es
  ?- goal.
ullins in schema timesys(period1,depart,_67)
 Let period1 be a new period
 Let depart be the bndy of period1 on the left.
 Let bndy1 be the bndy of period1 on the right.
 Let depart be a new moment
 Let bndy1 be a new moment
 Let reaction1 be the reaction of dome in direction dir.
 Note: reaction1 (of type force) was used in a reaction definition (3)
 Note: dir (of type angle) was used in a reaction definition (4)
storing proviso : non_neg(reaction1).
ullins in schema motion(p,dome,top,left,period1)
 Let relvel1 be the relvel of p in direction 90 relative to earth.
 Note: relvel1 (of type vel) was used in a relvel definition (3)
  p is in the same place as bottom during bndy1.
  p is in the same place as top during depart.
  p is in the same place as typical_point1 during period1.
otion(p,dome,top,left,period1) ok provided :
    non_neg(reaction1)

ttemptins to solve for [reaction1] in terms of [ra,ma,dir]


I am now trying to solve for reaction1 without introducing any unknowns.

 Applicable formulae : [moments,resolve]
 (try  moments)
 (try  resolve)
 Trying to apply strategy(resolve,situation(particle,p,[typical_point1,p],dir,peri

No luck - I will now accept unknowns in solving for reaction1.

 Applicable formulae : [moments,resolve]
 (try  moments)
 (try  resolve)
 Trying to apply strategy(resolve,situation(particle,p,[typical_point1,p],dir,peri
 Let relvel2 be the relvel of p in direction angle1 relative to earth.
 Note: relvel2 (of type vel) was used in a relvel definition (3)
 Note: angle1 (of type angle) was used in a relvel definition (4)
```

```
 Equation-1 : reaction1*1+0+(ma*s*cos(dir--270)+0)=ma*((-relvel2^2)*1/ra)
 formed by applying : strategy(resolve,situation(particle,p,[typical_point1,p],di

 This equation solves for reaction1 but introduces [relvel2].

[ Unknowns allowed ]   Do you accept this equation ? yes.

 So now I must solve for [relvel2]
                   given [reaction1,ra,ma,dir]

I am now trying to solve for relvel2 without introducing any unknowns.

 Applicable formulae : [consvenersy-1,consvenersy-2,relvel,constvel,constaccel-1,(
 (try  consvenersy-1)
 (try  consvenersy-2)
 Trying to apply strategy(consvenersy-2,situation(p,dome,top,period1))

 Equation-2 : s*(ra*(sin(90)-sin(dir)))=relvel2^2/2-zero^2/2
 formed by applying : strategy(consvenersy-2,situation(p,dome,top,period1))

 This equation solves for relvel2.

[ No unknowns ]   Do you accept this equation ? yes.

 So now I must solve for []
                   given [relvel2,reaction1,ra,ma,dir]


Equations extracted :
    reaction1*1+0+(ma*s*cos(dir--270)+0)=ma*((-relvel2^2)*1/ra)
    s*(ra*(sin(90)-sin(dir)))=relvel2^2/2-zero^2/2


es
  ?- core      83968  (54784 lo-ses + 29184 hi-ses)
esp      49664 =  47735 in use +    1929 free
lobal     1187 =     16 in use +    1171 free
ocal      1024 =     16 in use +    1008 free
rail       511 =      0 in use +     511 free
    0.01 sec. for 1 GCs gaining 361 words
    0.40 sec. for 23 local shifts and 33 trail shifts
   16.74 sec. runtime
```

```
/* bloc.prb */
/* de Kleers Slidins Block problem */
/* Alan Bundy 7.4.81 */

        isa(path,s0).
        partition(s0,[s1,s2,s3]).
        cue pathsys(s1,pt1,pt2,left,left).
        cue pathsys(s2,pt2,pt3,risht,left).
        cue pathsys(s3,pt3,pt4,risht,stline).
        cue linesys(path,s0,[pt1,pt4]).
        drop(pt1,pt2,he1).
        drop(pt2,pt3,he2).
        sround(s3,le).
        tansent(s3,ans).
        isa(particle,p).
        at(p,pt1,initial).
        vel(p,zero,dirs,initial).
        side(p,pt1,left,initial).

        probtype(roller_coaster).
        siven(he1).
        siven(he2).
        siven(le).
        siven(ans).

        soal :- as( at(p,pt4,Mom) , solveinea(X,Val) ).
```





START User BUNDY HPS [400,405] Job BLOC Seq. 3252 Date 05-May-81 15:18:45
File: DSKA:BLOC.SOL<005>[400,405,COAST] Created: 05-May-81 15:15:00 Printed: 05-M
QUEUE Switches: /FILE:ASCII /COPIES:1 /SPACING:1 /LIMIT:57 /FORMS:NORMAL

```
yes
| ?- input(bloc).
 Let s0 be a new path
Pullins in schema pathsys(s1,pt1,pt2,left,left)
Pullins in schema linesys(path,s1,[pt1,pt2])
 Let s1 be a new path
 Let pt1 be a new point
 Let pt2 be a new point
Pullins in schema pathsys(s2,pt2,pt3,right,left)
Pullins in schema linesys(path,s2,[pt2,pt3])
 Let s2 be a new path
 Let pt3 be a new point
Pullins in schema pathsys(s3,pt3,pt4,right,stline)
Pullins in schema linesys(path,s3,[pt3,pt4])
 Let s3 be a new path
 Let pt4 be a new point
Pullins in schema linesys(path,s0,[pt1,pt4])
 Note: ha1 (of type length) was used in a drop definition (3)
 Note: ha2 (of type length) was used in a drop definition (3)
 Note: la (of type length) was used in a ground definition (2)
 Let typical_point1 be a new typical_point
 Note: ang (of type angle) was used in a incline definition (2)
 Let p be a new particle
 Note: zero (of type vel) was used in a relvel definition (3)
 Note: dira (of type angle) was used in a relvel definition (4)

bloc problem read into data base.


yes
| ?- goal.
Pullins in schema timesys(_70,initial,_33)
 Let period1 be a new period
 Let initial be the bndy of period1 on the left.
 Let bndy1 be the bndy of period1 on the right.
 Let initial be a new moment
 Let bndy1 be a new moment
Pullins in schema timesys(period1,initial,_4372)

** ERROR Nfrec already recorded: cued(timesys(period1,initial,bndy1))
    ( continue after error )
 Let period2 be a new period
 Let period3 be a new period
 Let period4 be a new period
Divide period1 into :
    period2
    period3
    period4
Pullins in schema timesys(period2,initial,_7497)
 Let initial be the bndy of period2 on the left.
 Let bndy2 be the bndy of period2 on the right.
 Let bndy2 be a new moment
Pullins in schema motion(p,s1,pt1,left,period2)
 in direction angle1 at pt2.
 Note: angle1 (of type angle) was used in a incline definition (2)
 Let relvel1 be the relvel of p in direction angle1 relative to earth.
 Note: relvel1 (of type vel) was used in a relvel definition (3)
 Note: angle1 (of type angle) was used in a relvel definition (4)
 Let typical_point2 be a new typical_point
```

```
  P is in the same place as Pt2 during bndy2.
  P is in the same place as Pt1 during initial.
  P is in the same place as typical_point2 during period2.
Pulling in schema timesys(period3,bndy2,_20478)
 Let bndy3 be the bndy of period3 on the right.
 Let bndy3 be a new moment
Condition: positive(relvel1) holds
 Let relvel2 be the relvel of P in direction ang relative to earth.
 Note: relvel2 (of type vel) was used in a relvel definition (3)
 Note: ang (of type angle) was used in a relvel definition (4)
Storing proviso : real(relvel2).
Pulling in schema motion(P,s2,Pt2,left,period3)
 Let typical_point3 be a new typical_point
  P is in the same place as Pt3 during bndy3.
  P is in the same place as Pt2 during bndy2.
  P is in the same place as typical_point3 during period3.
Pulling in schema timesys(period4,bndy3,_39050)
 Let bndy4 be the bndy of period4 on the right.
 Let bndy4 be a new moment
Storing proviso : positive(relvel2).
 Let relvel3 be the relvel of P in direction ang relative to earth.
 Note: relvel3 (of type vel) was used in a relvel definition (3)
Storing proviso : real(relvel3).
Pulling in schema motion(P,s3,Pt3,left,period4)
  P is in the same place as Pt4 during bndy4.
  P is in the same place as Pt3 during bndy3.
  P is in the same place as typical_point1 during period4.
Pulling in schema motion(P,s0,Pt1,left,period1)
 Let relvel4 be the relvel of P in direction ang relative to earth.
 Note: relvel4 (of type vel) was used in a relvel definition (3)
 Let typical_point4 be a new typical_point
  P is in the same place as Pt4 during bndy1.
  P is in the same place as Pt1 during initial.
  P is in the same place as typical_point4 during period1.
at(P,Pt4,bndy1) ok provided :
   real(relvel3)
   positive(relvel2)
   real(relvel2)

Attempting to solve for [relvel3,relvel2] in terms of [ha1,ha2,la,ang]


I am now trying to solve for relvel3 without introducing any unknowns.

 Applicable formulae : [consvenersy,relvel,constvel,constaccel-1,constaccel-2,cons
 (try  consvenersy)
 Trying to apply strategy(consvenersy,situation(P,Pt3,Pt4,bndy3,bndy4))

 Equation-1 : s*(-la*tan(ang))=relvel3^2/2-relvel2^2/2
 formed by applying : strategy(consvenersy,situation(P,Pt3,Pt4,bndy3,bndy4))

 This equation solves for relvel3.

[ No unknowns ]   Do you accept this equation ? yes.

 So now I must solve for [relvel2]
                   given [relvel3,ha1,ha2,la,ang]

am now trying to solve for relvel2 without introducing any unknowns.
```

```
Applicable formulae : [consvenersy,relvel,constvel,constaccel-1,constaccel-2,con
(try  consvenersy)
Trying to apply strategy(consvenersy,situation(p,pt2,pt3,bndy2,bndy3))

Equation-2 : s*ha2=relvel2^2/2-relvel1^2/2
formed by applying : strategy(consvenersy,situation(p,pt2,pt3,bndy2,bndy3))

Equation-2 rejected.

(try  relvel)
Trying to apply strategy(relvel,situation(p,earth,pt4,bndy3))
Trying to apply strategy(relvel,situation(p,earth,pt3,bndy3))
Trying to apply strategy(relvel,situation(p,earth,pt2,bndy3))
Trying to apply strategy(relvel,situation(p,earth,pt1,bndy3))
(try  constvel)
(try  constaccel-1)
(try  constaccel-2)
(try  constaccel-3)

No luck - I will now accept unknowns in solving for relvel2.

Applicable formulae : [consvenersy,relvel,constvel,constaccel-1,constaccel-2,con
(try  consvenersy)
Trying to apply strategy(consvenersy,situation(p,pt2,pt3,bndy2,bndy3))

Equation-3 : s*ha2=relvel2^2/2-relvel1^2/2
formed by applying : strategy(consvenersy,situation(p,pt2,pt3,bndy2,bndy3))

This equation solves for relvel2 but introduces [relvel1].

[ Unknowns allowed ]   Do you accept this equation ? yes.

So now I must solve for [relvel1]
                  given [relvel2,relvel3,ha1,ha2,la,ans]

I am now trying to solve for relvel1 without introducing any unknowns.

Applicable formulae : [consvenersy,relvel,constvel,constaccel-1,constaccel-2,cc
(try  consvenersy)
Trying to apply strategy(consvenersy,situation(p,pt1,pt2,initial,bndy2))

Equation-4 : s*ha1=relvel1^2/2-zero^2/2
formed by applying : strategy(consvenersy,situation(p,pt1,pt2,initial,bndy2))

This equation solves for relvel1.

[ No unknowns ]   Do you accept this equation ? yes.

So now I must solve for []
                  given [relvel1,relvel2,relvel3,ha1,ha2,la,ans]

Equations extracted :
   s*(-la*tan(ans))=relvel3^2/2-relvel2^2/2
   s*ha2=relvel2^2/2-relvel1^2/2
   s*ha1=relvel1^2/2-zero^2/2

yes
! ?- core      84992  (55808 lo-segs + 29184 hi-segs)
```

```
Heap      50688 =  48574 in use +   2114 free
Global     1187 =     16 in use +   1171 free
Local      1024 =     16 in use +   1008 free
Trail       511 =      0 in use +    511 free
    0.90 sec. for 2 GCs gaining 43917 words
    1.16 sec. for 30 local shifts and 47 trail shifts
   37.46 sec. runtime
```

```
/* loop.prb */
/* de Kleers Loop the loop problem */
/* Alan Bundy 7.4.81 */

        partition(s0,[s1,s2,s3,s4,s5]).
        radius(circle1,ra).
        partition(circle1,[s2,s3,s4,s5]).
        cue linesys(path,s0,[pt1,pt2,pt3,pt4,pt5,pt2]).
        cue linesys(circle,circle1,[pt2,pt3,pt4,pt5,pt2]).
        cue pathsys(s1,pt1,pt2,left,left).
        cue pathsys(s2,pt2,pt3,right,left).
        cue pathsys(s3,pt4,pt3,left,right).
        cue pathsys(s4,pt5,pt4,right,right).
        cue pathsys(s5,pt5,pt2,left,left).
        angle(circle1,270,pt2).
        angle(circle1,0,pt3).
        angle(circle1,90,pt4).
        angle(circle1,180,pt5).
        drop(pt1,pt2,ha).
        isa(particle,p).
        at(p,pt1,initial).
        incline(s1,dir1,pt1).
        vel(p,zero,dir1,initial).
        side(p,pt1,left,initial).

        probtype(roller_coaster).
        given(ra).
        given(ha).

        goal :- ea( motion(p,s0,pt1,left,Per) , min(ha,ANS) ).
```

```
LL              000000      000000    PPPPPPPP                  SSSSSSSS    000000
LL              000000      000000    PPPPPPPP                  SSSSSSSS    000000
LL            00      00  00      00  PP      PP              SS          00      00
LL            00      00  00      00  PP      PP              SS          00      00
LL            00      00  00      00  PP      PP              SS          00      00
LL            00      00  00      00  PP      PP              SS          00      00
LL            00      00  00      00  PPPPPPPP                  SSSSSS  00      00
LL            00      00  00      00  PPPPPPPP                  SSSSSS  00      00
LL            00      00  00      00  PP                              SS  00      00
LL            00      00  00      00  PP                              SS  00      00
LL            00      00  00      00  PP                  ....        SS  00      00
LL            00      00  00      00  PP                  ....        SS  00      00
LLLLLLLLLL      000000      000000    PP                  ....    SSSSSSSS    000000
LLLLLLLLLL      000000      000000    PP                  ....    SSSSSSSS    000000


44    44        000000      000000                  44    44        000000    5555555555
44    44        000000      000000                  44    44        000000    5555555555
44    44      00      00  00      00                44    44      00      00  55
44    44      00      00  00      00                44    44      00      00  55
44    44      00    0000  00    0000                44    44      00    0000  555555
44    44      00    0000  00    0000                44    44      00    0000  555555
4444444444    00  00  00  00  00  00                4444444444    00  00  00        55
4444444444    00  00  00  00  00  00                4444444444    00  00  00        55
      44      0000    00  0000    00                      44      0000    00          55
      44      0000    00  0000    00                      44      0000    00          55
      44      00      00  00      00          ,,,,        44      00      00  55      55
      44      00      00  00      00          ,,,,        44      00      00  55      55
      44        000000      000000              ,,        44        000000      555555
      44        000000      000000              ,,        44        000000      555555


BBBB  U    U N    N DDDD   Y    Y                        H    H PPPP   SSSS
B   B U    U N    N D   D  Y    Y                       H    H P   P S
B   B U    U NN   N D   D   Y Y                        H    H P   P S
BBBB  U    U N N  N D   D    Y                         HHHHH PPPP   SSS
B   B U    U N  NN D   D    Y                         H    H P         S
B   B U    U N    N D   D    Y                         H    H P         S
BBBB  UUUUU N    N DDDD     Y                         H    H P     SSSS
```

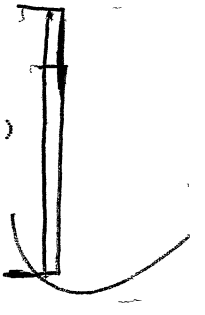
START User BUNDY HPS [400,405] Job LOOP Seq. 7037 Date 23-Apr-81 13:42:59
File: DSKA:LOOP.SOL<005>[400,405,COAST] Created: 23-Apr-81 13:32:03 Printed: 23-Ap
QUEUE Switches: /FILE:ASCII /COPIES:1 /SPACING:1 /LIMIT:60 /FORMS:NORMAL

```
yes
| ?- input(loop).
 Note: ra (of type length) was used in a radius definition (2)
Pullins in schema linesys(path,s0,[pt1,pt2,pt3,pt4,pt5,pt2])
 Let s0 be a new path
 Let pt1 be a new point
 Let pt2 be a new point
 Let pt3 be a new point
 Let pt4 be a new point
 Let pt5 be a new point

** ERROR Nfrec already recorded: point_of(path,s0,pt2)
   ( continue after error )
Pullins in schema linesys(circle,circle1,[pt2,pt3,pt4,pt5,pt2])
 Let circle1 be a new circle

** ERROR Nfrec already recorded: point_of(circle,circle1,pt2)
   ( continue after error )
Pullins in schema pathsys(s1,pt1,pt2,left,left)
Pullins in schema linesys(path,s1,[pt1,pt2])
 Let s1 be a new path
Pullins in schema pathsys(s2,pt2,pt3,right,left)
Pullins in schema linesys(path,s2,[pt2,pt3])
 Let s2 be a new path
Pullins in schema pathsys(s3,pt4,pt3,left,right)
Pullins in schema linesys(path,s3,[pt4,pt3])
 Let s3 be a new path
Pullins in schema pathsys(s4,pt5,pt4,right,right)
Pullins in schema linesys(path,s4,[pt5,pt4])
 Let s4 be a new path
Pullins in schema pathsys(s5,pt5,pt2,left,left)
Pullins in schema linesys(path,s5,[pt5,pt2])
 Let s5 be a new path
 Note: ha (of type length) was used in a drop definition (3)
 Let p be a new particle
 Note: dir1 (of type angle) was used in a incline definition (2)
 Note: zero (of type vel) was used in a relvel definition (3)
 Note: dir1 (of type angle) was used in a relvel definition (4)

loop problem read into data base.


yes
| ?- goal.
Pullins in schema timesys(_33,initial,_68)
 Let period1 be a new period
 Let initial be the bndy of period1 on the left.
 Let bndy1 be the bndy of period1 on the right.
 Let initial be a new moment
 Let bndy1 be a new moment
 Let period2 be a new period
 Let period3 be a new period
 Let period4 be a new period
 Let period5 be a new period
 Let period6 be a new period
Divide period1 into :
    period2
    period3
    period4
```

```
    period5
    period6
Pulling in schema timesys(period2,initial,_5413)
 Let initial be the bndy of period2 on the left.
 Let bndy2 be the bndy of period2 on the right.
 Let bndy2 be a new moment
Pulling in schema motion(p,s1,pt1,left,period2)
 Let relvel1 be the relvel of p in direction 0 relative to earth.
 Note: relvel1 (of type vel) was used in a relvel definition (3)
 Let typical_point1 be a new typical_point
  p is in the same place as pt2 during bndy2.
  p is in the same place as pt1 during initial.
  p is in the same place as typical_point1 during period2.
Pulling in schema timesys(period3,bndy2,_20288)
 Let bndy3 be the bndy of period3 on the right.
 Let bndy3 be a new moment
condition: positive(relvel1) holds
 Let relvel2 be the relvel of p in direction 90 relative to earth.
 Note: relvel2 (of type vel) was used in a relvel definition (3)
Storing proviso : real(relvel2).
Pulling in schema motion(p,s2,pt2,left,period3)
 Let typical_point2 be a new typical_point
  p is in the same place as pt3 during bndy3.
  p is in the same place as pt2 during bndy2.
  p is in the same place as typical_point2 during period3.
Pulling in schema timesys(period4,bndy3,_5041)
 Let bndy4 be the bndy of period4 on the right.
 Let bndy4 be a new moment
Storing proviso : positive(relvel2).
 Let relvel3 be the relvel of p in direction 180 relative to earth.
 Note: relvel3 (of type vel) was used in a relvel definition (3)
Storing proviso : real(relvel3).
 Let typical_point3 be a new typical_point
 in direction angle1 at typical_point3.
 Note: angle1 (of type angle) was used in a angle definition (2)
 Let relvel4 be the relvel of p in direction angle1++90 relative to earth.
 Note: relvel4 (of type vel) was used in a relvel definition (3)
 Note: angle1++90 (of type angle) was used in a relvel definition (4)
** Error: evaluate(s)
Storing proviso : relvel4^2*sin(angle1)>=ro*s.
Pulling in schema motion(p,s3,pt3,left,period4)
  p is in the same place as pt4 during bndy4.
  p is in the same place as pt3 during bndy3.
  p is in the same place as typical_point3 during period4.
Pulling in schema timesys(period5,bndy4,_30194)
 Let bndy5 be the bndy of period5 on the right.
 Let bndy5 be a new moment
Storing proviso : positive(relvel3).
 Let typical_point4 be a new typical_point
 in direction angle2 at typical_point4.
 Note: angle2 (of type angle) was used in a angle definition (2)
 Let relvel5 be the relvel of p in direction angle2++90 relative to earth.
 Note: relvel5 (of type vel) was used in a relvel definition (3)
 Note: angle2++90 (of type angle) was used in a relvel definition (4)
** Error: evaluate(s)
Storing proviso : relvel5^2*sin(angle2)>=ro*s.
Pulling in schema motion(p,s4,pt4,left,period5)
 Let relvel6 be the relvel of p in direction 270 relative to earth.
 Note: relvel6 (of type vel) was used in a relvel definition (3)
  p is in the same place as pt5 during bndy5.
```

```
  P is in the same place as pt4 during bndy4.
  P is in the same place as typical_point4 during period5.
Pulling in schema timesys(period6,bndy5,_13406)
 Let bndy6 be the bndy of period6 on the right.
 Let bndy6 be a new moment
condition: positive(relvel6) holds
Pulling in schema motion(P,s5,pt5,left,period6)
 Let relvel7 be the relvel of P in direction 0 relative to earth.
 Note: relvel7 (of type vel) was used in a relvel definition (3)
 Let typical_point5 be a new typical_point
  P is in the same place as pt2 during bndy6.
  P is in the same place as pt5 during bndy5.
  P is in the same place as typical_point5 during period6.
Pulling in schema motion(P,s0,pt1,left,period1)
 Let relvel8 be the relvel of P in direction 0 relative to earth.
 Note: relvel8 (of type vel) was used in a relvel definition (3)
 Let typical_point6 be a new typical_point
  P is in the same place as pt2 during bndy1.
  P is in the same place as pt1 during initial.
  P is in the same place as typical_point6 during period1.
motion(P,s0,pt1,left,period1) ok provided :
    relvel5^2*sin(angle2)>=ra*s
    positive(relvel3)
    relvel4^2*sin(angle1)>=ra*s
    real(relvel3)
    positive(relvel2)
    real(relvel2)

Attempting to solve for [relvel5,angle2,relvel4,angle1,relvel3,relvel2] in terms o


I am now trying to solve for relvel5 without introducing any unknowns.

 Applicable formulae : [consvenersy-1,consvenersy-2,relvel,constvel,constaccel-1,c
 (try  consvenersy-1)
 (try  consvenersy-2)
 Trying to apply strategy(consvenersy-2,situation(P,s4,pt4,period5))

 Equation-1 : s*(ra*(sin(90)-sin(angle2)))=relvel5^2/2-relvel3^2/2
 formed by applying : strategy(consvenersy-2,situation(P,s4,pt4,period5))

 This equation solves for relvel5.

[ No unknowns ]   Do you accept this equation ?
!: yes.

 So now I must solve for [angle2,relvel4,angle1,relvel3,relvel2]
                    given [relvel5,ra,ha]

I am now trying to solve for angle2 without introducing any unknowns.

 Applicable formulae : []

No luck - I will now accept unknowns in solving for angle2.

 Applicable formulae : []

I shall assume that angle2 can be eliminated!

I am now trying to solve for relvel4 without introducing any unknowns.
```

```
 Applicable formulae : [consvenersy-1,consvenersy-2,relvel,constvel,constaccel-1,
 (try  consvenersy-1)
 (try  consvenersy-2)
 Trysins to apply stratesy(consvenersy-2,situation(p,s3,pt3,period4))

 Equation-2 : s*(ra*(sin(0)-sin(ansle1)))=relvel4^2/2-relvel2^2/2
 formed by applysins : stratesy(consvenersy-2,situation(p,s3,pt3,period4))

 This equation solves for relvel4.

[ No unknowns ]   Do you accept this equation ? yes.

 So now I must solve for [ansle1,relvel3,relvel2]
                   siven [relvel4,relvel5,ra,ha]

I am now trysins to solve for ansle1 without introducins any unknowns.

 Applicable formulae : []

No luck - I will now accept unknowns in solvins for ansle1.

 Applicable formulae : []

I shall assume that ansle1 can be eliminated!

I am now trysins to solve for relvel3 without introducins any unknowns.

 Applicable formulae : [consvenersy-1,consvenersy-2,relvel,constvel,constaccel-1,c
 (try  consvenersy-1)
 Trysins to apply stratesy(consvenersy-1,situation(p,s4,pt4,period5))

 Equation-3 : s*(ra*(sin(90)-sin(180)))=relvel6^2/2-relvel3^2/2
 formed by applysins : stratesy(consvenersy-1,situation(p,s4,pt4,period5))

 Equation-3 rejected.

 Trysins to apply stratesy(consvenersy-1,situation(p,s3,pt3,period4))

 Equation-4 : s*(ra*(sin(0)-sin(90)))=relvel3^2/2-relvel2^2/2
 formed by applysins : stratesy(consvenersy-1,situation(p,s3,pt3,period4))

 This equation solves for relvel3.

[ No unknowns ]   Do you accept this equation ? yes.

 So now I must solve for [relvel2]
                   siven [relvel3,relvel4,relvel5,ra,ha]

[ am now trysins to solve for relvel2 without introducins any unknowns.

 Applicable formulae : [consvenersy-1,consvenersy-2,relvel,constvel,constaccel-1,c
 (try  consvenersy-1)
 Trysins to apply stratesy(consvenersy-1,situation(p,s2,pt2,period3))

 Equation-5 : s*(ra*(sin(270)-sin(0)))=relvel2^2/2-relvel1^2/2
 formed by applysins : stratesy(consvenersy-1,situation(p,s2,pt2,period3))

 Equation-5 rejected.
```



```
 (try  consvenersy-2)
 Trying to apply strategy(consvenersy-2,situation(p,s2,pt2,period3))
 Let typical_point7 be a new typical_point
 (try  relvel)
 Trying to apply strategy(relvel,situation(p,earth,pt5,bndy3))
 Trying to apply strategy(relvel,situation(p,earth,pt4,bndy3))
 Trying to apply strategy(relvel,situation(p,earth,pt3,bndy3))
 Trying to apply strategy(relvel,situation(p,earth,pt2,bndy3))
 Trying to apply strategy(relvel,situation(p,earth,pt1,bndy3))
 (try  constvel)
 (try  constaccel-1)
 (try  constaccel-2)
 (try  constaccel-3)

No luck - I will now accept unknowns in solving for relvel2.

 Applicable formulae : [consvenersy-1,consvenersy-2,relvel,constvel,constaccel-1,
 (try  consvenersy-1)
 Trying to apply strategy(consvenersy-1,situation(p,s2,pt2,period3))

 Equation-6 : s*(ra*(sin(270)-sin(0)))=relvel2^2/2-relvel1^2/2
 formed by applying : strategy(consvenersy-1,situation(p,s2,pt2,period3))

 This equation solves for relvel2 but introduces [relvel1].

[ Unknowns allowed ]   Do you accept this equation ? yes.

 So now I must solve for [relvel1]
                  given [relvel2,relvel3,relvel4,relvel5,ra,ha]

I am now trying to solve for relvel1 without introducing any unknowns.

 Applicable formulae : [consvenersy-1,consvenersy-2,relvel,constvel,constaccel-1,
 (try  consvenersy-1)
 Trying to apply strategy(consvenersy-1,situation(p,s1,pt1,period2))

 Equation-7 : s*ha=relvel1^2/2-zero^2/2
 formed by applying : strategy(consvenersy-1,situation(p,s1,pt1,period2))

 This equation solves for relvel1.

[ No unknowns ]   Do you accept this equation ? yes.

 So now I must solve for []
                  given [relvel1,relvel2,relvel3,relvel4,relvel5,ra,ha]


Equations extracted :
    s*(ra*(sin(90)-sin(angle2)))=relvel5^2/2-relvel3^2/2
    s*(ra*(sin(0)-sin(angle1)))=relvel4^2/2-relvel2^2/2
    s*(ra*(sin(0)-sin(90)))=relvel3^2/2-relvel2^2/2
    s*(ra*(sin(270)-sin(0)))=relvel2^2/2-relvel1^2/2
    s*ha=relvel1^2/2-zero^2/2


yes
| ?- core     87552  (58368 lo-ses + 29184 hi-ses)
heap     53248 =  50913 in use +   2335 free
global    1187 =     16 in use +   1171 free
local     1024 =     16 in use +   1008 free
```

```
trail      511 =      0 in use +    511 free
    3.29 sec. for 4 GCs ssinins 115668 words
    1.45 sec. for 23 local shifts and 43 trail shifts
  107.45 sec. runtime
```

```
DDDDDDDD      OOOOOO    MM      MM  EEEEEEEEEE                SSSSSSSS    OOOOOO    LL
DDDDDDDDD     OOOOOO    MM      MM  EEEEEEEEEE                SSSSSSSS    OOOOOO    LL
DD      DD  OO      OO  MMMM  MMMM  EE                      SS          OO      OO  LL
DD      DD  OO      OO  MMMM  MMMM  EE                      SS          OO      OO  LL
DD      DD  OO      OO  MM  MM  MM  EE                      SS          OO      OO  LL
DD      DD  OO      OO  MM  MM  MM  EE                      SS          OO      OO  LL
DD      DD  OO      OO  MM      MM  EEEEEEEE                  SSSSSS    OO      OO  LL
DD      DD  OO      OO  MM      MM  EEEEEEEE                  SSSSSS    OO      OO  LL
DD      DD  OO      OO  MM      MM  EE                            SS    OO      OO  LL
DD      DD  OO      OO  MM      MM  EE                            SS    OO      OO  LL
DD      DD  OO      OO  MM      MM  EE                ....        SS    OO      OO  LL
DD      DD  OO      OO  MM      MM  EE                ....        SS    OO      OO  LL
DDDDDDDD      OOOOOO    MM      MM  EEEEEEEEEE        ....  SSSSSSSS      OOOOOO    LLLLLLLLL
DDDDDDDD      OOOOOO    MM      MM  EEEEEEEEEE        ....  SSSSSSSS      OOOOOO    LLLLLLLLL
```

```
44    44      000000      000000                  44    44      000000    5555555555
44    44      000000      000000                  44    44      000000    5555555555
44    44    00      00  00      00                44    44    00      00  55
44    44    00      00  00      00                44    44    00      00  55
44    44    00    0000  00    0000                44    44    00    0000  555555
44    44    00    0000  00    0000                44    44    00    0000  555555
4444444444  00  00  00  00  00  00                4444444444  00  00  00        55
4444444444  00  00  00  00  00  00                4444444444  00  00  00        55
      44    0000    00  0000    00                      44    0000    00          55
      44    0000    00  0000    00                      44    0000    00          55
      44    00      00  00      00      ,,,,            44    00      00  55      55
      44    00      00  00      00      ,,,,            44    00      00  55      55
      44      000000      000000          ,,            44      000000      555555
      44      000000      000000          ,,            44      000000      555555
```

```
BBBB  U   U N   N DDDD  Y   Y        AAA
B   B U   U NN  N D   D Y   Y       A   A
B   B U   U N N N D   D  Y Y        A   A
BBBB  U   U N N N D   D   Y         A   A
B   B U   U N  NN D   D   Y         AAAAA
B   B U   U N   N D   D   Y         A   A
BBBB  UUUUU N   N DDDD    Y         A   A
```

```
LPTSPL Version 6(344)  Running on TTY70
 *START* User BUNDY A [400,405] Job  DOME Seq. 1505 Date 14-Sep-78 14:40:07 Monitor ERCC 0106
Request created: 14-Sep-78  14:41:10
File: DSKA0:DOME.SOL[400,405] Created: 14-Sep-78 14:05:00 (155)  Printed: 14-Sep-78 14:41:04
QUEUE Switches:  /PRINT:ARROW /FILE:ASCI
I /COPIES:1 /SPACING:1 /LIMIT:30 /FORMS:NORMAL
File will be RENAMEd to (055) protection
```

```
! ?-dome.
trying-to-solve-(mass1*g*(3*sin(dir)-2))=0&true)
(mass1*g*(3*sin(dir)-2))=0&true)-simplifies-to
assuming-mass1-positive
mass1*g*(sin(dir)*3+-2))=0
these-conditions-simplify-to-(mass1*g*(sin(dir)*3+-2))=0)
isolating-dir-in-(mass1*g*(sin(dir)*3+-2))=0)
assuming-mass1-positive
isolating-dir-in-(-2+sin(dir)*3)=0*(g*mass1)↑-1)
isolating-dir-in-(sin(dir)*3)=0*(g*mass1)↑-1+(--2))
isolating-dir-in-(sin(dir))=(0*(g*mass1)↑-1+(--2))*3↑-1)
arcsin((0*(g*mass1)↑-1+(--2))*3↑-1)-simplifies-to
arcsin(3↑-1*2)
arcsin(3↑-1*2)-simplifies-to
arcsin(2*3↑-1)
isolating-dir-on-the-lhs-gives-(dir)=arcsin(2*3↑-1))
trying-to-find-maximum-of-(dir)=arcsin(2*3↑-1))
(dir)=arcsin(2*3↑-1))-dominates-the-other-inequalities
hence-minimum-value-of-dir-is-arcsin(2*3↑-1)
ans-is-arcsin(2*3↑-1)


yes
```

```
BBBBBBBB    LL            OOOOOO      CCCCCCCC            SSSSSSSS    OOOOOO    LL
BBBBBBBB    LL            OOOOOO      CCCCCCCC            SSSSSSSS    OOOOOO    LL
BB      BB  LL          OO      OO  CC                  SS          OO      OO  LL
BB      BB  LL          OO      OO  CC                  SS          OO      OO  LL
BB      BB  LL          OO      OO  CC                  SS          OO      OO  LL
BB      BB  LL          OO      OO  CC                  SS          OO      OO  LL
BBBBBBBB    LL          OO      OO  CC                    SSSSSS    OO      OO  LL
BBBBBBBB    LL          OO      OO  CC                    SSSSSS    OO      OO  LL
BB      BB  LL          OO      OO  CC                          SS  OO      OO  LL
BB      BB  LL          OO      OO  CC                          SS  OO      OO  LL
BB      BB  LL          OO      OO  CC              ....        SS  OO      OO  LL
BB      BB  LL          OO      OO  CC              ....        SS  OO      OO  LL
BBBBBBBB    LLLLLLLLLL    OOOOOO      CCCCCCCC        ....    SSSSSSSS    OOOOOO    LLLLLLLLLL
BBBBBBBB    LLLLLLLLLL    OOOOOO      CCCCCCCC        ....    SSSSSSSS    OOOOOO    LLLLLLLLLL
```

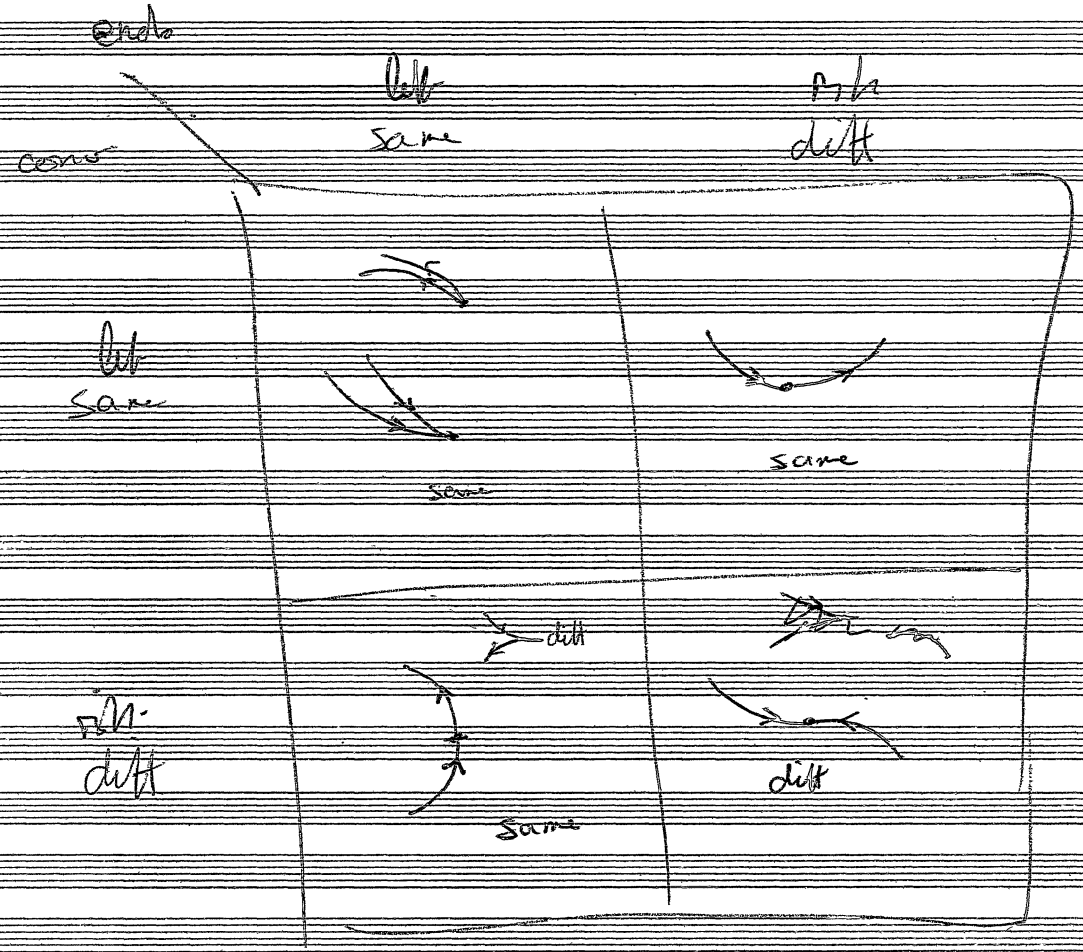
```
44    44      OOOOOO      OOOOOO                  44    44      OOOOOO    5555555555
44    44      OOOOOO      OOOOOO                  44    44      OOOOOO    5555555555
44    44    OO      OO  OO      OO                44    44    OO      OO  55
44    44    OO      OO  OO      OO                44    44    OO      OO  55
44    44    OO    OOOO  OO    OOOO                44    44    OO    OOOO  555555
44    44    OO    OOOO  OO    OOOO                44    44    OO    OOOO  555555
4444444444  OO  OO  OO  OO  OO  OO                4444444444  OO  OO  OO        55
4444444444  OO  OO  OO  OO  OO  OO                4444444444  OO  OO  OO        55
      44    OOOO    OO  OOOO    OO                      44    OOOO    OO          55
      44    OOOO    OO  OOOO    OO                      44    OOOO    OO          55
      44    OO      OO  OO      OO      ,,,,            44    OO      OO  55      55
      44    OO      OO  OO      OO      ,,,,            44    OO      OO  55      55
      44      OOOOOO      OOOOOO          ,,            44      OOOOOO      555555
      44      OOOOOO      OOOOOO          ,,            44      OOOOOO      555555
```

```
BBBB  U   U N   N DDDD  Y   Y        AAA
B   B U   U N   N D   D Y   Y       A   A
B   B U   U NN  N D   D  Y Y        A   A
BBBB  U   U N N N D   D   Y         A   A
B   B U   U N  NN D   D   Y         AAAAA
B   B U   U N   N D   D   Y         A   A
BBBB  UUUUU N   N DDDD    Y         A   A
```

```
LPTSPL Version 6(344)  Running on TTY70
 *START* User BUNDY A [400,405] Job  DOME Seq. 1505 Date 14-Sep-78 14:40:07 Monitor ERCC 0106
Request created: 14-Sep-78  14:41:10
File: DSKA0:BLOC.SOL[400,405] Created: 14-Sep-78 14:05:00 (155)  Printed: 14-Sep-78 14:42:26
QUEUE Switches:  /PRINT:ARROW /FILE:ASCI
I /COPIES:1 /SPACING:1 /LIMIT:29 /FORMS:NORMAL
File will be RENAMEd to (055) protection
```

```
! ?-bloc.
trying-to-solve-(sqrt(2*g*h1))0&real(sqrt(2*g*(h1-h2)))&sqrt(2*g*(h1-h2)))0&real(sqrt(2*g*(h1-
(sqrt(2*g*h1))0&real(sqrt(2*g*(h1-h2)))&sqrt(2*g*(h1-h2)))0&real(sqrt(2*g*(h1-h2-1*tan(t))))&
assuming-h1-positive
2*g*(h1+(-h2)))=0&2*g*(h1+(-h2)))0&2*g*(h1+(-h2)+-1*l*tan(t)))=0
these-conditions-simplify-to-(2*g*(h1+(-h2)))=0&2*g*(h1+(-h2)))0&2*g*(h1+(-h2)+-1*l*tan(t)))=(
isolating-h1-in-(2*g*(h1+(-h2)))=0)
isolating-h1-in-(-h2+h1)=0*(g*2):-1)
0*(g*2):-1+ -(-h2)-simplifies-to
h2
h2-simplifies-to
h2
isolating-h1-in-(2*g*(h1+(-h2)))0)
isolating-h1-in-(-h2+h1)0*(g*2):-1)
0*(g*2):-1+ -(-h2)-simplifies-to
h2
h2-simplifies-to
h2
isolating-h1-in-(2*g*(h1+(-h2)+-1*l*tan(t)))=0)
isolating-h1-in-(tan(t)*(l*-1)+(h1+(-h2)))=0*(g*2):-1)
isolating-h1-in-(h1+(-h2))=0*(g*2):-1+(--1*(l*tan(t))))
0*(g*2):-1+(--1*(l*tan(t)))+ -(-h2)-simplifies-to
tan(t)*l+h2
tan(t)*l+h2-simplifies-to
l*tan(t)+h2
isolating-h1-on-the-lhs-gives-(h1)=h2&h1)h2&h1)=l*tan(t)+h2)
trying-to-find-maximum-of-(h1)=h2&h1)h2&h1)=l*tan(t)+h2)
(h2)=h2)-simplifies-to
trying-to-find-maximum-of-(h1)h2&h1)=l*tan(t)+h2)
(l*tan(t)+h2)h2)-simplifies-to
trying-to-find-maximum-of-(h1)=l*tan(t)+h2)
(h1)=l*tan(t)+h2)-dominates-the-other-inequalities
ans-is-(h1)=l*tan(t)+h2)


wes
```

for condition 2



along(Path, Start, Acc, R) :-

do end(lineseg(path, Path, [Start | Rest])),

next to (Start1, Final, Rest)

do point(path, Path1, R),

path to (Path, Path without),

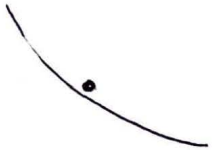
not member (Path1, Path without)

not found (Path1, Start1, Final1)

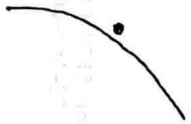
along (Path1, Start1, Acc, R)

* important

all motion left to right



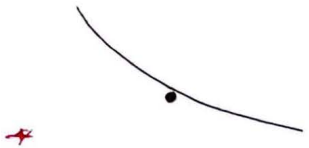
S_1 S_1' S_5'



dome.

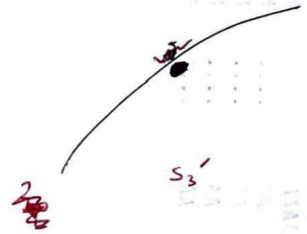
S_2 S_2'

S_1''



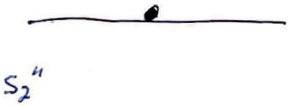
S_4'

* Overhang

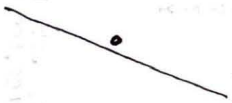


S_3'

Types of simple motion tackles so far



S_2''

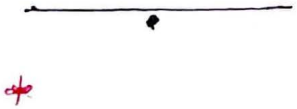


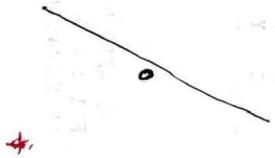
S_3.

S_x — block

S_x' — loop

S_x'' — ramp





```
setstarted(Part,Path,Start,Side,Besin) :- !,
  alons(Path,Start,Start,Dir),
  cc vel(Part,V,Dir,Besin),
  condition(positive(V)).
```

failure case

```
/*DOWNHILL RUN*/
nostoppins(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*/
nostoppins(Part,Path,Start,Side,Per) :- !,
 ncc farend(Path,Start,Finish),
 alons(Path,Start,Finish,Dir),
 cc finvel(Part,V,Dir,Per),
 condition(real(V)).
```

failure case

```
/*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,risht), !.

/*INSUFFICIENT VEL TO TAKE OFF*/
notakeoff(Part,Path,Start,Side,Per) :-
  dc concavity(Path,risht), !,
  cc typical_point(Path,TypPt),
  towards(Path,Part,TypPt,Per,Dir),
  cc reaction(Path,Part,N,Dir,Per),
  condition(non_neg(N)).
```

failure case

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

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

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

/*FREEFALL*/
nofalloff(Part,Path,Start,Side,Per) :-
  dc concavity(Path,Conc), diff(Conc,risht),
  dbentry(falls_off(Part,Path,Start,Side,Per)),
```

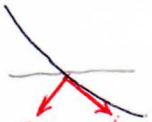
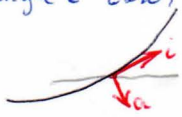
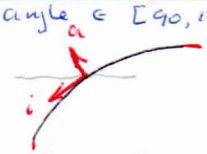
mecho note 48

Findins the Quadrant of Ansles

Alan Bundy 15.1.79

It is very useful in the Alsebra module to be able to decide whether a particular ansle is acute, reflex etc. With this information the equation solver can cut down the number of disjunctions in a solution, by eliminatins semantically unacceptable solutions.

If the ansle is the incline or ansle of a simple curve then which quadrant the ansle lies in can be calculated from the slope and concavity of the curve. We will specify the four quadrants to which an ansle misht belons with the ranses: [0,90]; [90,180]; [180,270] and [270,360]. For definitions of the conventions for definins incline and ansle see note 3. In particular note that the ansle is measured from the x-axis, anti-clockwise, to the normal and the incline is 90 bisser than the ansle. Usins these conventions and the conventions for concavity and slope (see note 4) we can classify the simple curves as follows, assisnins quadrants to ansles:

slope \ concavity	+ve left	-ve right	0 stline
-ve left	angle ∈ [180,270]  incline ∈ [270,360]	angle ∈ [0,90]  incline ∈ [90,180]	angle ∈ (180,270)  incline ∈ (270,360)
+ve right	angle ∈ [270,360)  incline ∈ [0,90]	angle ∈ [90,180]  incline ∈ [180,270]	angle ∈ (270,360)  incline ∈ (0,90)
0 hor	×	×	angle ~~∈ [270,360]~~  incline = 0 ~~∈ [0,90]~~

∞

Note that the desenerate cases of straisht lines, particularly horizontal straisht lines can be dealt with as curves provided we interpret the concavity

of straight lines as 'left' and the slope of horizontal lines as 'right'.

This table can be given to Mecho as a set of unit clauses, using the predicate 'quad'. 'quad' will take 4 arguments: the first will be 'angle' or 'incline' to specify which angle is being classified; the second will be the slope of the curve; the third will be the concavity of the curve and the fourth will be the quadrant the angle lies in. The clauses are:

```
quad(angle,left,right,[0,90])          quad(incline,left,right,[90,180])
quad(angle,right,right,[90,180])       quad(incline,right,right,[180,270])
quad(angle,left,left,[180,270])        quad(incline,left,left,[270,360])
quad(angle,right,left,[270,360])       quad(incline,right,left,[0,90])
```

Complex curves can be dealt with by: breaking the curve into simple curves; finding the range of the angle in each of the constituents and then finding the union of these ranges.

Now to test whether a particular inequality, say 120>=theta, Mecho can first find the range of theta, suppose this is [0,90], then it can test whether the upper bound of the range is less than 120, i.e. test 120>=90.

Roller Coaster (Quantative)

Bloc

Provisos

Vel1 > zero & ok (direction1)
Real (vel2)
Vel2 > zero & ok (direction2)
Real (vel3)

Equations

$\frac{1}{2}$ vel2^2 − $\frac{1}{2}$ vel1^2 = g . h2
$\frac{1}{2}$ vel3^2 − $\frac{1}{2}$ vel2^2 = g . −(l + tan(T))
$\frac{1}{2}$ vel3^2 − $\frac{1}{2}$ zero^2 = g (h1 + h2 − (l + tan(T)) + 0)

Loop

Provisos

Real (vel2) [vel2, vel4, vel5, angle1 ~~vel6~~, vel7.]
Vel3 > zero
Real (vel4)
Vel5^2 . sin(angle1) ≥ r * g
Vel4 > zero
Vel7 $\frac{3}{4}$ sin(angle1) >, r * g.

Equation

✓ $\frac{1}{2}$ vel1^2 − $\frac{1}{2}$ zero^2 = g . h
✓ $\frac{1}{2}$ vel2^2 − $\frac{1}{2}$ vel1^2 = gr(sin(270) − sin(0)) = −gr
✓ $\frac{1}{2}$ vel4^2 − $\frac{1}{2}$ vel2^2 = gr (sin(0) − sin(90)) = −gr
✓ $\frac{1}{2}$ vel5^2 − $\frac{1}{2}$ vel3^2 = gr (sin(0) − sin(angle1)) = −gr sin(angle1)
✓ $\frac{1}{2}$ vel7^2 − $\frac{1}{2}$ vel4^2 = gr (sin(90) − sin(angle1)) = gr(1 − sin(angle1))

Solving

vel2^2 = 2gh − 2gr
vel4^2 = 2gh − 4gr
vel5^2 = 2gh − 2gr (1 + sin(angle1))
vel7^2 = 2gh − 2gr (1 + sin(angle1))

Done

Provisions

reaction1 ≥ zero

Equations

mass1 * g + reaction1 * cos(270 - dir)
= mass1 * (-vel2²/r) cos(270 - dir) . (i)

* false → mass1 * g * cos(-270) + reaction1 * cos(0 - dir)
= mass1 * (-vel2²/r) cos(0 - dir) . (ii)

½ * vel2² - ½ zero² = g * r * (sin(90) - sin(dir)) . (iii)

(equations needed)

Reaction1 - mass1 . g sin(dir) = mass1 * -(vel2²/r) (iv)

plus (iii)

solving for Reaction1 = mass1 . (3g sin(-dir) - 2g)

∴ new provision is

mass1 * g * (3 * sin(dir) - 2) ≥ zero .

on conversion

m * g * (3 * sin(dir) - 2) ≥ 0.

goal is

?- Min(dir, -minval)

Bloc coté

Solving:

$$vel3^2 = 2g\,(h1 + h2 - l.\tan(\tau))$$
$$vel2^2 = 2g\,(h1 + h2)$$
$$vel1^2 = 2g\,h1$$

new problem

$$\begin{cases} \sqrt{2gh1} > 0 \\ \text{Real}\left(\sqrt{2g(h1+h2)}\right) \\ \sqrt{2g(h1+h2)} > 0 \\ \text{Real}\left(\sqrt{2g(h1+h2-l\tan(\tau))}\right) \end{cases}$$

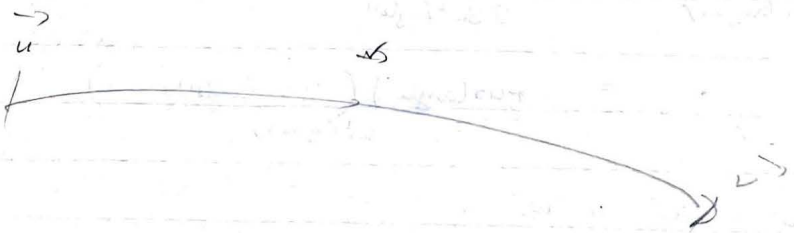
$$\begin{cases} 2gh1 > 0 \\ 2g(h1+h2) \geq 0 \\ 2g(h1+h2) > 0 \\ 2g(h1+h2-l\tan(\tau)) \geq 0 \end{cases}$$

delete duplicate

$$\begin{cases} h1+h2 \geq 0 \\ h1+h2-l\tan(\tau) \geq 0 \end{cases}$$

} subsumed

$$h1+h2-l\tan(\tau) \geq 0$$



Loop cont

New provisos

$$\text{Real}\left(\sqrt{2gh - 2gr}\right) \quad \text{(i)}$$

$$\sqrt{2gh - 2gr} > 0 \quad \text{(ii)}$$

$$\text{Real}\left(\sqrt{2gh - 4gr}\right) \quad \text{(iii)}$$

$$\{2gh - 2gr(1 + \sin(\text{angle1}))\}\sin(\text{angle1}) \geq r * g \quad \text{(iv)}$$

$$\sqrt{2gh - 4gr} > 0 \quad \text{(v)}$$

delete duplicate →

$$\{2gh - 2gr(1 + \sin(\text{angle1}))\}\sin(\text{angle1}) \geq r * g. \quad \text{(vi)}$$

Proof

$$2gh - 2gr \geq 0 \quad \& \quad 2gh - 2gr > 0$$

$$2gh - 4gr \geq 0 \quad \& \quad 2gh - 4gr > 0$$

$$\{2gh - 2gr(1 + \sin(\text{angle1}))\}\sin(\text{angle1}) \geq r * g.$$

- $h \geq 2gr/2g$. $(= r)$
  $h > r$
- $h \geq 4gr/2g$. $(= 2r)$
  $h > 2r$

$$h \geq \left(\frac{r * g}{\sin(\text{angle1})} + 2gr(1 + \sin(\text{angle1}))\right)/2g.$$

$h \geq F(\text{angle1})$

$$dF/d(\text{angle1}) = -\frac{r * \cos(\text{angle1})}{2\sin^2(\text{angle1})} + r\cos(\text{angle1})$$

$$0 = \frac{\frac{1}{2} r \cos(\text{angle1})(2\sin^2(\text{angle1}) - 1)}{\sin^2(\text{angle1})}$$

$$\text{angle1} = 45^\circ \text{ or } 90^\circ$$

$$\frac{d^2F}{d(\text{angle1})^2} =$$

New proviso $h \geq \frac{1}{2}\left(\frac{r}{\sin 90} + 2r(1 + \sin 90)\right) = \frac{r + 4r}{2} = \frac{5}{2}r$

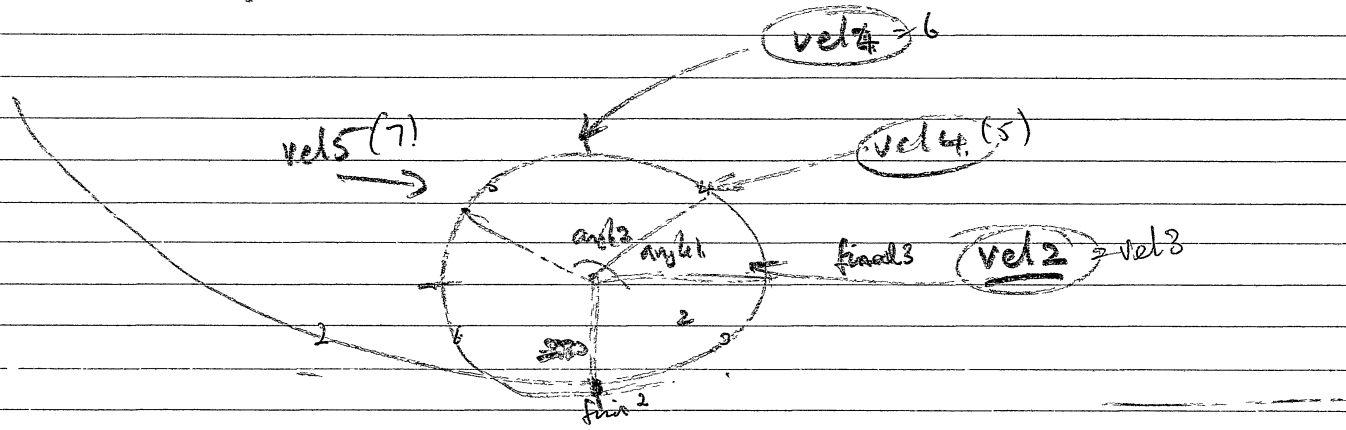
$$h \geq \max(5/2\, r,\ 2r,\ r) = 5/2\, r$$

$$\text{minval} = 5/2\, r.$$

Loop the loop

Real(vel2) & Real(vel3) . ⇒ $vel4 * \sin(ang4) \Rightarrow R.G$

$vel5^2 * \sin(ang2) \Rightarrow R.G$



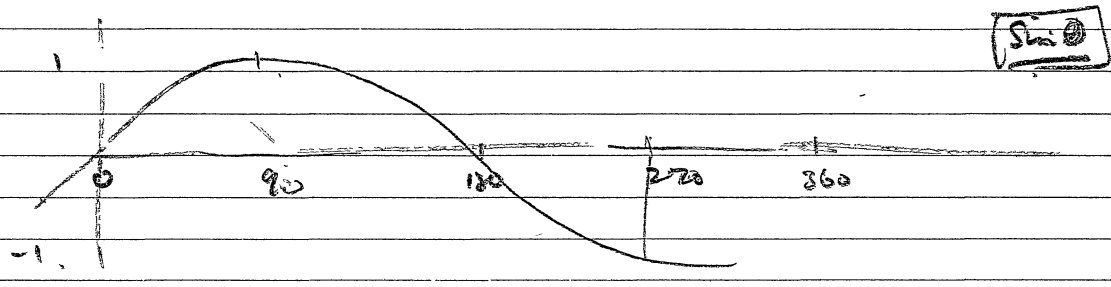
$\frac{1}{2}v_1^2 - \frac{1}{2}.gen^2 = g.h.$

$\frac{1}{2}v_2^2 - \frac{1}{2}v_1^2 = g.r(\sin(270) - \sin(\theta)) = -g.r$

$\frac{1}{2}v_3^2 - \frac{1}{2}v_2^2 = g.r(\sin\theta - \sin 90) = -g.r.$

$\frac{1}{2}v_4 - \frac{1}{2}v_2 = g.r(\sin\theta - \sin\theta) = -gr\sin\theta$:)

$\frac{1}{2}v_5 - \frac{1}{2}v_3 = g.r(\sin 90 - \sin\theta) = gr(1-\sin\theta)$



$v_2^2 = -2gr + 2g.h.$

$v_3^2 = -2gr + -2gr + 2gh = 2gh - 4gr.$

$v_4^2 = -2gr\sin\theta + -2gr + 2gh = 2gh - 2gr(\sin\theta + 1)$

$v_5^2 = 2gr(1-\sin\theta) + 2gh - 4gr = 2gh - 2gr(1+\sin\theta)$

$2gh - 4gr \geq rg$
when $\theta = 90$

new conditions are :

Real$(\sqrt{2gr+2gh})$ & Real$(\sqrt{2gh-4gr})$

& $(2gh + 2gr(\sin\theta+1))\sin\theta \geq rg$ & $(2gh - 2gr(\sin\theta+1))\sin\theta \geq rg$

eval. (identity drop 1.)

$\Leftrightarrow$

$r + gh \geq gr$ & $gh \geq 2gr$ ① follows from ②.

& $2gh\sin\theta - 2gr\sin^2\theta - 2gr\sin\theta \geq rg$?

$\Leftarrow$ $h \geq 2r$ & $2gh - 4gr\sin\theta \geq rg$

min when $\theta = 90°$??

$\Leftarrow$ $h \geq 2r$ & $h \geq \frac{5r}{2} = 2\frac{1}{2}r$

this is more

$h \geq \frac{r + 2r\sin\theta(1+\sin\theta)}{2\sin\theta} = F(\theta)$

$\frac{dF}{d\theta} = r \frac{d\left(\frac{1+2\sin\theta+2\sin^2\theta}{2\sin\theta}\right)}{d\theta}$

$\frac{d\,v/u}{d\theta} = \frac{1}{u}dv + v\left(\frac{-1}{u^2}\right)du = \frac{u\,dv - v\,du}{u^2}$

$= r.\frac{\sin\theta\, d(1+2\sin\theta+2\sin^2\theta) - (1+2\sin\theta+2\sin^2\theta)\cos\theta}{2\sin^2\theta}$

$= r\left(\frac{\sin\theta(2\cos\theta + 4\sin\theta\cos\theta) - (1+2\sin\theta+2\sin^2\theta)\cos\theta}{2\sin^2\theta}\right)$

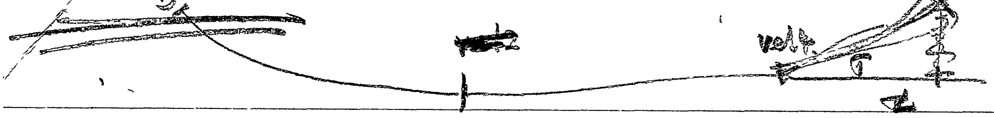
$= r\left(\frac{2sc + 4s^2c - c - 2sc - 2s^2c}{2s^2}\right)$

$0 = r\left(\frac{(2s^2-1)c}{2s^2}\right)$

$s^2 = \frac{1}{2}$ or $c = 0$ $\therefore$ $\theta = \sin^{-1}\frac{1}{\sqrt{2}}$ v 90

$s = \frac{1}{\sqrt{2}}$ $= 45°$ or $90°$ or

Sliding Bloc



$\tan T = \frac{x}{L}$

$$\frac{1}{2} V_2^2 - \frac{1}{2} 0^2 = g.(h_1 - h_2 - L\tan(T))$$

$$V_2 = 2\sqrt{g(h_1 - h_2 - L\tan(T))}$$

$$\Downarrow$$

Real (V_2) $\Leftrightarrow$ Real$(2 . \sqrt{\quad})$

$\Leftrightarrow$ Real $(\sqrt{\quad})$

$\Leftrightarrow$ $g(\quad) \geq 0$

$\Leftrightarrow$ $(\quad) \geq 0$

$\Leftrightarrow$ $h_1 \geq h_2 + L\tan(T)$

put in meta-language

Real (_x*_y) ← Real(_x), Real(_y).
Imag( ), Imag( )

Real(_x). ← Integer(_x).   via relations

Real(sqrt(_x)) ← _x ≥ 0

_x . _y ≥ 0 ← _x ≥ 0, _y ≥ 0.

— ≤ ≤

g ≥ 0.

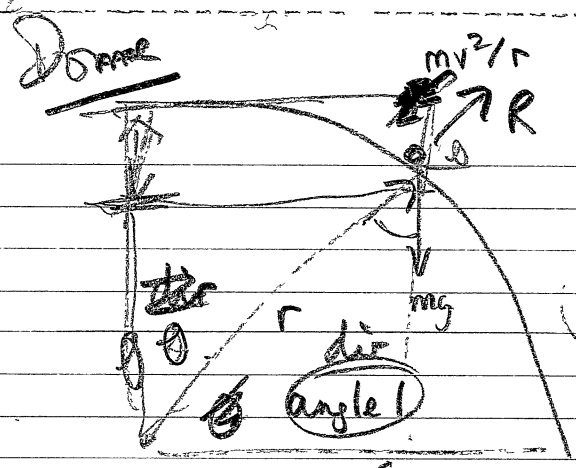
The questions

:- are (Timeseq (_P, Ma, _F)).

:- are (Motsgo (M, sϕ, Ca, left, period1)).

:- isegn (_E, conservenergy = (period1 _ M), Nil), Pr(_E)

:- drop(maSϕ, Ca, _h), Pr(_h).



$R - mg\cos\theta = -mv^2/r$

$R \geq 0$

$R\cos(270-\theta) + mg = -mv^2/r\cos(270-\theta)$ ↓ this dir

$R\cos(-\theta) + mg\cos 300 = -mv^2/r\cos(-\theta)$ → this dir

(& accels moving)

$-mv^2/r + mg\cos\theta \geq 0$

$-v^2 + gr\cos\theta \geq 0$

$\cos\theta \geq v^2/rg.$

$\frac{1}{2}v^2 - \frac{1}{2}u^2 = g(r - r\cos\theta)$

$v^2 = 2gr(1-\cos\theta)$

$\cos\theta \geq 2$

$rg(\cos\theta - 2 + 2\cos\theta) \geq 0$

$3\cos\theta \geq 2$

$\theta = \cos^{-1} 2/3.$ ✓

Block

$h_1 > 0$

$h_1 \geq -h_2$

$h_1 > -h_2$

$h_1 \geq l\tan(T) - h_2$

$0 \geq -h_2$? no

$0 \geq -h_2$? no.

$0 \geq l\tan(T) - h_2$? no

since h_2, l, $\tan T$ nonneg.

$-h_2 > -h_2$? no — simplify

$-h_2 \geq l\tan(T) - h_2$ no ← $l \tan(T)$ positive!

$-h_2 \geq l\tan(T) - h_2$ no

∴ positive

$-h_2, l, \tan T, r$ ✓ ✓ ✓ ✓

s	a
t	c

$0 < T \leq 90$

loop

$h \geq r$

$h > r$

$h \geq 2r$

$h > 2r$

$h \geq \frac{5}{2} r$

Ex. $r > r$ no

$r \geq 2r$ no

$r > 2r$ no

$r \geq \frac{5}{2} r$ no

} simplify — $r \neq 0$ — $r \neq 0$

$r \geq 2r$

$$\frac{-r*g*\cos(\text{angle})}{2g\sin^2(\text{angle})} + \frac{2*r*g\cos(\text{angle})}{2*g} = 0.$$

$$r.\cos(\text{angle}).\left(1 - \frac{1}{2\sin^2\text{angle}}\right) = 0$$

(collection)

$\cos(\text{angle}) = 0$ or $1 = \frac{1}{2\sin^2\text{angle}}$

~~factorize~~

angle $= 90$ or $\sin^2(\text{angle}) = \frac{1}{2}$

isolate

angle $= \arcsin\frac{1}{\sqrt{2}}$

$= 45°$.

Will it Reach the Top? Prediction in the Mechanics World *

Alan Bundy

University of Edinburgh, Edinburgh EH8 9NW, Scotland

Recommended by R. Boyer

ABSTRACT

We describe an extension of a mechanics problem solving program to the set of "roller coaster" problems, i.e. problems about the motion of a particle on a complex path. The reasoning strategy adopted by the program is described and compared to earlier work in this domain. Conclusions are drawn about the representation of motion and prediction. Questions are raised about Frames and Multiple Representations.

1. Introduction

In this paper we describe how the MECHO mechanics problem solver [2] was adapted to deal with the "roller coaster" problems described by de Kleer [6]. Why did we decide to tackle the roller coaster problems? The reasons are listed below.

(i) Our MECHO program is intended as a general problem solver for the mechanics world. de Kleer's NEWTON, on the other hand, was specifically designed for the roller coaster problems and was based on the ideas of "Frames" [11]. We wanted to see whether the study of the roller coaster problems would show up important limitations of MECHO.

Thus our main motivation was to test the existing MECHO mechanism and to see in what way it needed to be extended. We consider such a research methodology at least as valid as the more common one of using a problem domain to explore some new mechanism (e.g. Frames).

(ii) The roller coaster domain is important as it isolates in pure form the problems of motion, which are involved in all dynamics problems.

(iii) These problems are also involved in the prediction of subsequent events given some initial configuration. An issue we had not dealt with before.

* This research was supported by S.R.C. grant BRG 94493.

Artificial Intelligence **10** (1978 129–146),

(iv) We were dissatisfied with the reasoning processes which NEWTON went through to solve the problem. Some steps seemed irrelevant. We resolved to do better. This is not to underestimate the pioneering work of de Kleer, without which the present work would not have been possible.

2. The Problems

Below we describe the three roller coaster problems tackled by de Kleer's NEWTON.

2.1. The sliding block

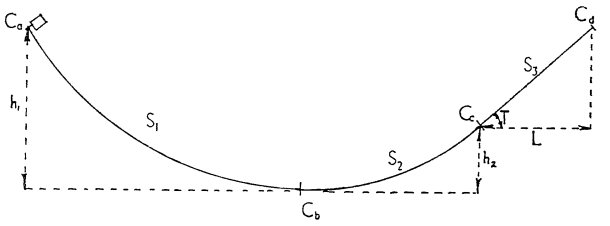


FIG. 2.1.

At time M_a the block starts from rest at point C_a. Will it reach point C_d?

2.2. The loop the loop

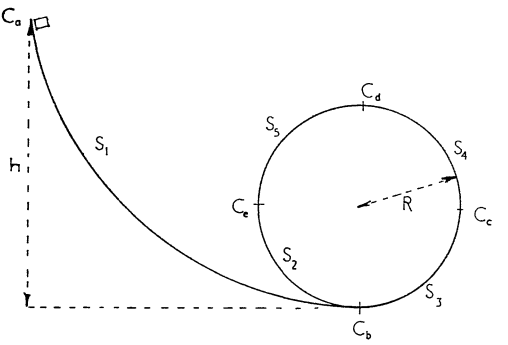


FIG. 2.2.

At time M_a the block starts from rest at point C_a. How small can h be made so that the block still successfully executes the loop the loop?

2.3. The great dome

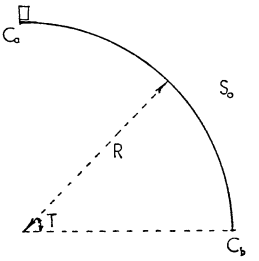


FIG. 2.3.

The block is given a nudge from rest. At what point does it fly off the dome?

All the paths are assumed to be smooth (i.e. friction is ignored).

2.4. Describing the problems

The above problems do not lend themselves to prose description without the aid of a diagram. Since we have excluded visual processing from the current project and they cannot be handled by the natural language processing alone, we decided to bypass this stage and describe the problems to MECHO in the form of symbolic descriptions. Fortunately de Kleer had already developed a symbolic notation for describing these problems and we adopted this in content, though not in form. The essential idea of de Kleer's notation is a description of the paths in terms of their first and second derivatives, i.e. their slope and concavity. These descriptions play a vital part in the prediction process.

The main task which would be required of a visual processing component of MECHO, if we were to build one, would be the partitioning of complex paths into subpaths with invariant slopes and concavities, e.g. given the slope

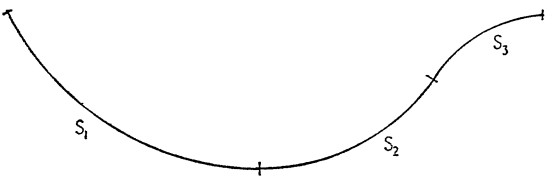


FIG. 2.4.

we would require the "vision" component to partition it into the subpaths: S_1; S_2; S_3, where

Path	*Slope*	*Concavity*
S_1	down	concave
S_2	up	concave
S_3	up	convex

We would be interested in the comments of those working in visual perception as to the feasibility of such processing.

3. Hypothesize and Test

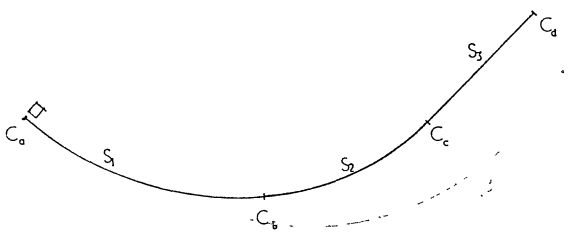
The reasoning strategy we developed for the three problems above is one of hypothesize and test. That is, we answer the question set by the problem in two stages. First the question is subjected to a qualitative test, i.e. is motion feasible given the rough shape of the path? etc. If this proves positive the answer "yes" becomes a hypothesis which is then subjected to a quantitative test, i.e. a detailed analysis involving extracting and solving equations and inequalities. This process can be carried out recursively, if necessary, by breaking the path into subpaths and testing each of them in turn.

It is not necessary, however, to carry out the quantitative testing locally. It can all be saved up to the end and done together.

This division into qualitative and quantitative knowledge was strongly suggested by de Kleer and we have adopted it in our program. The distinction is discussed more fully in Sections 4, 5 and 9. A small modification to the program would stop it from making the distinction, so that qualitative and quantitative testing would be done together. However, the two stage processing seemed to produce a more natural protocol and we have maintained de Kleer's division.

4. A Worked Example

A better idea of the reasoning process can be gained from considering an example. So in this section we consider how our strategy of hypothesize and test deals with the sliding block example.



path S_0

FIG. 4.1.

In our symbolic notation the question to be answered is expressed as

At (Block, C_d, _mom)?

which means "is the block at point C_d at some moment _mom", where _mom is a variable to be bound to a particular moment during the course of the problem

solving. On the grounds that "you can get to a place by travelling there" this question generates the sub-question

Motion (Block, S_0, C_a, left, P_0)?

which means "does the block travel on (above the) path S_0, starting at point C_a, during period P_0" (the fourth argument, left, will be explained later).

This question will be answered affirmatively provided a number of tests prove positive. These tests cannot be answered immediately because not enough is known about the shape of S_0: In particular its slope and concavity are not invariant. The question is, therefore, broken into three subproblems corresponding to the three subpaths, i.e.

Motion (Block, S_1, C_a, left, P_1)?

Motion (Block, S_2, C_b, left, P_2)?

Motion (Block, S_3, C_c, left, P_3)?

where $[P_1, P_2, P_3]$ is a partition of the time period P_0.

These three all pass the qualitative tests, storing up quantitative tests for later. The qualitative tests are expressed as four questions:

(a) Will the block get started?

(b) Will it run out of steam and stop?

(c) Will it go too fast and fly off?

(d) Will it fall off?

Some of these tests are passed without difficulty. For instance, the block will not fall off since it travels above all three paths. Nor will it take off, since none of the paths is convex. There is a possibility of its running out of steam on the second and third paths and this possibility causes a proviso to be stored against the velocities on these two paths. This proviso will be picked up and checked by the quantitative tester in the second stage. The proviso is that the velocities remain real valued. If a block does not reach the upper position of a path its velocity on that section will be imaginary.

When a motion description (e.g. Motion (Block, S_1, C_a, left, P_1)) has passed the qualitative tests and the quantitative provisos have been stored for future testing, it is asserted into the database as a hypothesis. It is necessary to do this because it is needed for the next round of qualitative tests. MECHO cannot decide that the block can "get started" on the next path unless it knows that the block is at the starting point at the right moment. This can be deduced from the fact that the block is at the finishing point of the previous path at the same moment and the two paths are consecutive.

When all the qualitative testing has been done, the provisos are gathered together for quantitative testing. Typically these provisos will involve unknown quantities, which must first be expressed in terms of quantities given in the statement of the problem. Therefore: a list is made of all unknowns in the provisos; equations are

extracted which express these unknowns in terms of the givens; the equations are solved; the solutions are substituted for the unknowns and the provisos are evaluated. By giving different numerical values to the given quantities the evaluation of the provisos can be made to return a positive or negative outcome. Alternatively, if the givens have symbolic values the evaluation process can return an expression as answer.

The algorithm for deciding what equations to extract was based on Marples 1974 study of Cambridge Engineering students. It works by carefully examining the unknown quantity whose value is sought to see what kind of quantity it is (e.g. acceleration, mass etc.) and what situation it occurs in (e.g. what objects, time and direction it is associated with). This information is then used to draw up an ordered short list of equations to be tried. If possible an equation is found which introduces no new intermediate unknowns, but if this is impossible intermediate unknowns are created and these are added to the list of unknowns whose value is sought. For further details see [2]. Notice that no information about the problem type is used, so that the algorithm is general purpose.

5. Finding Minima

The reasoning process for the other two examples is very similar except for the final stage. Instead of the proviso being evaluated to return a simple yes/no answer they are investigated to find minimum values under which they return true.

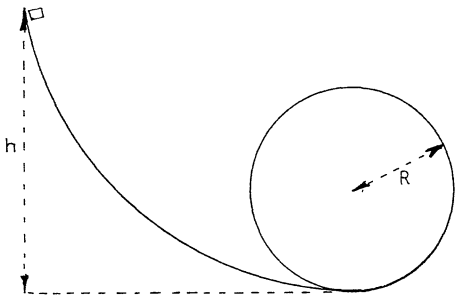


FIG. 5.1.

For instance, in the loop the loop example the original provisos concern conditions on the velocity of the block under which the block will not fall off or stop. After equation extraction, solution and substituting these become inequalities between h, the initial height of the block, and R, the radius of the circle. MECHO then has to find the minimum value h, under which the inequality is still true. This is done by "isolating" h to get an inequality of the form:

$$h \leqslant \tfrac{5}{2} R.$$

That is, the inequalities are manipulated until h is exposed, by itself on the left hand side and the right hand side contains no occurrence of h. One of the main methods of doing this is by repeatedly replacing the outermost left hand side function symbol by its inverse on the right hand side, e.g.

$$x \cdot y \leqslant z \;\&\; x > 0 \rightarrow y \leqslant z/x$$

(see [1] for a description of isolation in the case of equation solving). The problem of finding a minimum value for h to satisfy an inequality, can then be exchanged for the problem of finding the value for h which satisfies an equality, namely:

$$h = \tfrac{5}{2} R$$

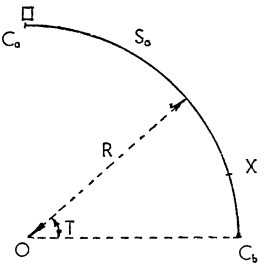


FIG. 5.2.

The great dome problem is handled similarly to the loop the loop problem. That is MECHO tries to prove that motion takes place from C_a to C_b, collects together the provisos, then finds the least T which makes them true. One interesting feature of this process is that we make use of a hypothesis which is actually false, since the block can never reach C_b, without taking off first. For the squeamish an alternative reasoning process is available where we consider motion from C_a to the point of take off (say X above). Similar conditions would be generated on T and the minimum value which makes these conditions true would also be the angle $\widehat{XOC_b}$.

We can easily visualize other processes which could be applied to the provisos, e.g. finding maxima, comparing with an answer expression given in the problem etc. One of the reasons for leaving quantitative testing to the end is that it makes such proviso processing easier to execute. In MECHO the proviso processing procedure is passed down as an argument to QA the main question answering procedure. The other argument is the top level question to be answered. Thus the top level goals of each of the problems is:

sliding block: QA(At (Block, C_d, _mom), eval)
loop the loop: QA(Motion (Block, S_0, C_a, left, _per), Min(h, _ans))
great dome: QA(Motion (Block, S_0, C_a, left, _per), Min(T, _ans))

6. Brief Description of de Kleer's NEWTON

As mentioned above NEWTON also works in two stages: qualitative followed by quantitative reasoning. In our opinion de Kleer's main contribution was in the design of the qualitative reasoner, called the *envisioner*.

The envisioners job was to work out all possible scenarios for the Block, given its initial position and the shape of the path. Thus in the sliding block example the Block might reach the top (point C_a) or it might stop during the course of traversing S_2 or S_3, slide back and oscillate about the lowest point (C_b). The scenarios are built up in the form of a tree, e.g.

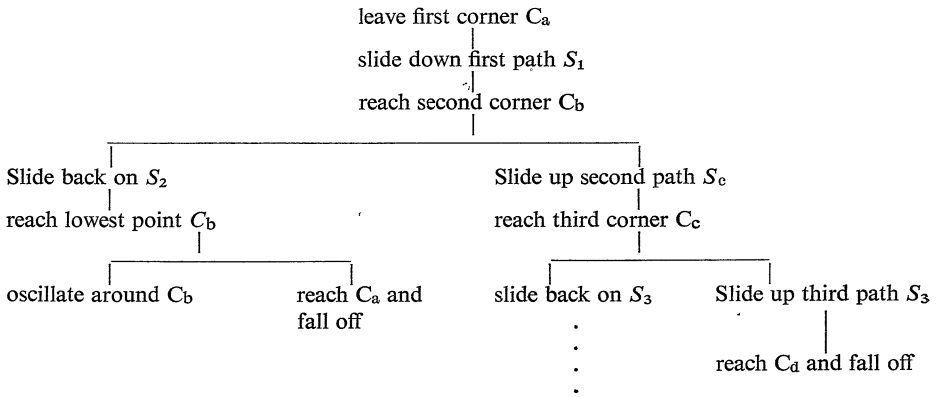


FIG. 6.1. Envisionment tree.

Envisionment is done by a set of 11 production rules. These look at the current state of the Block and local features of the path it is currently on and predict what might happen next. A typical rule is:

$$\text{velocity–u} \;\&\; \text{incline} \;\&\; \text{above} \;\&\; \text{on} \rightarrow \text{slide–u}$$

This means: if the Block is travelling up and is on and above an inclined path then it may continue to slide up.

The envisionment tree is later examined and quantitative tests applied to see which branch of the tree will actually be taken. For instance, the vertical height of the various paths will determine whether the Block reaches the top or oscillates about the lowest point. This is done by associating a different quantitative test with each qualitative ambiguity, e.g. a quantitative test of the velocity is associated with the ambiguity

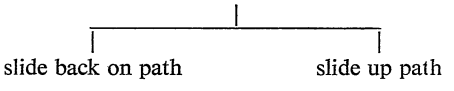


FIG. 6.2. Qualitative ambiguity.

7. A Comparison of NEWTON and MECHO

In applying MECHO to the roller coaster problems we naturally tried for an improved performance, where possible. The main improvement was that MECHO did not generate all the envisionment tree, but only those branches required to answer the question asked. Thus in the sliding block example only the right-most branch of the envisionment tree in Fig. 6.1 was considered. This was done, as described in Section 4, by finding a path between the initial and desired point, and asking only about motion between these points. This more goal-directed envisionment was always an improvement except when the question was a general one like "what happens next" which implies doing a forwards analysis. We did not come across, nor allow for, such questions.

This goal-directed reasoning entailed doing the envisionment process by applying something like de Kleer's production rule system, but in reverse, e.g. trying to prove that the block could slide up the path, by showing that the conditions were right. We first tried simply listing all situations under which sliding was possible, but this proved too cumbersome. Instead we discovered a way of simplifying the problem. Each question about whether motion could take place in a given situation was divided into four sub-questions, namely:

(a) Will the block get started?
(b) Will it run out of steam and stop?
(c) Will it go too fast and fly off?
(d) Will it fall off?

According to the situation these sub-questions may be answered simply on the basis of qualitative information. Alternatively they may require quantitative testing. This is delayed until later by storing the quantitative test (some algebraic expression) as a proviso. These provisos are added up, so that a motion question may generate up to four provisos. As an illustration the procedure for answering sub-question (d) above is explored in detail in Section 10.

This way of organising the qualitative tests seems to be more primitive than de Kleer's. For instance, he has a single quantitative test for the qualitative ambiguity fly-slide-slide illustrated by the following tree and diagram

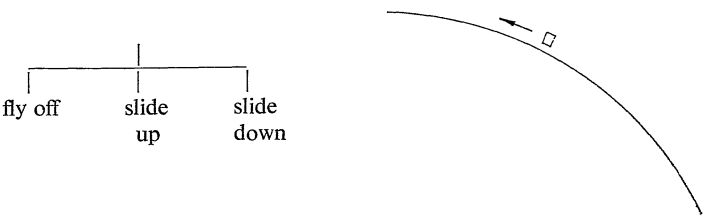


FIG. 7.1.

In MECHO the test for this situation is provided in two parts, by provisos from sub-questions (b) and (c). MECHO's four sub-questions provides a more intimate relationship between the quantitative and qualitative knowledge. A trivial amendment to the program would cause the quantitative testing to be done at the same time as the qualitative testing instead of being delayed to the end.

It is also easy to see how MECHO could be extended to deal with extensions to the roller-coaster domain, e.g. for rings threaded on wires or rough paths. Such extensions are discussed in Section 10. We attribute this flexibility to MECHO's primitive representation, where quantitative tests are automatically built as the situation demands, rather than pre-stored. To update NEWTON to deal with, say, threaded rings, it would be necessary for the programmer to do himself the analysis that MECHO does automatically. Thus the relationship between the two representations is analogous to that between the Huffman/Clowes line labelling [5] and Mackworth's analysis of it [9].

NEWTON's quantitative knowledge was stored as a collection of Frames [11] each of which contained related physical formulae. These Frames were intended to guide the equation extract on process in a sensible way. MECHO on the other hand used the general purpose Marples algorithm for equation extraction which works by backwards reasoning from the unknowns whose values are sought. Despite these apparently different organisations, we have been unable to detect any difference in the manner or efficiency with which equations are actually extracted by the two programs (except that NEWTON seems to have a difficulty with simultaneous equations which was not experienced by MECHO).

8. Representing Paths and Motion

In this and the next two sections we describe in more detail the descriptive terms (i.e. notation) and the procedures used by MECHO to describe and reason about the roller coaster world. We do this to try to make clear exactly how the program works—even down to the actual code in a few carefully chosen cases. To describe the program solely at the less detailed level we have used so far lays us open to the danger of ascribing more agency to the program than is, in fact, justified.

We also hope to promote more discussion of the description terms used in reasoning programs. In recent years the emphasis has been on control structures rather than descriptive power. The choice of terms, however, can make just as big a contribution to successful reasoning. The tradeoffs between different kinds of representation need discussion.

We start with the description of the motion of a particle on a path. This involves making four binary or tenary choices, i.e.

(i) Does the path slope downhill, uphill or is it horizontal?
(ii) Is the path convex, concave or straight?
(iii) Does the particle travel from left to right or right to left?
(iv) Is the particle above or below the path?

In fact, these choices are essentially binary with some degenerate cases. To simplify programming we have chosen the same parity pair left/right for indicating all these choices. Four predicates have argument places for containing either left, right or one of the degenerate cases, they are: End, Slope, Concavity and Motion. For instance, the following situation

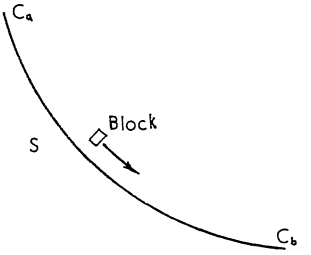


FIG. 8.1.

is described by

End (S, C_a, left) meaning C_a is the left end of S.

Slope (S, left) meaning the left end of S is highest.

Concavity (S, left) meaning, looking from the left end, the left side of S is concave.

Motion (Block, S, C_a, left, P) meaning the Block starts from C_a and travels on the left side of S, looking from C_a.

All these parities can be varied independently, e.g.

End (S, C_a, left)

Slope (S, left)

Concavity (S, right)

Motion (Block, S, C_a, right, P)

would describe the situation

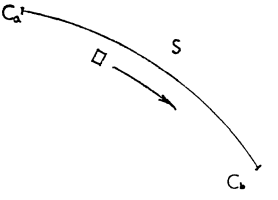


Fig. 8.2.

The choice of predicates may seem a bit arbitrary, but in fact it has been carefully developed to simplify the task of transforming motion descriptions between consecutive paths. Consider the following two situations

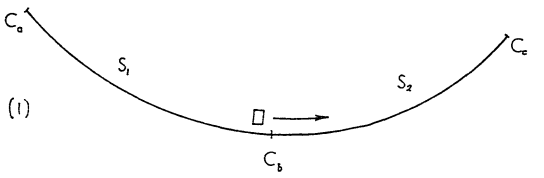


FIG. 8.3.

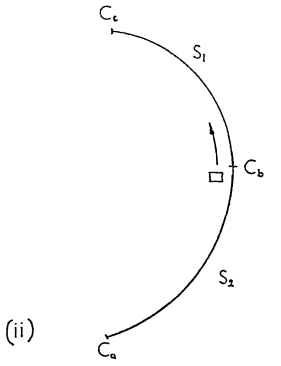


FIG. 8.4.

Using our notation

Motion (Block, S_1, C_a, left, P_1)

transfers to

Motion (Block, S_2, C_b, left, P_2)

in both cases. The fact that the block is now travelling from:

(a) left to right above the path in case (i)

and from

(b) right to left below the path in case (ii)

is handled by the different path descriptions. The task of transferring motion descriptions would have been much more tricky to handle neatly if we had tried to use descriptions (a) and (b) explicitly, to describe the motion.

9. Representing Time

In this section we describe the representation of time in MECHO. All descriptive terms dependent on time, have a time argument place (usually the last place), e.g.

At (Block, C_a, M_a)

or Motion (Block, S_1, C_a, left, P_1).

These time arguments can either be moments or periods. That is a description can be asserted to happen for an instant only, or to hold for some interval of time. Two special moments are associated with each period, namely its initial and final moments. The duration of a period is distinct from the period itself. Thus two different periods can have the same duration. A duration is a quantity, which can be measured in seconds, minutes, hours etc. The notation so far is:

Isa (M_a, Moment).

Isa (P_1, Period).

Initial (P_1, M_a).

Final (P_1, M_b).

Duration (P_1, T_1).

Breakdown (T_1, 2, secs).

To describe a situation in which a period can be divided up into several non-overlapping subperiods, the Partition predicate is used. This arises in the sliding block example where P_0 the period in which the block slides over the whole path is divided into P_1, P_2 and P_3, the periods in which the block slides over the three subpaths. This is represented as

Partition (P_0, [P_1, P_2, P_3]).

The second argument is always a list of sub-periods arranged in time order.

The same predicate, Partition, is used to describe the division of the whole path into its subpaths

Partition (S_0, [S_1, S_2, S_3]).

Here the subpaths are arranged in left/right order. That is the left end of S_0 is one of the ends of S_1.

In the description of the three roller coaster problems (see Section 2), the structure of the paths is given, but the structure of time is not. Only the initial moment is given. The rest of the time structure is erected during the course of solving the problem. For instance, in the sliding block problem the original question is

At (Block, C_d, _mom)?

The inference rule which translates this into a question about motion terminating at C_d (see Section 4), also invents a suitable period, P_0, during which the motion takes place. Since the Block is known to be at C_a at moment M_a, the initial moment of P_0 is asserted to be M_a. A suitable final moment M_d is also invented and the path to be traversed is S_0 since this connects C_a and C_d. (We have chosen the names M_d and P_0 for expositional purposes, the program "gensyms" up some fresh symbols different from this. Our gensym is a bit more readable than most, since you can specify a suitable prefix for the number, e.g. Period 1, vel 1, vel 2, etc.).

Not enough is known about S_0 for the qualitative tests to succeed, so MECHO

considers motion on the three subpaths S_1, S_2, S_3. For this it needs three more periods, so suitable ones are invented, say P_1, P_2 and P_3 and the relationship

Partition (P_0, [P_1, P_2, P_3]) is asserted.

MECHO does all this by mimicking the structure of the paths.

10. The Qualitative Tests

We are now in a position to describe the qualitative motion tests in more detail. These are expressed as a series of PROLOG *clauses* (see [12]). Each clause has the form

$A \leftarrow B, C, D.$

where A, B, C, and D are *literals*. A literal is a symbolic description like

Motion (Block, S_1, C_a, left, P_1) or Slope (S, right)

which may or may not contain variables (variables are indicated by the prefix _), e.g.

Motion (_part, _path, _start, left, P_1) or Slope (_path, right).

The meaning of

$A \leftarrow B, C, D.$

is that to prove A it is sufficient to prove B, C and D in that order.

The literals B, C, D are sometimes proved by a direct database look up, but usually involve further clauses of form $B \leftarrow E, F$, say, in a depth first search. In fact we have considerably modified the default PROLOG depth first search in MECHO: inserting traps to detect hopeless goals and reject them; and tests to decide that the search should be shortened in some cases and lengthened in others. For details of these modifications and the reasoning behind them see [3] or [4].

The main motion clause is

Motion (_part, _path, _start, _side, _per) ←
Getstarted (_part, _path, _start, _side, _per),
Nostopping (_part, _path, _start, _side, _per),
Notakeoff (_part, _path, _start, _side, _per),
Nofalloff (_part, _path, _start, _side, _per),
Record (Motion (_part, _path, _start, _side, _per)).

The first four literals after the backwards arrow are responsible for making the four qualitative tests. The last literal asserts the new hypothesis into the database, i.e. an attempt to "prove" Record (some-relation) has the effect of putting some-relation in the database.

The four qualitative tests are all handled in a similar way so we will describe only the last of them, Nofalloff. This test is divided into four cases: two cases which can be settled positively on qualitative information alone; one case which can be settled negatively on qualitative information alone and one case which requires a quantitative analysis. Each case is handled by a separate clause.

The first and simplest case is when the particle is above the path, then the particle cannot fall off the path

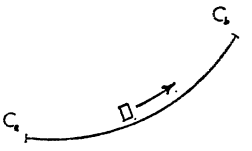


FIG. 10.1.

The clause for this case is:

Nofalloff (_part, _path, _start, _side, _per) ←
Above (_path, _start, _side), ! .

The ! is a control literal which prevents the backtracking mechanism recalculating this positive answer to Nofalloff in the event that later processing fails, i.e. an attempt to "prove" !, causes the search tree to be pruned of branches representing alternative ways of proving Nofalloff. Above (_path, _start, _side) tests that the particle is above the path using End and the value of _side, i.e.

Above (_path, _start, _side) ←
End (_path, _start, _side).

The second case concerns vertical slopes. If a particle is at the top of a vertical path then it is considered not to fall off, but to fall down to the end.

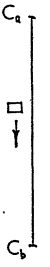


FIG. 10.2.

The representation of vertical paths is a bit messy in our notation. The top of the path is arbitrarily assigned as the left end. The verticality is noticed because the concavity is the degenerate "straight" case and the inclination from the top is 270°. The clause for this case is thus:

Nofalloff (_part, _path, _start, _side, _per) ←
Slope (_path, _start),
Concavity (_path, _),
Incline (_path, _start, 270), !.

The third case concerns convex slopes, where the velocity is sufficient for the particle to stick to the underside by centrifugal force.

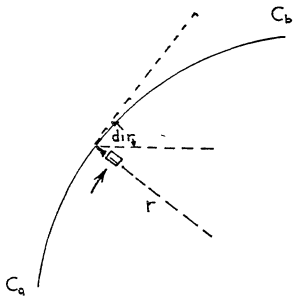


FIG. 10.3.

This case involves quantitative reasoning and so the calculation is delayed until the second stage by asserting a proviso. To calculate this it is necessary to know the velocity of the particle and the radius of curvature of the path. The clause for this case is:

Nofalloff (_part, _path, _start, _side, _per) ←
  Below (_path, _start, _side),
  Concavity (_path, right),
  CC (Vel (_part, _v, _dir, _per)),
  CC (Radius of curvature (_path, _r)),
  Postulate (Proviso ((_v ↑ 2) * Sin (_dir) ⩾ _r * g)), !

The first two literals make sure we are in the correct case. The second two literals recover the numeric quantities needed by the last literal which asserts the proviso into the database. The CC predicate ensures that the request for a numeric quantity succeeds even if this means creating some intermediate unknowns. This predicate is explained fully in [4]. The Postulate predicate is the same as Record except that the assertions are removed on backup.

The last case deals with the situation where the particle is below a non-convex path. In this case it is bound to fall off so the test fails.

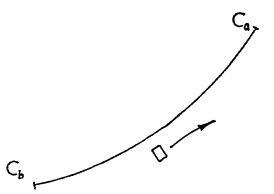


FIG. 10.4.

The clause for this case is

Nofalloff (_part, _path, _start, _side, _per) ←
  Below (_path, _start, _side)
  Concavity (_path, _conc), Diff (_conc, right),
  Record (Fallsoff (_part, _path, _start, _side, _per)),
  !, Fail

The literal Fail always fails thus ensuring that the call of Nofalloff will fail also. The ! guarantees that this failure cannot be reversed by the backtracking mechanism. The first two literals make sure we are in the correct case. The third literal "Record (. . ." puts a record in the database that the particle falls off the path during this period. This information is not currently used, since a suitable opportunity has not yet arisen (see Section 6).

We could easily add another case to Nofalloff designed to deal with rings on wires. Suppose the literal

Threaded (ring, wire, period) means ring is threaded on wire during period

then the extra clause would be

Nofalloff (_part, _path, _start, _side, _per) ←
  Threaded (_part, _path, _per), !.

This should be added first, so that it is tested before the other four cases.

11. Conclusion

In this paper we have described an extension of the mechanics problem solver MECHO to a new range of problems. We have compared the reasoning strategy adopted by MECHO with that used by de Kleer's NEWTON program on the same problems.

MECHO proved capable of solving these new problems provided it was extended to perform the "envisionment" process described by de Kleer. This extension was made and in the process the envisionment process improved by making it more goal-directed. The question of whether motion could take place was answered by breaking it into four sub-questions. This representation seemed more primitive than de Kleer's and thus more easily extended to new problem domains. The machinery developed here to handle the motion of a particle is now being used for a variety of problems in other areas of dynamics.

As de Kleer has said [7] it is important for a problem solver not only to solve hard problems, but to give simple solutions to easy problems. We disagree with de Kleer that this effect can only be achieved with a multiple representation. MECHO will apply the same technique to any problem. In the case of an easy problem this technique will rapidly degenerate and yield an easy solution. For instance, the question may be answered on the basis of qualitative information alone or after a simple calculation. Hard problems will need more work—for instance if friction were involved the algebra required to check the provisos would require detailed consideration of the shape of the curve etc. However, the qualitative knowledge

would still be used and the same basic proof plan would still be followed. Multiple representations are not desirable for their own sake, but only if forced by the nature of the problem, which in this case they are not.

To deal with the equation extraction process it was not necessary to use de Kleer's "Frame like" chunking of the physical formulae, our existing general purpose algorithm proved perfectly adequate. Indeed it was not obvious that the two mechanisms differed substantially in performance. This brings into question the nature of Frame type mechanisms in AI programs. Is there such a thing? Are there different types of Frames mechanisms? Do some of them correspond closely to previously known non-Frame mechanisms? Our suspicion here is that some uses of Frames and the old STRIPS plan formation techniques are much closer than is generally acknowledged (also see [8] and [13].

ACKNOWLEDGEMENTS

My thanks to my colleagues: George Luger, Martha Stone and Bob Welham, for many discussions about the MECHO project, Dave Warren for help with PROLOG and Peggy for her tireless typing.

REFERENCES

1. Bundy, A., Analysing mathematical proofs (or reading between the lines), in: Winston, P., Ed., Proceedings of IJCAI 4, Georgia, U.S.S.R. or DAI Research Report No. 2, Edinburgh (1975) 22–28.
2. Bundy, A., Luger, G., Stone, M. and Welham, R., MECHO: Year one, in: Brady, M., Ed., Proceedings of the 2nd AISB Conference, Edinburgh, or DAI Research Report No. 22, Edinburgh (1976) 94–103.
3. Bundy, A., Can domain specific knowledge be generalized? in: Reddy, R., Ed., Proceedings IJCAI-77, Cambridge, MA (1977) 496.
4. Bundy, A., Exploiting the properties of functions to control search. DAI Research Report No. 45, Edinburgh (1977).
5. Clowes, M. B., On the interpretation of line drawings as simple three dimensional scenes. Internal Memo, Sussex (1970).
6. de Kleer, J., Qualitative and quantitative knowledge in classical mechanics, AI Lab. Technical Report No. AI-TR-352, MIT, Cambridge (1975).
7. de Kleer, J., Multiple representations of knowledge in a mechanics problem solver, in: Reddy, R., Ed., Proceedings of IJCAI-77, Cambridge, MA (1977) 299–304.
8. Fikes, R. E., Automatic planning from a frames point of view, in: Schank, R. and Nash-Webber, B., Eds., TINLAP, Cambridge, MA (1975) 104–107.
9. Mackworth, A. K., Interpreting pictures of polyhedral scenes, Internal Memo, Sussex (1972).
10. Marples, D. L., Argument and technique in the solution of problems in mechanics and electricity, CUED/C–Educ/TRI, Department of Engineering Memo, University of Cambridge, England (1974).
11. Minsky, M., Frame–Systems: A framework for representation of knowledge, AI Lab. Memo No. AIM-306, MIT, Cambridge (1973).
12. Roussel, P., PROLOG: Manuel de reference et d'utilisation, Groupe dIA, Marseille-Luminy (1975).
13. Shank, R. C. and Abelson, R. P., Goals, plans, scripts and understanding: An inquiry into human knowledge structures, Erlbaum Press, N.J. (1977).

Received September 1977

S ⇒ done
M ⇒ P.

```
:- 'NOLC'.

/*PROBLEM 5, STAGE 4*/
/*DE KLEERS GREAT DOME PROBLEM*/
/*ALAN BUNDY 30/12/76*/


STEADY1 :-
  CUE(PATHSYS(S,TOP,BOTTOM,LEFT,RIGHT)).

STEADY2 :-
  CHECKLIST(PASSERTA,[
    VEL(M,ZERO,0,DEPART), AT(M,TOP,DEPART),
    SIDE(M,TOP,LEFT,DEPART), SOLID(S),
    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',

/*BLOC,PRB*/
/*PROBLEM 3, STAGE 4*/
/*DE KLEERS SLIDING BLOCK PROBLEM*/
/*ALAN BUNDY SEPT 1976*/


/*CHANGE TO PATHSYS?*/
PATHINFO(_NAME,_LEND,_REND,_SLOPE,_CONV) :-
  CUE(PATHSYS(S,TOP,BOTTOM,LEFT,RIGHT)).

PROBINFO :-
  CHECKLIST(PASSERTA,[ PROBTYPE(ROLLER-COASTER,_T), PARTICLE(M),
    PATH(S0), PARTITION(S0,[S1,S2,S3]),
    END(S0,CA,LEFT), END(S0,CD,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 :- QA(AT(M,CD,_MOM),SOLVEINEQ(_X,_VAL)).


:- END.
```

```
:- 'NOLC'.

/*PROBLEM 4, STAGE 4*/
/*DE KLEERS LOOP THE LOOP PROBLEM*/
/*ALAN BUNDY 30/12/76*/

PATHINFO(_NAME,_LEND,_REND,_SLOPE,_CONV) :-
  CUE(PATHSYS(S,TOP,BOTTOM,LEFT,RIGHT)).

PROBINFO :-
  CHECKLIST(PASSERTA,[ PROBTYPE(ROLLER-COASTER,_T), PARTICLE(M),
    PATH(S0), PARTITION(S0,[S1,S2,S3,S4,S5]),
    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,[
    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.

  AL :-  QA(MOTION(M,S0,CA,LEFT,_PER),MIN(H,_MINVAL)).



:- END.
```

argstr

```
/* BRICK  argument list routines.

                entry points     :  argstruct(?,?,?,?).

                internals        :  NONE.

                uses             :  NONE.

                db active        :  NONE.

                db passive       :  NONE.

                variables        :  NONE.

*/


  argstruct(duration,2,
                [period,duration],
                [arg,val] ).

  argstruct(normal,2,
                [line,angle],
                [arg,val] ).

  argstruct(tangent,2,
                [line,angle],
                [arg,val] ).

  argstruct(angle,3,
                [line,angle,point],
                [arg,val,arg] ).

  argstruct(incline,3,
                [line,angle,point],
                [arg,val,arg] ).


  argstruct(distance,3,
                [particle,length,period],
                [arg,val,arg] ).

  argstruct(separation,5,
                [point_of_ref,point_of_ref,length,angle,time],
                [arg,arg,val,val,arg] ).

  argstruct(ground,2,
                [line,length],
                [arg,val] ).

  argstruct(drop,3,
                [path,point,length],
                [arg,arg,val] ).

  argstruct(typical_drop,3,
                [path,point,length],
                [arg,arg,val] ).

%  argstruct(radius,2,
%                [line,length],
```

```
%                [arg,val] ).

  argstruct(constlength,2,
                [line,length],
                [arg,val] ).

  argstruct(varlength,3,
                [line,length,time],
                [arg,val,arg] ).


  argstruct(mass,3,
                [object,mass,time],
                [arg,val,arg] ).


  argstruct(tension,3,
                [string,force,time],
                [arg,val,arg] ).

  argstruct(reaction,5,
                [object,object,force,angle,time],
                [arg,arg,val,val,arg] ).


  argstruct(friction,5,
                [object,object,force,angle,time],
                [arg,arg,val,val,arg] ).

  argstruct(holding,5,
                [object,object,force,angle,time],
                [arg,arg,val,val,arg] ).


  argstruct(vel,4,
                [point_of_ref,vel,angle,time],
                [arg,val,val,arg] ).

  argstruct(relvel,5,
                [point_of_ref,point_of_ref,vel,angle,time],
                [arg,arg,val,val,arg] ).


  argstruct(accel,4,
                [point_of_ref,accel,angle,time],
                [arg,val,val,arg] ).

  argstruct(relaccel,5,
                [point_of_ref,point_of_ref,accel,angle,time],
                [arg,arg,val,val,arg] ).


  argstruct(coeff,2,
                [path,constant],
                [arg,val] ).

  argstruct(elastic,2,
                [string,constant],
                [arg,val] ).
```

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

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

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

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

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

argstruct(at,3,
              [point_of_ref,place,moment],
              [ars,val,ars] ).

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

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

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


argstruct(c_of_gravity,2,
              [object,point],
              [ars,val] ).


argstruct(measure,3,
              [quantity,number,unit],
              [ars,ars,ars] ).

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

argstruct(elasticity,3,
              [string,force,time],
              [ars,val,ars]).

argstruct(natlength,3,
              [string,length,time],
              [ars,val,ars]).
```

```
argstruct(extension,3,
                [string,length,time],
                [arg,val,arg]).

/* END OF BRICK  argument list routines. */
```

```
/*CONTRL*/
/*TOP LEVEL ROUTINES*/
/*ALAN BUNDY MAY 1977*/


/*TOP LEVEL*/
/*---------*/

/*STANDARD TOP LEVEL ROUTINE*/

GO :- SOUGHTS(_XS), GIVENS(_GS),
    TRACE(ATTEMPTING-TO-SOLVE-FOR-_XS-IN-TERMS-OF-_GS,5),
    GETEQNS(_XS,_GS,NIL,_ES,_XS1),
    TRACE(EQUATIONS-EXTRACTED,2), PPR(_ES) ,
    CONVERT(_ES,_ES1), TRACE(CONVERTED-TO,2), PPR(_ES1),
    SIMPLIFY(_ES1,_ES2), TRACE(SIMPLIFIED-TO,2), PPR(_ES2),
    MAPLIST(CCMEASURE,_XS1,_XS2), TRACE(UNKNOWNS-ARE-_XS2,2),
    SIMSOLVE(_ES2,_XS2,_ANS), TRACE(ANSWER-IS,2), PPR(_ANS).


/*GENERAL QUESTION ANSWERING ROUTINE*/
QA(_QUAL,_QUAN) :-
  _QUAL, FINDALL(PROVISO,_CONDLIST),
  TRACE(_QUAL-OK-PROVIDED-_CONDLIST,2),
  SOLVEALL(_CONDLIST,_SOLNS),
  SUBST(_SOLNS,_CONDLIST,_NCONDLIST),
  TRACE(NEW-CONDITIONS-ARE-_NCONDLIST,2),
  APPLY(_QUAN,_NCONDLIST.[]).


/*CALL MARPLES TO UNPACK PROVISOS*/

SOLVEALL(_CONDLIST,_ANS) :-
  SETUP(_CONDLIST,_ES,_XS1), CRUNCH(_ES,_XS1,_ANS).

SETUP(_CONDLIST,_ES,_XS1) :- SWORDSIN(_CONDLIST,_VARS),
    GIVENS(_GS), SUBTRACT(_VARS,_GS,_XS),
    TRACE(ATTEMPTING-TO-SOLVE-FOR-_XS-IN-TERMS-OF-_GS,5),
    GETEQNS(_XS,_GS,[],_ES,_XS1),
    TRACE(EQUATIONS-EXTRACTED,2), PPR(_ES).

CRUNCH(_ES,_XS1,_ANS) :-
    CONVERT(_ES,_ES1), TRACE(CONVERTED-TO,2), PPR(_ES1),
    SIMPLIFY(_ES1,_ES2), TRACE(SIMPLIFIED-TO,2), PPR(_ES2),
    MAPLIST(CCMEASURE,_XS1,_XS2), TRACE(UNKNOWNS-ARE-_XS2,2),
    SIMSOLVE(_ES2,_XS2,_ANS), TRACE(ANSWER-IS,2), PPR(_ANS).

/*APOLOGIES FOR ABSENCE*/
/*--------------------*/

CONVERT(_ES,_ES) :- TRACE(CONVERT-NOT-LOADED,3).

CCMEASURE(_X,_Y) :- TRACE(CCMEASURE-NOT-LOADED,3).

SIMPLIFY(_ES,_ES) :- TRACE(SIMPLIFY-NOT-LOADED,3).

SIMSOLVE(_E,_X,_E) :- TRACE(SIMSOLVE-NOT-LOADED,3).
```

```
SETUP(_CL,_ES,_XS1) :- TRACE(SETUP-NOT-LOADED,3).

CRUNCH(_ES,_XS1,_ES) :- TRACE(CRUNCH-NOT-LOADED,3).

SUBST(_SOLNS,_CL,_CL) :- TRACE(SUBST-NOT-LOADED,3).

EVAL(_CONDS) :- TRACE(EVAL-NOT-LOADED,3).

MIN(_Q,_MV,_CONDS) :- TRACE(MIN-NOT-LOADED,3).


/*SCHEMATA (AND CUES)*/
/*-------------------*/

/*CUE IN SCHEMA*/
CUE(_KEY) :- SCHEMA(_KEY,_DECL,_ASS,_DEF),
  TRACE(PULLING-IN-SCHEMA-_KEY,6), NWL(6),
  TRACE(DECLARATIONS,7),CHECKLIST(CALL,_DECL),NWL(7),
  TRACE(ASSERTIONS,7), CHECKLIST(PASSERTA,_ASS),NWL(7),
  TRACE(DEFAULTS,7), CHECKLIST(PASSERTZ,_DEF),NWL(7),
  PASSERTA(_KEY),NWL(7), !.

SCHEMA(LINESYS(_LINE,_LEND,_REND),
  [CREATE(LINE(_LINE)),
   CC(END(_LINE,_LEND,LEFT)), CC(END(_LINE,_REND,RIGHT))],
  [POINT(_LEND),POINT(_REND)],
  [] ).

SCHEMA(TIMESYS(_PER,_MOM1,_MOM2),
  [CREATE(PERIOD(_PER)),CC(INITIAL(_PER,_MOM1)),
    CC(FINAL(_PER,_MOM2))],
  [MOMENT(_MOM1), MOMENT(_MOM2)],
  [] ).

SCHEMA(MOTSYS(_PART,_PATH,_START,_SIDE1,_PER),
  [CC(FAREND(_PATH,_FINISH,_START)), CC(FINAL(_PER,_END)),
   CC(INCLINE(_PATH,_ANG1,_FINISH)),CONDTURN3(_PATH,_START,_ANG1,_ANG2),
   CC(VEL(_PART,_V,_ANG2,_END)), ASSSP(_PART,_FINISH,_END),
   CC(INITIAL(_PER,_BEGIN)), ASSSP(_PART,_START,_BEGIN)],
  [DBC(MOTION(_PART,_PATH,_START,_SIDE1,_PER)),
   DBC(AT(_PART,_FINISH,_END)) ],
  [(PROBTYPE(MOTION-IN-A-STRAIGHT-LINE) :-
    (CONSTVEL(_PART,_PER);CONSTACCEL(_PART,_PER))),
   (_V)ZERO :- TOP(_PATH,_START), PROBTYPE(ROLLER-COASTER))] ).


SCHEMA(PATHSYS(_NAME,_LEND,_REND,_SLOPE,_CONV),
  [CUE(LINESYS(_NAME,_LEND,_REND))],
  [PATH(_NAME), CONCAVITY(_NAME,_CONV), SLOPE(_NAME,_SLOPE)],
  [] ).

SCHEMA(STRINGSYS(_SYS,_LBIT,_MIDPT,_RBIT,_TIME),
  [CUE(LINESYS(_SYS,_LEND,_REND)),
   CUE(LINESYS(_LBIT,_LEND,_MIDPT)),
   CUE(LINESYS(_RBIT,_MIDPT,_REND))],
  [PARTITION(_SYS,[_LBIT,_RBIT]),STRING(_SYS),
   STRING(_LBIT), STRING(_RBIT),
   CONCAVITY(_LBIT,STLINE), CONCAVITY(_RBIT,STLINE)],
  [ELASTIC(_SYS,ZERO), MASS(_SYS,ZERO,_TIME)] ).

SCHEMA(PULLSYS_MIN(_SYS,_PULL,_STR,_DIR1,_DIR2,_TIME),
  [CUE(STRINGSYS(_STR,_LBIT,_MIDPT,_RBIT,_TIME)),
```

```
   ASSSP(_PULL,_MIDPT,_TIME)],
  [PROBTYPE(PULLEY), PARTICLE(_PULL),
   TANGENT(_LBIT,_DIR1), TANGENT(_RBIT,_DIR2),
   (TENSION(_LBIT,_T,_TIME) :- COEFF(_PULL,ZERO),TENSION(_STR,_T,_TIME)),
   (TENSION(_RBIT,_T,_TIME) :- COEFF(_PULL,ZERO),TENSION(_STR,_T,_TIME))],
   [COEFF(_PULL,ZERO), MASS(_PULL,ZERO,_TIME)] ).

/*DEFAULT PULLEY IS FIXED*/


SCHEMA(PULLSYS_MAJ(_SYS,_PULL,_STR,_P1,_DIR1,_P2,_DIR2,_TIME),
  [CUE(PULLSYS_MIN(_SYS,_PULL,_STR,_DIR1,_DIR2,_TIME)),
   END(_STR,_LEND,LEFT), END(_STR,_REND,RIGHT),
   ASSSP(_P1,_LEND,_TIME), ASSSP(_P2,_REND,_TIME) ],
  [CONSTACCEL(_P1,_TIME), CONSTACCEL(_P2,_TIME)],
  [] ).

/*ASSERTING UNKNOWNS*/
/*------------------*/

/*ASSERT INFORMATION ABOUT UNKNOWN OF TYPE CLASS IN DEFN*/
ASSUNK(_DEFN,_CLASS) :- PASSERTA(_DEFN),
  _DEFN=.._PROP._ARGS, SEPERATE(_PROP,_ARGS,_PARGS,_FARGS),
  ASSTYPE(_FARGS,_PROP,_CLASS).

/*TWO CASES, Q IS VECTOR OR SCALAR*/
ASSTYPE([_Q],_PROP,_CLASS) :-
  ASSERTA(KIND(_Q,_PROP)), ASSAPP(_CLASS,_Q).

ASSTYPE([_Q,_DIR],_PROP,_CLASS) :-
  ASSERTA(KIND(_Q,_PROP)), ASSERTA(KIND(_DIR,ANGLE)),
  ASSAPP(_CLASS,_Q).

/*2ND ORDER ASSERT*/
ASSAPP(_CLASS,_Q) :-
  _L=.. _CLASS.[_Q], ASSERTA(_L).




/*FUNCTION CALL PREDICATE CALL*/
/*----------------------------*/

/*CREATE OR NON-CREATE CALL*/
CC(_L) :- FPC(_L,ON).
NCC(_L) :- FPC(_L,OFF).

/*FUNCTION OR PREDICATE CALL*/
FPC(_L,_SW) :- TRACE(CALLING-_L,7),FAIL.
FPC(_L,_SW) :- _L=.._PROP._ARGS, CHECKLIST(BOUND,_ARGS),
  CHECKCALL(_L,_SW), TRACE(CCALL-OF-_L-SUCCESSFUL,7).
FPC(_L,_SW) :- CHECKARGS(_L), !, FC(_L,_SW),
  TRACE(FCALL-OF-_L-SUCCESSFUL,7).
FPC(_L,_SW) :- _L,
  TRACE(PCALL-OF-_L-SUCCESSFUL,7).

/*CHECK CALL*/
CHECKCALL(_L,_SW) :- SILLY(_L), !, FAIL.
CHECKCALL(_L,ON) :- ASSERTA(_L), !.
CHECKCALL(_L,OFF) :- _L, !.

/*FUNCTION CALL*/
```

```
PC(_L,_SW) :- _L, !.
PC(_L,ON) :- CCFLAG(ON), DECLARE(_L), !.

CCFLAG(ON).

/*NO BACKUP FOR GROUND CALL*/
PC(_L) :- TRACE(CALLING-_L,7), FAIL.
PC(_L) :- _L=.._PROP._ARGS,
  CHECKLIST(BOUND,_ARGS), _L,
  TRACE(_L-CHECK-SUCCESSFUL,7), !.
PC(_L) :- _L, TRACE(CALL-OF-_L-SUCCESSFUL,7).


/*PREDICATE ARGS ALL BOUND*/
CHECKARGS(_L) :- _L=.. _PROP._ARGS,
  SEPERATE(_PROP,_ARGS,_PARGS,_FARGS),
  CHECKLIST(BOUND,_PARGS).

/*IS CALL SILLY?*/
SILLY(_L) :- _L=.. _PROP._ARGS, SEPERATE(_PROP,_ARGS,_PARGS,_FARGS),
  SEPERATE(_PROP,_NARGS,_PARGS,_NFARGS),
  _NL =.. _PROP._NARGS, DBC(_NL), DIFF(_FARGS,_NARGS).

/*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), !.

/*CREATIVE CALL*/
/*-------------*/

/*DECLARATIONS*/

DECLARE(_L) :- _L =.. _PROP._ARGS,
    MESS(_PROP,_ARGS), ASSERTA(_L).


MESS(_PROP,_ARGS) :-
  ARGSTRUC(_PROP,_ROLES),
  GETOBJ(_ROLES,_ROLES1,_OBJ,_ARGS,_ARGS1),
  GETVAL(_ROLES1,_ROLES2,_VAL,_ARGS1,_ARGS2),
  MAKE(_PROP,_VAL),
  WRITE(LET-_VAL-BE-THE-_PROP-OF-_OBJ),
  PREPS(_ROLES2,_ARGS2), NL.

/*GET VALUE AND OBJECT ARGS*/
GETOBJ(_ROLES1,_ROLES2,_OBJ,_ARGS1,_ARGS2) :-
  REMOVE(OBJ,_ROLES1,_ROLES2,_OBJ,_ARGS1,_ARGS2), !.

GETOBJ(_ROLES1,_ROLES2,_TIME,_ARGS1,_ARGS2) :-
```

```
  REMOVE(TIME,_ROLES1,_ROLES2,_TIME,_ARGS1,_ARGS2), !.

GETVAL(_ROLES1,_ROLES2,_VAL,_ARGS1,_ARGS2) :-
  REMOVE(VAL,_ROLES1,_ROLES2,_VAL,_ARGS1,_ARGS2), !.

GETVAL(_ROLES1,_ROLES2,_QUAN,_ARGS1,_ARGS2) :-
  REMOVE(QUAN,_ROLES1,_ROLES2,_QUAN,_ARGS1,_ARGS2), !.

GETVAL(_ROLES1,_ROLES2,_ANG,_ARGS1,_ARGS2) :-
  REMOVE(ANG,_ROLES1,_ROLES2,_ANG,_ARGS1,_ARGS2), !.

/*REMOVE ARG PLAYING ROLE IN ARGS*/
REMOVE(_ROLE,_ROLE._RTL,_RTL,_ARG,_ARG._ATL,_ATL).

REMOVE(_R1,_R2._RTL1,_R2._RTL2,_A1,_A2._ATL1,_A2._ATL2) :-
  REMOVE(_R1,_RTL1,_RTL2,_A1,_ATL1,_ATL2).

/*EXTRA PREPOSITIONAL PHRASES*/
PREPS([],[]).

PREPS(OBJ._RTL,_PT._ATL) :- POINT(_PT),
  WRITE(-AT-_PT), !, PREPS(_RTL,_ATL).

PREPS(OBJ._RTL,_PT._ATL) :- POINT_OF_REF(_PT),
  WRITE(-RELATIVE-TO-_PT), !, PREPS(_RTL,_ATL).

PREPS(OBJ._RTL,_PAR._ATL) :-
  WRITE(-ON-THE-_PAR), !, PREPS(_RTL,_ATL).

PREPS(ANG._RTL,_DIR._ATL) :-
  MAKE(ANGLE,_DIR),
  WRITE(-IN-DIRECTION-_DIR), !,PREPS(_RTL,_ATL).

PREPS(TIME._RTL,_PER._ATL) :- PERIOD(_PER),
  WRITE(-DURING-_PER), !, PREPS(_RTL,_ATL).

PREPS(TIME._RTL,_MOM._ATL) :-
  WRITE(-AT-_MOM), !, PREPS(_RTL,_ATL).


MAKE(_PROP,_Q) :- GENSYM(_PROP,_Q), !, ASSERTA(KIND(_Q,_PROP)).

MAKE(_PROP,_Q).


/*DECLARATION OF NON FUNCTION VALUES*/

CREATE(_L) :- _L=.._PROP.[_ARG],CREATE1(_L,_PROP,_ARG).

CREATE1(_L,_PROP,_ARG) :- GENSYM(_PROP,_ARG),!,ASSERTA(_L),
  TRACE(LET-_ARG-BE-A-NEW-_PROP,2).
CREATE1(_L,_PROP,_ARG).

/*INFORMATION ABOUT QUANTITY RELATIONS*/
/*-------------------------------------*/


/*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,FORCE]).

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 RELATIONS*/

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]), !.


.
.
```