```
;; MOFI.SUB - All Alan's Moments of Inertia stuff
;;
mofi.sub          ;; This file
mofi.             ;  Program load file
ctmeas.
defn.
coord.
senkn.
form1.
form2.
inf1.
inf2.
mk1.
typ1.
patch.
fndb.
sel.
qik.
mofi4.old         ;; Problem files
mofi1.prb         ;;
mofi2.prb         ;;
mofi3.prb         ;;
mofi4.prb         ;;
mofi5.prb         ;;
mofi6.prb         ;;
mofi7.prb         ;;
mofi8.prb         ;;
mofi9.prb         ;;
mofi10.prb        ;;
mofi11.prb        ;;
mofi1.sol         ;; Solution files (traces)
mofi2.sol         ;;
mofi3.sol         ;;
mofi4.sol         ;;
mofi5.sol         ;;
mofi6.sol         ;;
mofi7.sol         ;;
mofi8.sol         ;;
mofi9.sol         ;;
mofi10.sol        ;;
mofi11.sol        ;;
```

```
/* MOFI : All the bits for the Moment of inertia problems

                                        Updated: 10 February 81
*/


:- [
     sel,                 % Utilities for conjunctions

     ctmeas,              % Fibre Schema
     defn,                % Algebraic Shape Definitions
     genkn,               % Some general knowledge
     fndb,                % Date base entry with function properties

     patch,               % Various temporary patches
     qik,                 % Quick save and restore
     'arith:polyis',      % for poly_form
     'press:misc',        % for mult_occ
     'arith:polpak',      % for poly
     'press:match'        % for recomp
   ],


   load([ mk1,            % Meta level knowledge
          typ1,           % Type hierarchy
          inf1,           % Inference rules (length, area)
          inf2,           % Inference rules (mass, rofs)
          form1,          % Formulae (mass)
          form2           % Formulae (rofs)
        ]).
```

```
/* CTMEAS : Creation of Continuous Measure Systems

                                                        Updated: 1st April 1981
*/

                                        % continuous measure system already known

cont_meas(Obj,X0,Axis,Fibre,A,B) :-
        cont_meas1(Obj,X0,Axis,Fibre,A,B).


                                        % make continuous measure system

cont_meas(Obj,X0,axis(Y),Fibre,A,B) :-
        get_defn(Obj,origin,Defn1),
        sel(Defn1, A =< X =< B, VAR=X0, Defn2),
        uniform(X,Obj),
        gensym(X,X0),
        subst(X=X0,Defn2,Defn3),        % replace any Xs on right of =
        VAR = X,                        % put in the X on the left of =
        move_origin(Y,X,X0,Defn3,origin,Defn4,NewOrigin),
        rec_defn(Fibre,NewOrigin,Defn4),
        assert( const(X0) ),            % hack to prevent X0 being solved for!
        assert(cont_meas1(Obj,X0,axis(Y),Fibre,A,B)).



                                        % Find definition of object

get_defn(Obj,Origin,Defn) :-
        type(Shape,Obj),
        is_defn(Shape,Obj,Origin,Defn,Relns),
        checklist(ncc,Relns).


                                        % Recognise definition of object

rec_defn(Obj,Origin,Defn) :-
        is_defn(Shape,Obj,Origin,Defn,Relns),
        gensym(Shape,Obj),
        checklist(fn_dbentry,Relns),
        trace('\n %t is new fibre defined by %l\n', [Obj,Relns], 2).


                                        % Test for uniform fibre

uniform(X,_) :- distance_coord(X), !.   % either parameter is not angle

uniform(_,Obj) :-  ncc body1d(Obj), !.  % or fibre is 0 dimensional


                                        % Types of coordinate

distance_coord(x).
distance_coord(y).
distance_coord(z).
distance_coord(r).

angle_coord(theta).
angle_coord(phi).
```

```
/* Move coordinate system if axis of rotation not
perpendicular to continuous measure coordinate */

move_origin(Y,X,X0,Defn,Orgn,Defn,Orgn).          % they are perpendicular

move_origin(X,X,X0,ODefn,OOrgn,NDefn,NOrgn) :-
        sel(ODefn,X=X0,X=0,NDefn),                % axis and fibre coord
        ncc tangent(axis(X),Dir),                 % are the same
        define_orgn(X0,Dir,OOrgn,NOrgn).


move_origin(rr,C,C0,ODefn,OOrgn,NDefn,NOrgn) :-
        ncc tangent(axis(rr), [Alpha,0]),
        get_incr(C,C0,Alpha,X0,Y0,R0),
        update_defns(X0,Y0,ODefn,NDefn),
        define_orgn(R0,[Alpha,0],OOrgn,NOrgn).

/* Get x, y and r increments of coordinate change */
get_incr(x,X0,Alpha,X0,X0/tan(Alpha),X0/cos(Alpha)).
get_incr(y,Y0,Alpha,Y0*tan(Alpha),Y0,Y0/sin(Alpha)).


/* Update definitions by replacing old coordinates by new */

update_defns(X0,Y0,OXineq & OYineq & Zineq, NXineq & NYineq & Zineq) :-
        update_ineq(X0,OXineq,NXineq),
        update_ineq(Y0,OYineq,NYineq).

update_ineq(C0, C=V, C=V1) :-
        poly_form(V+(-1)*C0,V1).

update_ineq(C0, A=<C=<B, A1=<C=<B1) :-
        poly_form(A+(-1)*C0,A1), poly_form(B+(-1)*C0,B1).


/* assert definitions of new origin */

define_orgn(R0,Dir,OOrgn,NOrgn) :-
        gensym(origin,NOrgn),
        fn_dbentry( on(NOrgn,axis(rr)) ),
        fn_dbentry( separation(OOrgn,NOrgn,R0,Dir) ).
```

```
/* DEFN : Algebraic Definitions of Ideal Objects

                                                   Updated: 1st April 1981
*/


                          % sphere in cylindrical polars

is_defn(sphere, Sph, Origin, 0=<r=<sqrt(A^2-z^2) & 0=<theta=<2*pi & -A=<z=<A,
        [isa(sphere,Sph), centre(Sph,Origin), radius(Sph,A)] ).



                          % cylinder in cylindrical polars

is_defn(cylinder, Cyl, Origin, 0=<r=<A & 0=<theta=<2*pi & 0=<z=<H,
        [isa(cylinder,Cyl), centre(Cyl,Origin),
                radius(Cyl,A), height(Cyl,H)] ).



                          % tube in cylindrical polars

is_defn(tube, Tbe, Origin, r=A & 0=<theta=<2*pi & 0=<z=<H,
        [isa(tube,Tbe), centre(Tbe,Origin),
                radius(Tbe,A), height(Tbe,H)] ).



                          % cone in cylindrical polars

is_defn(cone, Cne, Origin, 0=<r=<A*(H-z)/H & 0=<theta=<2*pi & 0=<z=<H,
        [isa(cone,Cne), centre(Cne,Origin),
                radius(Cne,A), height(Cne,H)] ).



                          % hollow sphere in cylindrical polars

is_defn(shell, Shll, Origin, r=sqrt(A^2-z^2) & 0=<theta=<2*pi & -A=<z=<A,
        [isa(shell,Sph), centre(Sph,Origin), radius(Sph,A)] ).



                          % circular disc in cylindrical polar coordinates

is_defn(disc, Dsc, Origin, 0=<r=<A & 0=<theta=<2*pi & z=0,
        [isa(disc,Dsc), centre(Dsc,Origin), radius(Dsc,A),
        meets(axis(z),Dsc,Origin)] ) .

                          % circular ring in cylindrical polar coordinates

is_defn(ring, Rng, Origin, r=A & 0=<theta=<2*pi & z=0,
        [isa(ring,Rng), centre(Rng,Origin), radius(Rng,A),
        meets(axis(z),Rng,Origin)] ).

                          % radial line in spherical polar coordinates
```

```
is_defn(line, Lne, Origin, A=<r=<B & theta=T & phi=P,
        [isa(line,Lne),
        line_sys(Lne,Lend,Rend),
        isa(point,Lend), isa(point,Rend),
        on(Origin,Lne),
        separation(Origin,Lend,A,[T,P]), separation(Origin,Rend,B,[T,P])] ) .

                         % horizontal line in cartesian coordinates

is_defn(line, Lne, Origin, A=<x=<B & y=0 & z=0,
        [isa(line,Lne), line_sys(Lne,Lend,Rend),
        isa(point,Lend), isa(point,Rend),
        isa(point,Origin),
        on(Origin,Lne),
        separation(Origin,Lend,A,[0,0]), separation(Origin,Rend,B,[0,0])] ) .


                         % point in spherical polar coordinates

is_defn(point, Pt, Origin, r=R & theta=T & phi=P,
        [isa(point,Pt), separation(Origin,Pt,R,[T,P])] ).


                         % point in cylindrical polar coordinates

is_defn(point, Pt, Origin, r=R & theta=T & z=0,
        [isa(point,Pt),
        separation(Origin,Pt,R,[T,0])] ).

/*                       % point in cartesian coordinates

is_defn(point, Pt, Origin, x=X & y=Y & z=Z,
        [isa(point,Pt),
        separation(Origin,Pt,sqrt(X^2+Y^2+Z^2),
                [arctan(Y/X),arctan(Z/sqrt(X^2+Y^2))]) ] ).
*/


                         % rectangle in cartesian coordinates

is_defn(rectangle, Rect, Origin, A=<x=<B & C=<y=<D & z=0,
        [isa(rectangle,Rect), quad_sys(Rect,Top,Bottom,Left,Right),
        isa(line,Top), isa(line,Bottom), isa(line,Left), isa(line,Right),
        perp_dist(Origin,Top,D,[90,0]), perp_dist(Origin,Bottom,C,[90,0]),
        perp_dist(Origin,Left,A,[0,0]), perp_dist(Origin,Right,B,[0,0])] ).


                         % parallelogram in cartesian coordinates

is_defn(parallelogram, Para, Origin,
                A/sin(E)+y/tan(E)=<x=<B/sin(E)+y/tan(E) & C=<y=<D & z=0,
        [isa(parallelogram,Para), quad_sys(Para,Top,Bottom,Left,Right),
        isa(line,Top), isa(line,Bottom), isa(line,Left), isa(line,Right),
        perp_dist(Origin,Top,D,[90,0]),
        perp_dist(Origin,Bottom,C,[90,0]),
        perp_dist(Origin,Left,A,[E-90,0]),
```

```
perp_dist(Origin,Right,B,[E-90,0])] ),
```

```
/* MK1 : Meta level knowledge for the Moment of inertia problems

                                                        Updated: 10 February 81
*/


{ meta_knowledge }.
%-------------------


argstruct(on,2,
                  [point,object],
                  [arg,arg]).

argstruct(area,2,
                  [object,area],
                  [arg,val]).

argstruct(basetype,2,
                  [type,object],
                  [arg,arg]).

argstruct(centre,2,
                  [object,point],
                  [arg,val]).

argstruct(dist_along,4,
                  [object,point,point,length],
                  [arg,arg,arg,arg]).

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

argstruct(mass_per_vol,2,
                  [object,mass],
                  [arg,val]).

argstruct(mass_per_area,2,
                  [object,mass],
                  [arg,val]).

argstruct(mass_per_length,2,
                  [line,mass],
                  [arg,val]).

argstruct(mass,2,
                  [object,mass],
                  [arg,val]).

argstruct(meets,3,
                  [line,object,point],
                  [arg,arg,arg]).

argstruct(radius,2,
                  [object,length],
                  [arg,val]).

argstruct(height,2,
                  [object,length],
```

```
                [arg,val]).

argstruct(rad_of_gyr,3,
                [object,line,rofg],
                [arg,arg,val]).

argstruct(perp_dist,4,
                [object,object,length,angle],
                [arg,arg,val,val]).

argstruct(separation,4,
                [point,point,length,angle],
                [arg,arg,val,val]).

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

argstruct(quad_sys,5,
                [object,line,line,line,line],
                [arg,val,val,val,val]).

argstruct(cont_meas,6,
                [object,scalar,point,object,scalar,scalar],
                [arg,val,arg,arg,val,val]).


{ normal_form }.
%--------------

defn( separation(Obj1,Obj2,Sep,Angle), edb,
                [ sameclass(Obj1,Obj2,[Sep|Angle],sep(always)),
                  merge_sep(Obj1,Obj2,[Sep|Angle],always) ],
                [] ).


/* temporarily suspended
defn( line_sys(Line,Lend,Rend),  dbinf,
                ( isa(line,Line) & end(Line,Lend,left) &
                  end(Line,Rend,right) ),
                [] ).

defn( quad_sys(Quad,Top,Bot,Left,Right), dbinf,
                ( line_sys(Top,TL,TR) & line_sys(Bot,BL,BR) &
                  line_sys(Left,BL,TL) & line_sys(Right,BR,TR) ),
                [] ).
*/
```

```
*START* User BUNDY    HPS [400,405]  Job  TYP1 Seq. 8138 Date 02-Apr-81 14:51:1
File: DSKA:TYP1<005>[400,405,MYMOFI] Created: 02-Apr-81 11:46:07 Printed: 02-Apr
QUEUE Switches:  /FILE:ASCII /COPIES:1 /SPACING:1 /LIMIT:415 /FORMS:NORMAL
```

```
/* TYP1 : Part of type hierarchy for Moments of Inertia problems */


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


  body0d -> object.
  body1d -> object.
  body2d -> object.
  body3d -> object.

  point -> body0d.
  line -> body1d.
  ring -> body1d.
  square -> rectangle.
  rectangle -> parallelogram.
  parallelogram -> body2d.
  disc -> body2d.
  shell -> body2d.
  tube -> body2d.
  cube -> body3d.
  sphere -> body3d.
  cylinder -> body3d.
  cone -> body3d.


  mass -> scalar.
  length -> scalar.
  area -> scalar.
  vol -> scalar.
  rofs -> scalar.
  angle -> scalar.
```

```
/* INF1 : Inference rules concerning the lengths, areas etc. of
                  various kinds of body

                                                  Updated: 10 February 81
*/


                           % The length of a line is the distance between
                           % the end points

line_length(Lne,B-A)
    <-- line_sys(Lne,Lend,Rend) & on(Pt,Lne) &
        separation(Pt,Lend,A,Dir) & separation(Pt,Rend,B,Dir).
                           % needs generalizing to any point


                           % The length of a ring based on its radius

line_length(Ring,2*pi*R)
    <-- ring(Ring) &
        radius(Ring,R).


                           % The area of a rectangle is the product of the
                           % lengths of the sides

area(Rect,(D-C)*(B-A))
    <-- rectangle(Rect) &
        quad_sys(Rect,Top,Bottom,Left,Right) &
        perp_dist(Pt,Top,D,[90,0]) & perp_dist(Pt,Bottom,C,[90,0]) &
        perp_dist(Pt,Left,A,[0,0]) & perp_dist(Pt,Right,B,[0,0]).


                           % The area of a parallelogram is the product of the
                           % lengths of a side and the width

area(Para,(D-C)*(B/sin(E)-A/sin(E)))
    <-- parallelogram(Para) &
        quad_sys(Para,Top,Bottom,Left,Right) &
        perp_dist(Pt,Top,D,[90,0]) & perp_dist(Pt,Bottom,C,[90,0]) &
        perp_dist(Pt,Left,A,[E-90,0]) & perp_dist(Pt,Right,B,[E-90,0]).


                           % The area of a disc

area(Disc,pi*R^2)
    <-- disc(Disc) &
        radius(Disc,R).


                           % The surface area of a sphere

area(Shll,4*pi*R^2)
    <-- shell(Shll) &
        radius(Shll,R).
```

```
                                    % The surface area of a cylinder

area(Cyl,2*pi*R*H)
     <-- tube(Cyl) &
          radius(Cyl,R) &
          height(Cyl,H).


                                    % The volume of a sphere.

vol(Sph,(4/3)*pi*R^3)
     <-- sphere(Sph) &
          radius(Sph,R).


                                    % The volume of a cylinder

vol(Cyl,pi*R^2*H)
     <-- cylinder(Cyl) &
          radius(Cyl,R) &
          height(Cyl,H).


                                    % The volume of a cone

vol(Cne,(1/3)*pi*R^2*H)
     <-- cone(Cne) &
          radius(Cne,R) &
          height(Cne,H).
```

```
/* INF2 : Other inference rules

                                                Updated: 5 February 81
*/

                                % The mass of a 0D fibre is the mass
                                % per length of its supporting body
                                % times its infinitesimal thickness.

mass(Fibre, d(X)*Mu)
    <-- { cont_meas1(Obj,X,Axis,Fibre,A,B) } &
        body1d(Obj) &
        mass_per_length(Obj,Mu).


                                % The mass per length of a 1D fibre is the
                                % mass per area of its supporting body
                                % times its infinitesimal thickness.

mass_per_length(Fibre,d(X)*Mu)
    <-- { cont_meas1(Obj,X,Axis,Fibre,A,B) } &
        body2d(Obj) &
        mass_per_area(Obj,Mu).


                                % The mass per area of a 2D fibre is the
                                % mass per volume of its supporting body
                                % times its infinitesimal thickness.

mass_per_area(Fibre, d(X)*Mu)
    <-- { cont_meas1(Ob,X,Axis,Fibre,A,B) } &
        body3d(Obj) &
        mass_per_vol(Obj,Mu).

% The above three rules should be united into one.

                                % Radius of gyration of a point.

rad_of_gyr(Pt,Axis,K)
    <-- point(Pt) & perp_dist(Pt,Axis,K,Dir).


                                % Perpendicular distance of x point to y axis.

perp_dist(Pt,axis(y),R,[180,0])
    <-- separation(origin,Pt,R,[0,0]).
                                % rather special purpose!

                                % Perpendicular distance of xy point to z axis.

perp_dist(Pt,axis(z),R,[180+T,0])
    <-- separation(origin,Pt,R,[T,0]).
                                % rather special purpose!

                                % Perpendicular distance of xy point to rr axis.

perp_dist(Pt,axis(rr),R*sin(Alpha-T),[90+Alpha,0])
    <-- separation(origin,Pt,R,[T,0]),
```

```
tangent(axis(rr),[Alpha,0]),
                        % rather special purpose!
```

```
/* GENKN : Bits of general knowledge about certain objects

                                              Updated: 5 February 81
*/


isa(period,always).
isa(point,origin).
isa(line,axis(x)).
isa(line,axis(y)).
isa(line,axis(z)).

:- dbentry( tangent(axis(x),[0,0]) ).
:- dbentry( tangent(axis(y),[90,0]) ).
:- dbentry( tangent(axis(z),[0,90]) ).

:- dbentry( on(origin,axis(x)) ).
:- dbentry( on(origin,axis(y)) ).
:- dbentry( on(origin,axis(z)) ).
```

```
/* FORM1 : Formulae for calculating masses of bodies */


                              % Mass per length

relates(mass_per_length,[mass,length]).



prepare(mass_per_length,Q,mass,mass(Obj,M),
                situation(Obj) ) :- ncc body1d(Obj).

prepare(mass_per_length,Q,mass,mass_per_length(Obj,Mu),
                situation(Obj) ) :- ncc body1d(Obj).



isform(mass_per_length, situation(Obj),
                M = L*Mu )
    <-- mass(Obj,M) &
        mass_per_length(Obj,Mu) &
        length(Obj,L).



                              % Mass per area

relates(mass_per_area,[mass,area]).



prepare(mass_per_area,Q,mass,mass(Obj,M),
                situation(Obj) ) :- ncc body2d(Obj).

prepare(mass_per_area,Q,mass,mass_per_area(Obj,Mu),
                situation(Obj) ) :- ncc body2d(Obj).



isform(mass_per_area, situation(Obj),
                M = A*Mu )
    <-- mass(Obj,M) &
        mass_per_area(Obj,Mu) &
        area(Obj,A).

                              % Mass per volume

relates(mass_per_vol,[mass,vol]).



prepare(mass_per_vol,Q,mass,mass(Obj,M),
                situation(Obj) ) :- ncc body3d(Obj).

prepare(mass_per_vol,Q,mass,mass_per_vol(Obj,Mu),
                situation(Obj) ) :- ncc body3d(Obj).
```

```
isform(mass_per_vol, situation(Obj),
                   M = V*Mu )
    <-- mass(Obj,M) &
        mass_per_vol(Obj,Mu) &
        vol(Obj,V).
```

```
/* FORM2 : Formulae et al for the Moment of inertia problems */


{ problem_solving_rules }.
%-----------------------------


                         % Moment of Inertia

relates(moment_of_inertia,[mass,rofg]).


prepare(moment_of_inertia,Q,rofg,rad_of_gyr(Obj,axis(X),RG),
                situation(Obj,axis(X))).


isform(moment_of_inertia, situation(Obj,axis(X)),
                M*RG^2 = integrate(Mf*RGf^2, A,B,X))
      <-- mass(Obj,M) &                                    Yo
          rad_of_gyr(Obj,axis(X),RG) &
          { cont_meas(Obj,YO,axis(X),Fibre,A,B) } &
          mass(Fibre,Mf) &
          rad_of_gyr(Fibre,axis(X),RGf).


                         % Parallel Axes

relates(parallel_axes,[rofg]).


prepare(parallel_axes,Q,rofg,rad_of_gyr(Obj,Axis,RG),
          intersect? situation(Obj,Axis,Newaxis))
      <-- meets(Axis,Obj,Axpt) &
          centre(Obj,Origin) &
          { diff(Axpt,Origin) } &
         ? meets(Newaxis,Obj,Origin)? &
          parallel(Axis,Newaxis).


isform(parallel_axes, situation(Obj,RealAxis,CentreAxis),
                RG = RGC + A^2 )
      <-- rad_of_gyr(Obj,RealAxis,RG) &
          rad_of_gyr(Obj,CentreAxis,RGC) &
          perp_dist(RealAxis,CentreAxis,A,Dir).
```

```
/* FNDB : Temporary stuff for doing function stuff on dbentry

                                                   Updated: 5 February 81

*/


fn_dbentry(X)
        :- groundtest(X,ground),
           !,
           trace('>-> %p\n',[X],4),
           dbentry(X).

                     Pred,N
fn_dbentry(X)
        :- functor(X,F,A),
           argstruct(Pred,N,Types,Fmap),
           X =.. [_|Args],
           maksolid(Fmap,Args,Types),
           trace('>*> %p\n',[X],4),
           dbentry(X),
           !.

fn_dbentry(X)
        :- error(ungr,[X],trace),
           continue.


              errmess(ungr,'Unground database entry: %p\n').


                              % Apply function properties

maksolid([],[],[]).

maksolid([F|Frest],[A|Arest],[T|Trest])
        :- maksl(F,A,T),
           maksolid(Frest,Arest,Trest).


maksl(arg,A,_) :- nonvar(A).

maksl(val,A,T) :- cgensym(T,A).
```

```
/* PATCH: Various temporary patches

                                                        Updated: 10 February 81
*/



                                        % Fix sameclass to deal with 3D vectors
                                        %  for separation. vecadd can handle these
                                        %  as well for simple cases.

:- retract(treeid(sep,_)),
   asserta(treeid(sep,[0,0,0])).
```

```
/* SEL : Utilities for selecting elements from conjunctions */

                                        % Select and construct

sel(A&Rest,A,B,B&Rest).

sel(X&Rest1,A,B,X&Rest2) :- !, sel(Rest1,A,B,Rest2).

sel(A,A,B,B).


                                        % Select only

sel(A&Rest,A).

sel(X&Rest,A) :- !, sel(Rest,A).

sel(A,A).
```

```
/* QIK */

s :- save('scra:foo').

r :- restore(foo).
```

```
/* MOFI1.PRB : A moment of inertia problem */

problem(mofi1,'Radius of Gyration of a line\n\n',[]).


line(l1).
point(l).
point(r).
line_sys(l1,l,r).
separation(origin,l,-a,[0,0]).
separation(origin,r,a,[0,0]).
on(origin,l1).
mass(l1,m).

rad_of_gyr(l1,axis(y),k).

given(a).
given(m).
sought(k).
```

```
/* MOFI2.PRB : 2nd Moments of Inertia problem */

problem(mofi2,'Radius of Gyration of a rectangle\n\n',[]).


rectangle(rect).
line(top).
line(bot).
line(left).
line(right).
quad_sys(rect,top,bot,left,right).
perp_dist(origin,top,b,[90,0]).
perp_dist(origin,bot,-b,[90,0]).
perp_dist(origin,left,-a,[0,0]).
perp_dist(origin,right,a,[0,0]).

mass(rect,m).

rad_of_gyr(rect,axis(y),k).

given(a).
given(b).
given(m).
sought(k).

                                % radius of gyration of a line

rad_of_gyr(Lne,axis(y),A/sqrt(3))
    <-- line_sys(Lne,Lend,Rend) &
        on(Pt,Lne) & on(Pt,axis(y)) &
        separation(Pt,Rend,A,[0,0]) &
        separation(Pt,Lend,-A,[0,0]).
```

```
/* MOFI3.PRB : 3rd Moment of Inertia problem */

problem(mofi3,'Radius of Gyration of an inclined line\n\n',[]):

line(l1).
point(l).
point(r).
line_sys(l1,l,r).
separation(origin,l,-a,[0,0]).
separation(origin,r,a,[0,0]).
on(origin,l1).
mass(l1,m).

line(axis(rr)).
on(origin,axis(rr)).
tangent(axis(rr),[alpha,0]).
rad_of_gyr(l1,axis(rr),k).

given(a).
given(alpha).
given(m).
sought(k).
```

START User BUNDY   HPS [400,405] Job TYP1 Seq. 8138 Date 02-Apr-81 14:51:1
File: DSKA:MOFI4.PRB<005>[400,405,MYMOFI] Created: 02-Apr-81 14:04:31 Printed: (
QUEUE Switches: /FILE:ASCII /COPIES:1 /SPACING:1 /LIMIT:415 /FORMS:NORMAL

```
/* MOFI4.PRB : 4th Moments of Inertia problem */

problem(mofi4,'Radius of Gyration of a parallelogram\n\n',[]).


parallelogram(pagrm).
line(top).
line(bot).
line(left).
line(right).
quad_sys(pagrm,top,bot,left,right).
perp_dist(origin,top,b,[90,0]).
perp_dist(origin,bot,-b,[90,0]).
perp_dist(origin,left,-a,[alpha-90,0]).
perp_dist(origin,right,a,[alpha-90,0]).

mass(pagrm,m).

line(axis(rr)).
on(origin,axis(rr)).
tangent(axis(rr),[alpha,0]).
rad_of_gyr(pagrm,axis(rr),k).

given(a).
given(b).
given(alpha).
given(m).
sought(k).

                          % radius of gyration of an inclined line

rad_of_gyr(Lne,Ax,A*sin(T2-T1)/sqrt(3))
    <-- line_sys(Lne,Lend,Rend) &
        on(Pt,Lne) & on(Pt,Ax) &
        tangent(Ax,[T2,P]) &
        separation(Pt,Rend,A,[T1,P]) &
        separation(Pt,Lend,-A,[T1,P]).
```

```
/* MOFI5.PRB : 5th Moments of Inertia p        */

problem(mofi5,'Radius of Gyration of a ring\n\n',[]).


ring(ring1).
centre(ring1,origin).
radius(ring1,a).
meets(axis(z),ring1,origin).

mass(ring1,m).

rad_of_gyr(ring1,axis(z),k).

given(a).
given(m).
sought(k).
```

```
/* MOFI6.PRB : 6th Moments of Inertia problem */

problem(mofi6,'Radius of Gyration of a Disc\n\n',[]).


disc(disc1).
centre(disc1,origin).
radius(disc1,a).
meets(axis(z),disc1,origin).

mass(disc1,m).

rad_of_gyr(disc1,axis(z),k).

given(a).
given(m).
sought(k).


                                % The radius of gyration of a ring about a
                                % perpendicular axis through its centre is its radius

rad_of_gyr(Ring,Axis,R)
     <-- ring(Ring) & centre(Ring,C) &
         meets(Axis,Ring,C) & radius(Ring,R).
```

```
/* MOFI7.PRB : 7th Moments of Inertia problem */

problem(mofi7,'Radius of Gyration of a sphere\n\n',[]).


sphere(sphere1).
centre(sphere1,origin).
radius(sphere1,a).

mass(sphere1,m).

rad_of_gyr(sphere1,axis(z),k).

given(a).
given(m).
sought(k).


                        % The radius of gyration of a disc about a
                        % perpendicular axis through its centre is its radius
                        % divided by root 2.

rad_of_gyr(Disc,Axis,R/sqrt(2))
    <-- disc(Disc) & centre(Disc,C) &
        meets(Axis,Disc,C) & radius(Disc,R).
```

START User BUNDY   HPS [400,405] Job MOFI8 Seq. 8102 Date 02-Apr-81 12:13:
File: DSKA:MOFI8.PRB<005>[400,405,MYMOFI] Created: 02-Apr-81 11:39:46 Printed: C
QUEUE Switches: /FILE:ASCII /COPIES:1 /SPACING:1 /LIMIT:105 /FORMS:NORMAL

```
/* MOFI8.PRB : 8th Moments of Inertia problem */

problem(mofi8,'Radius of Gyration of a shell\n\n',[]).


shell(shell1).
centre(shell1,origin).
radius(shell1,a).

mass(shell1,m).

rad_of_gyr(shell1,axis(z),k).

given(a).
given(m).
sought(k).


                              % The radius of gyration of a ring about a
                              % perpendicular axis through its centre is its radius

rad_of_gyr(Ring,Axis,R)
    <-- ring(Ring) & centre(Ring,C) &
        meets(Axis,Ring,C) & radius(Ring,R).
```

```
 *START* User BUNDY    HPS [400,405]  Job  MOFI9 Seq. 8112 Date 02-Apr-81 13:29:
File: DSKA:MOFI9.PRB<005>[400,405,MYMOFI] Created: 02-Apr-81 13:25:48 Printed: 0
QUEUE Switches:  /FILE:ASCII /COPIES:1 /SPACING:1 /LIMIT:310 /FORMS:NORMAL
```

```
/* MOFI9.PRB : 9th Moments of Inertia problem */

problem(mofi9,'Radius of Gyration of a cylinder\n\n',[]).


cylinder(cylinder1).
centre(cylinder1,origin).
radius(cylinder1,a).
height(cylinder1,h).

mass(cylinder1,m).

rad_of_gyr(cylinder1,axis(z),k).

given(a).
given(h).
given(m).
sought(k).

                             % The radius of gyration of a disc about a
                             % perpendicular axis through its centre is its radius
                             % divided by root 2.

rad_of_gyr(Disc,Axis,R/sqrt(2))
    <-- disc(Disc) & centre(Disc,C) &
        meets(Axis,Disc,C) & radius(Disc,R).
```

```
*START* User BUNDY    HPS [400,405]  Job  MOFI9 Seq. 8112 Date 02-Apr-81 13:29
File: DSKA:MOFI10.PRB<005>[400,405,MYMOFI] Created: 02-Apr-81 13:26:16 Printed:
QUEUE Switches:  /FILE:ASCII /COPIES:1 /SPACING:1 /LIMIT:310 /FORMS:NORMAL
```

```
/* MOFI10.PRB : 10th Moments of Inertia problem */

problem(mofi10,'Radius of Gyration of a cone\n\n',[]).


cone(cone1).
centre(cone1,origin).
radius(cone1,a).
height(cone1,h).
mass(cone1,m).

rad_of_gyr(cone1,axis(z),k).

given(a).
given(h).
given(m).
sought(k).


                                % The radius of gyration of a disc about a
                                % perpendicular axis through its centre is its radius
                                % divided by root 2.

rad_of_gyr(Disc,Axis,R/sqrt(2))
    <-- disc(Disc) & centre(Disc,C) &
        meets(Axis,Disc,C) & radius(Disc,R).
```

```
/* MOFI11.PRB : 11th Moments of Inertia problem */

problem(mofi11,'Radius of Gyration of a tube\n\n',[]).


tube(tube1).
centre(tube1,origin).
radius(tube1,a).
height(tube1,h).

mass(tube1,m).

rad_of_gyr(tube1,axis(z),k).

given(a).
given(h).
given(m).
sought(k).


                                % The radius of gyration of a ring about a
                                % perpendicular axis through its centre is its radius

rad_of_gyr(Ring,Axis,R)
     <-- ring(Ring) & centre(Ring,C) &
          meets(Axis,Ring,C) & radius(Ring,R).
```

```
yes
| ?- restore(save).

yes
| ?- input(mofi1).

Problem from file : mofi1.prb .

Radius of Gyration of a line

 Let l1 be a new line
 Let l be a new point
 Let r be a new point
 Note: m (of type mass) was used in a mass definition (2)
 Note: k (of type rofg) was used in a rad_of_gyr definition (3)

mofi1 problem read into data base.


yes
| ?- go.

** ERROR Type unknown -- [a]
    ( continue after error )

Attempting to solve for [k] in terms of [a,m]


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

 Applicable formulae : [parallel_axes,moment_of_inertia]
 (try  parallel_axes)
 (try  moment_of_inertia)
 Trying to apply strategy(moment_of_inertia,situation(l1,axis(y)))
 Let point1 be a new point

 point1 is new fibre defined by
     isa(point,point1)
     separation(origin,point1,r1,[0,0])


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

 Applicable formulae : [parallel_axes,moment_of_inertia]
 (try  parallel_axes)
 (try  moment_of_inertia)
 Trying to apply strategy(moment_of_inertia,situation(l1,axis(y)))
 Let mass_per_length1 be the mass_per_length of l1.
 Note: mass_per_length1 (of type mass) was used in a mass_per_length definition (2)

 Equation-1 : m*k^2=integrate(d(r1)*mass_per_length1*(a*-1+a+a+r1+a+a*-1+a*-1)^2,a*-1+a+a+a*-1+a*-1,a*-1+a+a,y)
 formed by applying : strategy(moment_of_inertia,situation(l1,axis(y)))

 This equation solves for k but introduces [mass_per_length1].

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

 So now I must solve for [mass_per_length1]
                given [k,a,m]
```

```
I am now trying to solve for mass_per_length1 without introducing any unknowns.
                                                                    [moment_of_inertia,mass_per_vol,mass_per_area,mass_per_length,resolve]
 Applicable formulae : 
 (try  moment_of_inertia)
 (try  mass_per_vol)
 (try  mass_per_area)
 (try  mass_per_length)
 Trying to apply strategy(mass_per_length,situation(l1))                   (a*-1+a+a+r1+a+a*-1+a*-1+a*-1+a*-1+r1*-1+a*-1+a+a))*mass_per_length1

 Equation-2 : m=(a*-1+a+a+r1+a+a*-1+a*-1+r1*-1+a*-1+a+a-
 formed by applying : strategy(mass_per_length,situation(l1))

 This equation solves for mass_per_length1.

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

 So now I must solve for []
                 given [mass_per_length1,k,a,m]


Equations extracted :                                                    ^2,a*-1+a+a+a*-1+a*-1,a*-1+a+a,y)
    m*k^2=integrate(d(r1)*mass_per_length1*(a*-1+a+a+r1+a+a*-1+a*-1)
    m=(a*-1+a+a+r1+a+a*-1+a*-1+r1*-1+a*-1+a+a-(a*-1+a+a+r1+a+a*-1+a*-1+a*-1+a*-1+r1*-1+a*-1+a+a))*mass_per_length1


yes
| ?- core      83456  (54272 lo-seg + 29184 hi-seg)
heap     49152 =  48243 in use +     909 free
global    1187 =     16 in use +    1171 free
local     1024 =     16 in use +    1008 free
trail      511 =      0 in use +     511 free
    0.05 sec. for 2 GCs gaining 993 words
    0.51 sec. for 26 local shifts and 43 trail shifts
   11.01 sec. runtime
```

```
| ?- input(mofi2).

Problem from file : mofi2.prb

Radius of Gyration of a rectangle

 Let rect be a new rectangle
 Let top be a new line
 Let bot be a new line
 Let left be a new line
 Let right be a new line
 Note: b (of type length) was used in a perp_dist definition (3)
 Note: [90,0] (of type angle) was used in a perp_dist definition (4)
 Note: -b (of type length) was used in a perp_dist definition (3)
 Note: -a (of type length) was used in a perp_dist definition (3)
 Note: [0,0] (of type angle) was used in a perp_dist definition (4)
 Note: a (of type length) was used in a perp_dist definition (3)
 Note: m (of type mass) was used in a mass definition (2)
 Note: k (of type rofg) was used in a rad_of_gyr definition (3)

mofi2 problem read into data base.


yes
| ?- go.

Attempting to solve for [k] in terms of [a,b,m]


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

 Applicable formulae : [parallel_axes,moment_of_inertia]
 (try  parallel_axes)
 (try  moment_of_inertia)
 Trying to apply strategy(moment_of_inertia,situation(rect,axis(y)))

** ERROR nfrec3   ( continue after error )
 Let line1 be a new line
 Let point1 be a new point
 Let point2 be a new point
 Let typ_pt1 be a new point

 line1 is new fibre defined by
    isa(line,line1)
    line_sys(line1,point1,point2)
    isa(point,point1)
    isa(point,point2)
    isa(point,typ_pt1)
    on(typ_pt1,line1)
    separation(typ_pt1,point1,-a,[0,0])
    separation(typ_pt1,point2,a,[0,0])


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

 Applicable formulae : [parallel_axes,moment_of_inertia]
 (try  parallel_axes)
 (try  moment_of_inertia)
 Trying to apply strategy(moment_of_inertia,situation(rect,axis(y)))
```

```
Let mass1 be the mass of line1.
Note: mass1 (of type mass) was used in a mass definition (2)

Equation-1 : m*k^2=integrate(mass1*((a*-1+a+a)/sqrt(3))^2,-b,b,y)
formed by applying : strategy(moment_of_inertia,situation(rect,axis(y)))

This equation solves for k but introduces [mass1].

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

So now I must solve for [mass1]
                  given [k,a,b,m]

I am now trying to solve for mass1 without introducing any unknowns.

Applicable formulae : [moment_of_inertia,mass_per_vol,mass_per_area,mass_per_length,resolve]
(try  moment_of_inertia)
(try  mass_per_vol)
(try  mass_per_area)
(try  mass_per_length)
Trying to apply strategy(mass_per_length,situation(linei))
(try  resolve)

No luck - I will now accept unknowns in solving for mass1.

Applicable formulae : [moment_of_inertia,mass_per_vol,mass_per_area,mass_per_length,resolve]
(try  moment_of_inertia)
(try  mass_per_vol)
(try  mass_per_area)
(try  mass_per_length)
Trying to apply strategy(mass_per_length,situation(line1))
Let mass_per_area1 be the mass_per_area of rect.
Note: mass_per_area1 (of type mass) was used in a mass_per_area definition (2)

Equation-2 : mass1=(a*-1+a+a-(a*-1+a+a+a*-1+a*-1))*(d(y1)*mass_per_area1)
formed by applying : strategy(mass_per_length,situation(line1))

This equation solves for mass1 but introduces [mass_per_area1].

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

So now I must solve for [mass_per_area1]
                  given [mass1,k,a,b,m]

I am now trying to solve for mass_per_area1 without introducing any unknowns.

Applicable formulae : [moment_of_inertia,mass_per_vol,mass_per_area,mass_per_length,resolve]
(try  moment_of_inertia)
(try  mass_per_vol)
(try  mass_per_area)
Trying to apply strategy(mass_per_area,situation(rect))

Equation-3 : m=(b-(-b))*(a-(-a))*mass_per_area1
formed by applying : strategy(mass_per_area,situation(rect))

This equation solves for mass_per_area1.

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

So now I must solve for []
```

```
                      given [mass_per_area1,mass1,k,a,b,m]


Equations extracted :
    m*k^2=integrate(mass1*((a*-1+a+a)/sqrt(3))^2,-b,b,y)
    mass1=(a*-1+a+a-(a*-1+a+a+a*-1+a*-1))*(d(y1)*mass_per_area1)
    m=(b-(-b))*(a-(-a))*mass_per_area1


yes
| ?- core      87552  (58368 lo-seg + 29184 hi-seg)
heap     53248 =  51256 in use +   1992 free
global    1187 =     16 in use +   1171 free
local     1024 =     16 in use +   1008 free
trail      511 =      0 in use +    511 free
    0.03 sec. for 1 GCs gaining 556 words
    0.24 sec. for 11 local shifts and 17 trail shifts
    4.52 sec. runtime
```

```
| ?- input(mofi3).

Problem from file : mofi3.prb

Radius of Gyration of an inclined line

 Let l1 be a new line
 Let l be a new point
 Let r be a new point
 Note: -a (of type length) was used in a separation definition (3)
 Note: [0,0] (of type angle) was used in a separation definition (4)
 Note: a (of type length) was used in a separation definition (3)
 Note: m (of type mass) was used in a mass definition (2)
 Let axis(rr) be a new line
 Let typical_point4 be a new typical_point
 Note: [alpha,0] (of type angle) was used in a incline definition (2)
 Note: k (of type rofg) was used in a rad_of_gyr definition (3)

mofi3 problem read into data base.


yes
| ?- go.

** ERROR Type unknown: alpha
   ( continue after error )

Attempting to solve for [k] in terms of [a,alpha,m]


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

 Applicable formulae : [parallel_axes,moment_of_inertia]
 (try  parallel_axes)
 (try  moment_of_inertia)
 Trying to apply strategy(moment_of_inertia,situation(l1,axis(rr)))
 Let point1 be a new point
 Note: r1 (of type length) was used in a separation definition (3)

 point1 is new fibre defined by
    isa(point,point1)
    separation(origin,point1,r1,[0,0])

 Note: x1/cos(alpha) (of type length) was used in a separation definition (3)
 Note: [alpha,0] (of type angle) was used in a separation definition (4)

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

 Applicable formulae : [parallel_axes,moment_of_inertia]
 (try  parallel_axes)
 (try  moment_of_inertia)
 Trying to apply strategy(moment_of_inertia,situation(l1,axis(rr)))
 Let mass_per_length1 be the mass_per_length of l1.
 Note: mass_per_length1 (of type mass) was used in a mass_per_length definition (2)

 Equation-1 : m*k^2=integrate(d(r1)*mass_per_length1*(r1*sin(alpha-0))^2,-a,a,r1)
 formed by applying : strategy(moment_of_inertia,situation(l1,axis(rr)))

 This equation solves for k but introduces [mass_per_length1].
```

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

 So now I must solve for [mass_per_length1]
                    given [k,a,alpha,m]

I am now trying to solve for mass_per_length1 without introducing any unknowns.

 Applicable formulae : [moment_of_inertia,mass_per_vol,mass_per_area,mass_per_length,resolve]
 (try  moment_of_inertia)
 (try  mass_per_vol)
 (try  mass_per_area)
 (try  mass_per_length)
 Trying to apply strategy(mass_per_length,situation(11))

 Equation-2 : m=(a-(-a))*mass_per_length1
 formed by applying : strategy(mass_per_length,situation(11))

 This equation solves for mass_per_length1.

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

 So now I must solve for []
                    given [mass_per_length1,k,a,alpha,m]


Equations extracted :
    m*k^2=integrate(d(r1)*mass_per_length1*(r1*sin(alpha-0))^2,-a,a,r1)
    m=(a-(-a))*mass_per_length1


yes
| ?- core     93184  (64000 lo-seg + 29184 hi-seg)
heap      58880 =  56798 in use +   2082 free
global     1187 =     16 in use +   1171 free
local      1024 =     16 in use +   1008 free
trail       511 =      0 in use +    511 free
    0.02 sec. for 1 GCs gaining 353 words
    0.14 sec. for 12 local shifts and 18 trail shifts
    7.89 sec. runtime
```

not Scanned

```
yes
| ?- input(mofi4).

Problem from file : mofi4.prb

Radius of Gyration of a parallelogram

 Let pgrm be a new parallelogram
 Let top be a new line
 Let bot be a new line
 Let left be a new line
 Let right be a new line
 Note: b (of type length) was used in a perp_dist definition (3)
 Note: [90,0] (of type angle) was used in a perp_dist definition (4)
 Note: -b (of type length) was used in a perp_dist definition (3)
 Note: -a (of type length) was used in a perp_dist definition (3)
 Note: [alpha-90,0] (of type angle) was used in a perp_dist definition (4)
 Note: a (of type length) was used in a perp_dist definition (3)
 Note: m (of type mass) was used in a mass definition (2)
 Let axis(rr) be a new line
 Let typical_point4 be a new typical_point
 Note: [alpha,0] (of type angle) was used in a incline definition (2)
 Note: k (of type rofg) was used in a rad_of_gyr definition (3)

mofi4 problem read into data base.


yes
| ?- go.

** ERROR Type unknown: alpha
   ( continue after error )

Attempting to solve for [k] in terms of [a,b,alpha,m]


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

 Applicable formulae : [parallel_axes,moment_of_inertia]
 (try  parallel_axes)
 (try  moment_of_inertia)
 Trying to apply strategy(moment_of_inertia,situation(pgrm,axis(rr)))
 Note: x1/cos(alpha) (of type length) was used in a separation definition (3)
 Note: [alpha,0] (of type angle) was used in a separation definition (4)
 Note: y1/sin(alpha) (of type length) was used in a separation definition (3)
 Let line1 be a new line
 Let point1 be a new point
 Let point2 be a new point
 Let origin2 be a new point
 Note: a*sin(alpha)^-1*-1+y1*tan(alpha)^-1+tan(alpha)*y1*-1 (of type length) was
 Note: [0,0] (of type angle) was used in a separation definition (4)
 Note: a*sin(alpha)^-1+y1*tan(alpha)^-1+tan(alpha)*y1*-1 (of type length) was us

 line1 is new fibre defined by
    isa(line,line1)
    line_sys(line1,point1,point2)
    isa(point,point1)
    isa(point,point2)
    isa(point,origin2)
```

```
    on(origin2,line1)
    separation(origin2,point1,a*sin(alpha)^-1*-1+y1*tan(alpha)^-1+tan(alpha)*y1*
    separation(origin2,point2,a*sin(alpha)^-1+y1*tan(alpha)^-1+tan(alpha)*y1*-1,

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

 Applicable formulae : [parallel_axes,moment_of_inertia]
 (try  parallel_axes)
 (try  moment_of_inertia)
 Trying to apply strategy(moment_of_inertia,situation(pasrm,axis(rr)))
 Let mass1 be the mass of line1.
 Note: mass1 (of type mass) was used in a mass definition (2)
 Let -(a*sin(alpha)^-1+y1*tan(alpha)^-1+tan(alpha)*y1*-1) be the separation of o
 Note: -(a*sin(alpha)^-1+y1*tan(alpha)^-1+tan(alpha)*y1*-1) (of type length) was

 Equation-1 : m*k^2=integrate(mass1*((a*sin(alpha)^-1+y1*tan(alpha)^-1+tan(alpha
 formed by applying : strategy(moment_of_inertia,situation(pasrm,axis(rr)))

 This equation solves for k but introduces [mass1].

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

 So now I must solve for [mass1]
                  given [k,a,b,alpha,m]

I am now trying to solve for mass1 without introducing any unknowns.

 Applicable formulae : [moment_of_inertia,mass_per_vol,mass_per_area,mass_per_len
 (try  moment_of_inertia)
 (try  mass_per_vol)
 (try  mass_per_area)
 (try  mass_per_length)
 Trying to apply strategy(mass_per_length,situation(line1))
 (try  resolve)

No luck - I will now accept unknowns in solving for mass1.

 Applicable formulae : [moment_of_inertia,mass_per_vol,mass_per_area,mass_per_len
 (try  moment_of_inertia)
 (try  mass_per_vol)
 (try  mass_per_area)
 (try  mass_per_length)
 Trying to apply strategy(mass_per_length,situation(line1))
 Let mass_per_area1 be the mass_per_area of pasrm.
 Note: mass_per_area1 (of type mass) was used in a mass_per_area definition (2)

 Equation-2 : mass1=(a*sin(alpha)^-1+y1*tan(alpha)^-1+tan(alpha)*y1*-1- -(a*sin(a
*mass_per_area1)
 formed by applying : strategy(mass_per_length,situation(line1))

 This equation solves for mass1 but introduces [mass_per_area1].

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

 So now I must solve for [mass_per_area1]
                  given [mass1,k,a,b,alpha,m]

I am now trying to solve for mass_per_area1 without introducing any unknowns.
```

```
Applicable formulae : [moment_of_inertia,mass_per_vol,mass_per_area,mass_per_len
(try  moment_of_inertia)
(try  mass_per_vol)
(try  mass_per_area)
Trying to apply strategy(mass_per_area,situation(pagrm))

Equation-3 : m=(b-(-b))*(a/sin(alpha)-(-a)/sin(alpha))*mass_per_area1
formed by applying : strategy(mass_per_area,situation(pagrm))

This equation solves for mass_per_area1.

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

So now I must solve for []
                  given [mass_per_area1,mass1,k,a,b,alpha,m]


Equations extracted :
    m*k^2=integrate(mass1*((a*sin(alpha)^-1+y1*tan(alpha)^-1+tan(alpha)*y1*-1)*s
    mass1=(a*sin(alpha)^-1+y1*tan(alpha)^-1+tan(alpha)*y1*-1- -(a*sin(alpha)^-1+y
area1)
    m=(b-(-b))*(a/sin(alpha)-(-a)/sin(alpha))*mass_per_area1


yes
| ?- core     93696  (64512 lo-seg + 29184 hi-seg)
heap      59392 =  57587 in use +   1805 free
global     1187 =     16 in use +   1171 free
local      1024 =     16 in use +   1008 free
trail       511 =      0 in use +    511 free
    0.02 sec. for 1 GCs gaining 353 words
    0.33 sec. for 16 local shifts and 28 trail shifts
   10.90 sec. runtime
```

```
yes
| ?- input(mofi5).

Problem from file : mofi5.prb

Radius of Gyration of a ring

 Let ring1 be a new ring
 Note: a (of type length) was used in a radius definition (2)
 Note: m (of type mass) was used in a mass definition (2)
 Note: k (of type rofg) was used in a rad_of_gyr definition (3)

mofi5 problem read into data base.


yes
| ?- go.

Attempting to solve for [k] in terms of [a,m]


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

  Applicable formulae : [parallel_axes,moment_of_inertia]
  (try  parallel_axes)
  (try  moment_of_inertia)
  Trying to apply strategy(moment_of_inertia,situation(ring1,axis(z)))
  Let point1 be a new point

  point1 is new fibre defined by
      isa(point,point1)
      separation(origin,point1,a,[theta1,0])


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

  Applicable formulae : [parallel_axes,moment_of_inertia]
  (try  parallel_axes)
  (try  moment_of_inertia)
  Trying to apply strategy(moment_of_inertia,situation(ring1,axis(z)))
  Let mass_per_length1 be the mass_per_length of ring1.
  Note: mass_per_length1 (of type mass) was used in a mass_per_length definition (2)

  Equation-1 : m*k^2=integrate(d(theta1)*mass_per_length1*((a^2)^number(+,[1],[2]))^2,0,2*pi,z)
  formed by applying : strategy(moment_of_inertia,situation(ring1,axis(z)))

  This equation solves for k but introduces [mass_per_length1,+].

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

 ** ERROR Type unknown -- [+]
    ( continue after error )

  So now I must solve for [mass_per_length1,+]
                     given [k,a,m]

 I am now trying to solve for mass_per_length1 without introducing any unknowns.

  Applicable formulae : [moment_of_inertia,mass_per_vol,mass_per_area,mass_per_length,resolve]
```

```
(try  moment_of_inertia)
(try  mass_per_vol)
(try  mass_per_area)
(try  mass_per_length)
Trying to apply strategy(mass_per_length,situation(ring1))

Equation-2 : m=2*pi*a*mass_per_length1
formed by applying : strategy(mass_per_length,situation(ring1))

 This equation solves for mass_per_length1.

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

 So now I must solve for [t]
                  given [mass_per_length1,k,a,m]

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

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

I am unable to solve for t.

I will go back to solve for k again

 Equation-1 rejected.


I am unable to solve for k.

no
| ?- core     83456  (54272 lo-seg + 29184 hi-seg)
heap     49152 =  48078 in use +   1074 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 353 words
    0.22 sec. for 18 local shifts and 21 trail shifts
    4.90 sec. runtime
```

```
yes
| ?- input(mofi6).

Problem from file : mofi6.prb

Radius of Gyration of a Disc

 Let disc1 be a new disc
 Note: a (of type length) was used in a radius definition (2)
 Note: m (of type mass) was used in a mass definition (2)
 Note: k (of type rofg) was used in a rad_of_gyr definition (3)

mofi6 problem read into data base.


yes
| ?- go.

Attempting to solve for [k] in terms of [a,m]


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

 Applicable formulae : [parallel_axes,moment_of_inertia]
 (try  parallel_axes)
 (try  moment_of_inertia)
 Trying to apply strategy(moment_of_inertia,situation(disc1,axis(z)))
 Let ring1 be a new ring
 Note: r1 (of type length) was used in a radius definition (2)

 ring1 is new fibre defined by
    isa(ring,ring1)
    centre(ring1,origin)
    radius(ring1,r1)
    meets(axis(z),ring1,origin)


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

 Applicable formulae : [parallel_axes,moment_of_inertia]
 (try  parallel_axes)
 (try  moment_of_inertia)
 Trying to apply strategy(moment_of_inertia,situation(disc1,axis(z)))
 Let mass1 be the mass of ring1.
 Note: mass1 (of type mass) was used in a mass definition (2)

 Equation-1 : m*k^2=integrate(mass1*r1^2,0,a,z)
 formed by applying : strategy(moment_of_inertia,situation(disc1,axis(z)))

 This equation solves for k but introduces [mass1].

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

 So now I must solve for [mass1]
                  given [k,a,m]

I am now trying to solve for mass1 without introducing any unknowns.

 Applicable formulae : [moment_of_inertia,mass_per_vol,mass_per_area,mass_per_length,resolve]
```

```
   (try  moment_of_inertia)
   (try  mass_per_vol)
   (try  mass_per_area)
   (try  mass_per_length)
   Trying to apply strategy(mass_per_length,situation(ring1))
   (try  resolve)

 No luck - I will now accept unknowns in solving for mass1.

  Applicable formulae : [moment_of_inertia,mass_per_vol,mass_per_area,mass_per_length,resolve]
   (try  moment_of_inertia)
   (try  mass_per_vol)
   (try  mass_per_area)
   (try  mass_per_length)
   Trying to apply strategy(mass_per_length,situation(ring1))
   Let mass_per_area1 be the mass_per_area of disc1.
   Note: mass_per_area1 (of type mass) was used in a mass_per_area definition (2)
   Let line_sys1 be the line_sys of ring1 Let line_sys2 be the line_sys of ring1.

   Equation-2 : mass1=2*pi*r1*(d(r1)*mass_per_area1)
   formed by applying : strategy(mass_per_length,situation(ring1))

   This equation solves for mass1 but introduces [mass_per_area1].

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

   So now I must solve for [mass_per_area1]
                  given [mass1,k,a,m]

 I am now trying to solve for mass_per_area1 without introducing any unknowns.

  Applicable formulae : [moment_of_inertia,mass_per_vol,mass_per_area,mass_per_length,resolve]
   (try  moment_of_inertia)
   (try  mass_per_vol)
   (try  mass_per_area)
   Trying to apply strategy(mass_per_area,situation(disc1))

   Equation-3 : m=pi*a^2*mass_per_area1
   formed by applying : strategy(mass_per_area,situation(disc1))

   This equation solves for mass_per_area1.

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

   So now I must solve for []
                  given [mass_per_area1,mass1,k,a,m]

 Equations extracted :
     m*k^2=integrate(mass1*r1^2,0,a,z)
     mass1=2*pi*r1*(d(r1)*mass_per_area1)
     m=pi*a^2*mass_per_area1

yes
| ?- core      83456  (54272 lo-seg + 29184 hi-seg)
heap      49152 =  48080 in use +   1072 free
global     1187 =     16 in use +   1171 free
local      1024 =     16 in use +   1008 free
trail       511 =      0 in use +    511 free
```

```
| ?- input(mofi7).

Problem from file : mofi7.prb

Radius of Gyration of a sphere

 Let sphere1 be a new sphere
 Note: a (of type length) was used in a radius definition (2)
 Note: m (of type mass) was used in a mass definition (2)
 Note: k (of type rofg) was used in a rad_of_gyr definition (3)

mofi7 problem read into data base.


yes
| ?- go.

Attempting to solve for [k] in terms of [a,m]


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

 Applicable formulae : [parallel_axes,moment_of_inertia]
 (try  parallel_axes)
 (try  moment_of_inertia)
 Trying to apply strategy(moment_of_inertia,situation(sphere1,axis(z)))
 Let disc1 be a new disc
 Note: sqrt(a^2-z1^2) (of type length) was used in a radius definition (2)

 disc1 is new fibre defined by
    isa(disc,disc1)
    centre(disc1,typ_pt1)
    radius(disc1,sqrt(a^2-z1^2))
    meets(axis(z),disc1,typ_pt1)


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

 Applicable formulae : [parallel_axes,moment_of_inertia]
 (try  parallel_axes)
 (try  moment_of_inertia)
 Trying to apply strategy(moment_of_inertia,situation(sphere1,axis(z)))
 Let mass1 be the mass of disc1.
 Note: mass1 (of type mass) was used in a mass definition (2)

 Equation-1 : m*k^2=integrate(mass1*(sqrt(a^2-z1^2)/sqrt(2))^2,-a,a,z)
 formed by applying : strategy(moment_of_inertia,situation(sphere1,axis(z)))

 This equation solves for k but introduces [mass1].

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

 So now I must solve for [mass1]
                  given [k,a,m]

I am now trying to solve for mass1 without introducing any unknowns.

 Applicable formulae : [moment_of_inertia,mass_per_vol,mass_per_area,mass_per_length,resolve]
 (try  moment_of_inertia)
 (try  mass_per_vol)
```

```
(try  mass_per_area)
Trying to apply strategy(mass_per_area,situation(disc1))
(try  mass_per_length)
(try  resolve)

No luck - I will now accept unknowns in solving for mass1.

 Applicable formulae : [moment_of_inertia,mass_per_vol,mass_per_area,mass_per_length,resolve]
 (try  moment_of_inertia)
 (try  mass_per_vol)
 (try  mass_per_area)
 Trying to apply strategy(mass_per_area,situation(disc1))
 Let mass_per_vol1 be the mass_per_vol of sphere1.
 Note: mass_per_vol1 (of type mass) was used in a mass_per_vol definition (2)

 Equation-2 : mass1=pi*sqrt(a^2-z1^2)^2*(d(z1)*mass_per_vol1)
 formed by applying : strategy(mass_per_area,situation(disc1))

 This equation solves for mass1 but introduces [mass_per_vol1].

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

 So now I must solve for [mass_per_vol1]
                   given [mass1,k,a,m]

I am now trying to solve for mass_per_vol1 without introducing any unknowns.

 Applicable formulae : [moment_of_inertia,mass_per_vol,mass_per_area,mass_per_length,resolve]
 (try  moment_of_inertia)
 (try  mass_per_vol)
 Trying to apply strategy(mass_per_vol,situation(sphere1))

 Equation-3 : m=4/3*pi*a^3*mass_per_vol1
 formed by applying : strategy(mass_per_vol,situation(sphere1))

 This equation solves for mass_per_vol1.

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

 So now I must solve for []
                   given [mass_per_vol1,mass1,k,a,m]


Equations extracted :
    m*k^2=integrate(mass1*(sqrt(a^2-z1^2)/sqrt(2))^2,-a,a,z)
    mass1=pi*sqrt(a^2-z1^2)^2*(d(z1)*mass_per_vol1)
    m=4/3*pi*a^3*mass_per_vol1


yes
| ?- core      83968  (54784 lo-seg + 29184 hi-seg)
heap      49664 =   48440 in use +   1224 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 312 words
    0.17 sec. for 10 local shifts and 15 trail shifts
    3.40 sec. runtime
```

```
 *START* User BUNDY    HPS [400,405]  Job  MOFI8 Seq. 8102 Date 02-Apr-81 12:13:56 Monitor ERCC ICF DEC10 7.01(043)  *START*
File: DSKA:MOFI8.SOL<005>[400,405,MYMOFI] Created: 02-Apr-81 12:13:16 Printed: 02-Apr-81 12:14:54
QUEUE Switches:  /FILE:ASCII /COPIES:1 /SPACING:1 /LIMIT:105 /FORMS:NORMAL
```

```
| ?- input(mofi8).

Problem from file : mofi8.prb

Radius of Gyration of a shell

 Let shell1 be a new shell
 Note: a (of type length) was used in a radius definition (2)
 Note: m (of type mass) was used in a mass definition (2)
 Note: k (of type rofg) was used in a rad_of_gyr definition (3)

mofi8 problem read into data base.


yes
| ?- go.

Attempting to solve for [k] in terms of [a,m]


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

  Applicable formulae : [parallel_axes,moment_of_inertia]
  (try  parallel_axes)
  (try  moment_of_inertia)
  Trying to apply strategy(moment_of_inertia,situation(shell1,axis(z)))
  Let ring1 be a new ring
  Note: sqrt(a^2-z1^2) (of type length) was used in a radius definition (2)

  ring1 is new fibre defined by
      isa(ring,ring1)
      centre(ring1,origin1)
      radius(ring1,sqrt(a^2-z1^2))
      meets(axis(z),ring1,origin1)


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

  Applicable formulae : [parallel_axes,moment_of_inertia]
  (try  parallel_axes)
  (try  moment_of_inertia)
  Trying to apply strategy(moment_of_inertia,situation(shell1,axis(z)))
  Let mass1 be the mass of ring1.
```

```
 Note: mass1 (of type mass) was used in a mass definition (2)

 Equation-1 : m*k^2=integrate(mass1*sqrt(a^2-z1^2)^2,-a,a,z)
 formed by applying : strategy(moment_of_inertia,situation(shell1,axis(z)))

 This equation solves for k but introduces [mass1].

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

 So now I must solve for [mass1]
                  given [k,a,m]

I am now trying to solve for mass1 without introducing any unknowns.

 Applicable formulae : [moment_of_inertia,mass_per_vol,mass_per_area,mass_per_length,resolve]
 (try  moment_of_inertia)
 (try  mass_per_vol)






 (try  mass_per_area)
 (try  mass_per_length)
 Trying to apply strategy(mass_per_length,situation(ring1))
 (try  resolve)

No luck - I will now accept unknowns in solving for mass1.

 Applicable formulae : [moment_of_inertia,mass_per_vol,mass_per_area,mass_per_length,resolve]
 (try  moment_of_inertia)
 (try  mass_per_vol)
 (try  mass_per_area)
 (try  mass_per_length)
 Trying to apply strategy(mass_per_length,situation(ring1))
 Let mass_per_area1 be the mass_per_area of shell1.
 Note: mass_per_area1 (of type mass) was used in a mass_per_area definition (2)
 Let line_sys1 be the line_sys of ring1 Let line_sys2 be the line_sys of ring1.

 Equation-2 : mass1=2*pi*sqrt(a^2-z1^2)*(d(z1)*mass_per_area1)
 formed by applying : strategy(mass_per_length,situation(ring1))

 This equation solves for mass1 but introduces [mass_per_area1].

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

 So now I must solve for [mass_per_area1]
                  given [mass1,k,a,m]

I am now trying to solve for mass_per_area1 without introducing any unknowns.

 Applicable formulae : [moment_of_inertia,mass_per_vol,mass_per_area,mass_per_length,resolve]
 (try  moment_of_inertia)
 (try  mass_per_vol)
 (try  mass_per_area)
 Trying to apply strategy(mass_per_area,situation(shell1))

 Equation-3 : m=4*pi*a^2*mass_per_area1
 formed by applying : strategy(mass_per_area,situation(shell1))

 This equation solves for mass_per_area1.

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

 So now I must solve for []
```

```
                    siven [mass_per_area1,mass1,k,a,m]


Equations extracted :
    m*k^2=integrate(mass1*sqrt(a^2-z1^2)^2,-a,a,z)
    mass1=2*pi*sqrt(a^2-z1^2)*(d(z1)*mass_per_area1)
    m=4*pi*a^2*mass_per_area1


yes
| ?- core      87552  (58368 lo-seg + 29184 hi-seg)
heap      53248 =  51358 in use +    1890 free
global     1187 =     16 in use +    1171 free
local      1024 =     16 in use +    1008 free
trail       511 =      0 in use +     511 free
    0.02 sec. for 1 GCs gaining 353 words
    0.20 sec. for 13 local shifts and 19 trail shifts




    6.07 sec. runtime
```

```
*START* User BUNDY    HPS [400,405]  Job  MOFI9 Seq. 8112 Date 02-Apr-81 13:29:38
File: DSKA:MOFI9.SOL<005>[400,405,MYMOFI] Created: 02-Apr-81 13:26:28 Printed: 02-A
QUEUE Switches:  /FILE:ASCII /COPIES:1 /SPACING:1 /LIMIT:310 /FORMS:NORMAL
```

```
yes
| ?- input(mofi9).

Problem from file : mofi9.prb

Radius of Gyration of a cylinder

 Let cylinder1 be a new cylinder
 Note: a (of type length) was used in a radius definition (2)

** ERROR reckind    ( continue after error )
 Note: m (of type mass) was used in a mass definition (2)
 Note: k (of type rofg) was used in a rad_of_gyr definition (3)

mofi9 problem read into data base.


yes
| ?- go.

** ERROR Type unknown    ( continue after error )

Attempting to solve for [k] in terms of [a,h,m]


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

 Applicable formulae : [parallel_axes,moment_of_inertia]
 (try  parallel_axes)
 (try  moment_of_inertia)
 Trying to apply strategy(moment_of_inertia,situation(cylinder1,axis(z)))
 Let disc1 be a new disc

 disc1 is new fibre defined by
     isa(disc,disc1)
     centre(disc1,origin1)
     radius(disc1,a)
     meets(axis(z),disc1,origin1)

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

```
 Applicable formulae : [parallel_axes,moment_of_inertia]
 (try  parallel_axes)
 (try  moment_of_inertia)
 Trying to apply strategy(moment_of_inertia,situation(cylinder1,axis(z)))
 Let mass1 be the mass of disc1.
 Note: mass1 (of type mass) was used in a mass definition (2)

 Equation-1 : m*k^2=integrate(mass1*(a/sqrt(2))^2,0,h,z)
 formed by applying : strategy(moment_of_inertia,situation(cylinder1,axis(z)))

 This equation solves for k but introduces [mass1].

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

 So now I must solve for [mass1]
                   given [k,a,h,m]




I am now trying to solve for mass1 without introducing any unknowns.

 Applicable formulae : [moment_of_inertia,mass_per_vol,mass_per_area,mass_per_lengt
 (try  moment_of_inertia)
 (try  mass_per_vol)
 (try  mass_per_area)
 Trying to apply strategy(mass_per_area,situation(disc1))
 (try  mass_per_length)
 (try  resolve)

No luck - I will now accept unknowns in solving for mass1.

 Applicable formulae : [moment_of_inertia,mass_per_vol,mass_per_area,mass_per_lengt
 (try  moment_of_inertia)
 (try  mass_per_vol)
 (try  mass_per_area)
 Trying to apply strategy(mass_per_area,situation(disc1))
 Let mass_per_vol1 be the mass_per_vol of cylinder1.
 Note: mass_per_vol1 (of type mass) was used in a mass_per_vol definition (2)

 Equation-2 : mass1=pi*a^2*(d(z1)*mass_per_vol1)
 formed by applying : strategy(mass_per_area,situation(disc1))

 This equation solves for mass1 but introduces [mass_per_vol1].

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

 So now I must solve for [mass_per_vol1]
                   given [mass1,k,a,h,m]

I am now trying to solve for mass_per_vol1 without introducing any unknowns.

 Applicable formulae : [moment_of_inertia,mass_per_vol,mass_per_area,mass_per_lengt
 (try  moment_of_inertia)
 (try  mass_per_vol)
 Trying to apply strategy(mass_per_vol,situation(cylinder1))

 Equation-3 : m=pi*a^2*h*mass_per_vol1
 formed by applying : strategy(mass_per_vol,situation(cylinder1))

 This equation solves for mass_per_vol1.

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

```
 So now I must solve for []
                     given [mass_per_vol1,mass1,k,a,h,m]


Equations extracted :
    m*k^2=integrate(mass1*(a/sqrt(2))^2,0,h,z)
    mass1=pi*a^2*(d(z1)*mass_per_vol1)
    m=pi*a^2*h*mass_per_vol1


yes
| ?- core      87552  (58368 lo-seg + 29184 hi-seg)
heap      53248 =  51312 in use +   1936 free
global     1187 =     16 in use +   1171 free
local      1024 =     16 in use +   1008 free
trail       511 =      0 in use +    511 free



    0.04 sec. for 1 GCs gaining 523 words
    0.18 sec. for 8 local shifts and 14 trail shifts
    3.28 sec. runtime
```

```
 *START* User BUNDY    HPS [400,405]  Job  MOFI9 Seq. 8112 Date 02-Apr-81 13:29:38 Monitor ERCC ICF DEC10 7.01(043)   *START*
File: DSKA:MOFI10.SOL<005>[400,405,MYMOFI] Created: 02-Apr-81 13:28:55 Printed: 02-Apr-81 13:33:19
QUEUE Switches:  /FILE:ASCII /COPIES:1 /SPACING:1 /LIMIT:310 /FORMS:NORMAL
```

```
Mecho Problem Solver
Prolog-10  version 3.2

| ?- restore(save).

yes
| ?- input(mofi10).

Problem from file : mofi10.prb

Radius of Gyration of a cone

 Let cone1 be a new cone
 Note: a (of type length) was used in a radius definition (2)

** ERROR reckind   ( continue after error )
 Note: m (of type mass) was used in a mass definition (2)
 Note: k (of type rofg) was used in a rad_of_gyr definition (3)

mofi10 problem read into data base.


yes
| ?- go.

** ERROR Type unknown   ( continue after error )

Attempting to solve for [k] in terms of [a,h,m]


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

 Applicable formulae : [parallel_axes,moment_of_inertia]
 (try  parallel_axes)
 (try  moment_of_inertia)
 Trying to apply strategy(moment_of_inertia,situation(cone1,axis(z)))
 Let disc1 be a new disc
 Note: a*(h-z1)/h (of type length) was used in a radius definition (2)

 disc1 is new fibre defined by
    isa(disc,disc1)
    centre(disc1,origin1)
    radius(disc1,a*(h-z1)/h)
```

```
   meets(axis(z),disc1,origin1)


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

 Applicable formulae : [parallel_axes,moment_of_inertia]
 (try  parallel_axes)
 (try  moment_of_inertia)
 Trying to apply strategy(moment_of_inertia,situation(cone1,axis(z)))
 Let mass1 be the mass of disc1.
 Note: mass1 (of type mass) was used in a mass definition (2)

 Equation-1 : m*k^2=integrate(mass1*(a*(h-z1)/h/sqrt(2))^2,0,h,z)
 formed by applying : strategy(moment_of_inertia,situation(cone1,axis(z)))

 This equation solves for k but introduces [mass1].




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

 So now I must solve for [mass1]
                given [k,a,h,m]

I am now trying to solve for mass1 without introducing any unknowns.

 Applicable formulae : [moment_of_inertia,mass_per_vol,mass_per_area,mass_per_length,resolve]
 (try  moment_of_inertia)
 (try  mass_per_vol)
 (try  mass_per_area)
 Trying to apply strategy(mass_per_area,situation(disc1))
 (try  mass_per_length)
 (try  resolve)

No luck - I will now accept unknowns in solving for mass1.

 Applicable formulae : [moment_of_inertia,mass_per_vol,mass_per_area,mass_per_length,resolve]
 (try  moment_of_inertia)
 (try  mass_per_vol)
 (try  mass_per_area)
 Trying to apply strategy(mass_per_area,situation(disc1))
 Let mass_per_vol1 be the mass_per_vol of cone1.
 Note: mass_per_vol1 (of type mass) was used in a mass_per_vol definition (2)

 Equation-2 : mass1=pi*(a*(h-z1)/h)^2*(d(z1)*mass_per_vol1)
 formed by applying : strategy(mass_per_area,situation(disc1))

 This equation solves for mass1 but introduces [mass_per_vol1].

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

 So now I must solve for [mass_per_vol1]
                given [mass1,k,a,h,m]

I am now trying to solve for mass_per_vol1 without introducing any unknowns.

 Applicable formulae : [moment_of_inertia,mass_per_vol,mass_per_area,mass_per_length,resolve]
 (try  moment_of_inertia)
 (try  mass_per_vol)
 Trying to apply strategy(mass_per_vol,situation(cone1))

 Equation-3 : m=1/3*pi*a^2*h*mass_per_vol1
```

```
formed by applying : strategy(mass_per_vol,situation(cone1))

This equation solves for mass_per_vol1.

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

So now I must solve for []
                 given [mass_per_vol1,mass1,k,a,h,m]


Equations extracted :
    m*k^2=integrate(mass1*(a*(h-z1)/h/sqrt(2))^2,0,h,z)
    mass1=pi*(a*(h-z1)/h)^2*(d(z1)*mass_per_vol1)
    m=1/3*pi*a^2*h*mass_per_vol1


yes



| ?- core     87552  (58368 lo-seg + 29184 hi-seg)
heap     53248 =  51370 in use +    1878 free
global    1187 =     16 in use +    1171 free
local     1024 =     16 in use +    1008 free
trail      511 =      0 in use +     511 free
    0.05 sec. for 1 GCs gaining 523 words
    0.16 sec. for 9 local shifts and 15 trail shifts
    3.37 sec. runtime
```

```
| ?- input(mofi11).

Problem from file : mofi11.prb

Radius of Gyration of a tube

 Let tube1 be a new tube
 Note: a (of type length) was used in a radius definition (2)

** ERROR Cannot record kind for height(tube1,h)
    ( continue after error )
 Note: m (of type mass) was used in a mass definition (2)
 Note: k (of type rofg) was used in a rad_of_gyr definition (3)

mofi11 problem read into data base.


yes
| ?- go.

** ERROR Type unknown: h
    ( continue after error )

Attempting to solve for [k] in terms of [a,h,m]


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

  Applicable formulae : [parallel_axes,moment_of_inertia]
  (try  parallel_axes)
  (try  moment_of_inertia)
  Trying to apply strategy(moment_of_inertia,situation(tube1,axis(z)))
   origin and origin1 are seperated by [z1,0,90] during always.
  Let ring1 be a new ring

  ring1 is new fibre defined by
      isa(ring,ring1)
      centre(ring1,origin1)
      radius(ring1,a)
      meets(axis(z),ring1,origin1)


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

  Applicable formulae : [parallel_axes,moment_of_inertia]
  (try  parallel_axes)
  (try  moment_of_inertia)
  Trying to apply strategy(moment_of_inertia,situation(tube1,axis(z)))
  Let mass1 be the mass of ring1.
  Note: mass1 (of type mass) was used in a mass definition (2)

  Equation-1 : m*k^2=integrate(mass1*a^2,0,h,z1)
  formed by applying : strategy(moment_of_inertia,situation(tube1,axis(z)))

  This equation solves for k but introduces [mass1].

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

  So now I must solve for [mass1]
                    given [k,a,h,m]
```

```
I am now trying to solve for mass1 without introducing any unknowns.

 Applicable formulae : [moment_of_inertia,mass_per_vol,mass_per_area,mass_per_length,resolve]
 (try  moment_of_inertia)
 (try  mass_per_vol)
 (try  mass_per_area)
 (try  mass_per_length)
 Trying to apply strategy(mass_per_length,situation(ring1))
 (try  resolve)

No luck - I will now accept unknowns in solving for mass1.

 Applicable formulae : [moment_of_inertia,mass_per_vol,mass_per_area,mass_per_length,resolve]
 (try  moment_of_inertia)
 (try  mass_per_vol)
 (try  mass_per_area)
 (try  mass_per_length)
 Trying to apply strategy(mass_per_length,situation(ring1))
 Let mass_per_area1 be the mass_per_area of tube1.
 Note: mass_per_area1 (of type mass) was used in a mass_per_area definition (2)
 Let line_sys1 be the line_sys of ring1 Let line_sys2 be the line_sys of ring1.

 Equation-2 : mass1=2*pi*a*(d(z1)*mass_per_area1)
 formed by applying : strategy(mass_per_length,situation(ring1))

 This equation solves for mass1 but introduces [mass_per_area1].

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

 So now I must solve for [mass_per_area1]
                   given [mass1,k,a,h,m]

 I am now trying to solve for mass_per_area1 without introducing any unknowns.

  Applicable formulae : [moment_of_inertia,mass_per_vol,mass_per_area,mass_per_length,resolve]
  (try  moment_of_inertia)
  (try  mass_per_vol)
  (try  mass_per_area)
  Trying to apply strategy(mass_per_area,situation(tube1))

  Equation-3 : m=2*pi*a*h*mass_per_area1
  formed by applying : strategy(mass_per_area,situation(tube1))

  This equation solves for mass_per_area1.

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

  So now I must solve for []
                   given [mass_per_area1,mass1,k,a,h,m]


 Equations extracted :
     m*k^2=integrate(mass1*a^2,0,h,z1)
     mass1=2*pi*a*(d(z1)*mass_per_area1)
     m=2*pi*a*h*mass_per_area1


 yes
 | ?- core     88064  (58880 lo-seg + 29184 hi-seg)
```

```
heap      53760 =  51520 in use +   2240 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 353 words
    0.19 sec. for 12 local shifts and 20 trail shifts
    6.13 sec. runtime
```

```
/* MOFI : All the bits for the Moment of inertia problems */

:- [ ops2,                  % Operator declarations
     load,                  % Routines for loading rules
     sel,                   % Utilities for conjunctions

     defn,                  % Equational definition generation
     find,                  % Finding fibres for bodies

     magic                  % Fibre Schema
   ],


   load([ mk1,              % Meta level knowledge
          typ1,             % Type hierarchy
          inf1,             % Inference rules (length, area)
          inf2,             % Inference rules (mass of fibres)
          form1,            % Formulae (mass)
          form2,            % Formulae (rofs)

          cough             % Current hacks
        ]).
```

Older code

```
/* Operator declarations for Moments of Inertia problems */

   :- op(1160,xfx,[ <==, <--, -->, -> ]).
   :- op(900,fy,add).
   :- op(850,xfy,&).
   :- op(700,xfy,=<).
```

```
/* FORM1 : Formulae for calculating masses of bodies */


                              % Mass per length

relates(mass_per_length,[mass,length]).



prepare(mass_per_length,Q,mass,mass(Obj,M,Per),
                situation(Obj,Per) ).

prepare(mass_per_length,Q,mass,mass_per_length(Obj,Mu,Per),
                situation(Obj,Per) ).



isform(mass_per_length, situation(Obj,Per),
                M = L*Mu )
    <-- mass(Obj,M,Per) &
        mass_per_length(Obj,Mu,Per) &
        length(Obj,L,Per).



                              % Mass per area

relates(mass_per_area,[mass,area]).



prepare(mass_per_area,Q,mass,mass(Obj,M,Per),
                situation(Obj,Per) ).

prepare(mass_per_area,Q,mass,mass_per_area(Obj,Mu,Per),
                situation(Obj,Per) ).



isform(mass_per_area, situation(Obj,Per),
                M = A*Mu )
    <-- mass(Obj,M,Per) &
        mass_per_area(Obj,Mu,Per) &
        area(Obj,A,Per).
```

```
/* FORM2 : Formulae et al for the Moment of inertia problems */


{ problem_solving_rules }.
%-------------------------------


                        % Moment of Inertia

relates(moment_of_inertia,[mass,rofg]).


prepare(moment_of_inertia,Q,rofg,rad_of_gyr(Obj,Axis,RG,Per),
                situation(Obj,Axis,Fibre,Per))
    <-- meets(Axis,Obj,Origin) &
        centre(Obj,Origin) &
        regular_fibre(Obj,Fibre).


isform(moment_of_inertia, situation(Obj,Axis,Fibre,Per),
                M*RG^2 = integrate(Mf*RGf^2, A,B,X))
    <-- mass(Obj,M,Per) &
        rad_of_gyr(Obj,Axis,RG,Per) &
        mass(Fibre,Mf,Per) &
        rad_of_gyr(Fibre,Axis,RGf,Per) &
        cont_meas(Obj,X,_,Fibre,A,B).


                        % Parallel Axes

relates(parallel_axes,[rofg]).


prepare(parallel_axes,Q,rofg,rad_of_gyr(Obj,Axis,RG,Per),
                situation(Obj,Axis,Newaxis,Per))
    <-- meets(Axis,Obj,Axpt) &
        centre(Obj,Origin) &
        { diff(Axpt,Origin) } &
        meets(Newaxis,Obj,Origin) &                       || ?
        parallel(Axis,Newaxis).


isform(parallel_axes, situation(Obj,RealAxis,CentreAxis,Per),
                RG = RGC + A^2 )
    <-- rad_of_gyr(Obj,RealAxis,RG,Per) &
        rad_of_gyr(Obj,CentreAxis,RGC,Per) &
        perp_dist(RealAxis,CentreAxis,A,Per).
```

```
/* INF1 : Inference rules concerning the lengths, areas etc. of
                 various kinds of body

*/



                        % The length of a line is twice its radius

length(Line,2*R,Per)
    <-- line(Line) &
        radius(Line,R).


                        % The length of a ring based on its radius

length(Ring,2*pi*R,Per)
    <-- ring(Ring) &
        radius(Ring,R).


                        % Hack rule for the area of a 1D fibre

area(Fibre,L*d(X),Per)
    <-- body1d(Fibre) &
        length(Fibre,L,Per) &
        cont_meas(Body,X,Origin,Fibre,A,B).


                        % The area of a square

area(Square,(2*A)^2,Per)
    <-- square(Square) &
        radius(Square,A).


                        % The area of a disc

area(Disc,pi*R^2,Per)
    <-- disc(Disc) &
        radius(Disc,R).
```

```
/* INF2 : Inference rules for masses of fibres */


                              % The mass per length of a fibre is that
                              %  of its supporting body.

mass_per_length(Fibre,Mu,Per)
    <-- % fibre(Fibre) &
        cont_meas(Obj,X,Origin,Fibre,A,B) &
        body1d(Obj) &
        mass_per_length(Obj,Mu,Per).



                              % The mass per area of a fibre is that
                              %  of its supporting body.

mass_per_area(Fibre,Mu,Per)
    <-- % fibre(Fibre) &
        cont_meas(Obj,X,Origin,Fibre,A,B) &
        body2d(Obj) &
        mass_per_area(Obj,Mu,Per).
```

```
/* COUGH : Current hacks */


                                    % R of G of a ring.
                                    %  Need more info about the shape
                                    %  of a fibre - ie its centre and
                                    %  radius - to do this properly.

rad_of_gyr(Ring,Axis,R^2,Per)
     <-- ring(Ring) &
          radius(Ring,R).



                                    % This rule works for a ring fibre

radius(Ring,X)
     <-- ring(Ring) &
          cont_meas(Obj,X,Origin,Ring,A,B).



                                    % cont_meas Meta knowledge

argstruct(cont_meas,6,
                    [foo,foo,foo,foo,foo,foo],
                    [arg,arg,arg,arg,arg,arg]).
```

```
/* MK1 : Meta level knowledge for the Moment of inertia problems */


{ meta_knowledge }.
%----------------------


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

argstruct(basetype,2,
                [type,object],
                [arg,arg]).

argstruct(centre,2,
                [object,point],
                [arg,val]).

argstruct(dist_along,4,
                [object,point,point,length],
                [arg,arg,arg,arg]).

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

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

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

argstruct(meets,3,
                [axis,object,point],
                [arg,arg,arg]).

argstruct(radius,2,
                [object,length],
                [arg,val]).

argstruct(rad_of_gyr,4,
                [object,axis,rofg,time],
                [arg,arg,val,arg]).

argstruct(range,3,
                [fibre,length,length],
                [arg,arg,arg]).

argstruct(regular_fibre,2,
                [object,fibre],
                [arg,arg]).

argstruct(perp_dist,4,
                [object,object,length,time],
                [arg,arg,val,arg]).
```

```
/* TYP1 : Part of type hierarchy for Moments of Inertia problems */


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


  body0d -> object.
  body1d -> object.
  body2d -> object.
  body3d -> object.

  point -> body0d.
  line -> body1d.
  ring -> body1d.
  square -> body2d.
  disc -> body2d.
  shell -> body2d.
  tube -> body2d.
  cube -> body3d.
  sphere -> body3d.
  cylinder -> body3d.

  axis -> object.
  fibre -> object.


  mass -> scalar.
  length -> scalar.
  area -> scalar.
  rofg -> scalar.
```

```
{ normal_form }.
%---------------

defn( regular_fibre(Obj,Fibre), hedb,
                  [ magic_fibre(Obj,Fibre), _ ],
                  []   ).
```

```
/* MAGIC : The magic routine which creates fibres */



                              % Any particular fibre only belongs to one object

magic_fibre(Obj,Fibre)
     :- nonvar(Fibre),
        !,
        ?(resfib(Obj,Fibre)).



                              % Every object, however, has an infinite number of
                              %  fibres, so examine the conditions and produce a
                              %  new fibre of the appropriate sort.

magic_fibre(Obj,Fibre)

     :- nonvar(Obj), var(Fibre),

        cc centre(Obj,C),
        cc radius(Obj,R),

        type(Shape,Obj),
        find_fibre_type(Shape,Fshape,Froz),
        find_fibre_range(Froz,R,L1,L2),

        gensym(fibre,Fibre),
        gensym(d,D),

        trace('\nNew %t fibre created, %t, for %t %t.\n',
                                [Fshape,Fibre,Shape,Obj], 2),

        assert( const(D) ),      % Hack to prevent D being solved for!

        (add cont_meas(Obj,D,C,Fibre,L1,L2) &

             % fibre(Fibre) &
             isa(Fshape,Fibre) &
             resfib(Obj,Fibre)   ).



                              % Add facts to data-base

add(Facts)
     :- sel(Facts,F),
        dbentry(F),
        fail.

add(_).


                              % 'resfib' is a hack, here is some more help

nokind(resfib,2).
```

```
/* FIND : Meta level reasoner about what sort of object would
          be a possible fibre for some entity.

        find_fibre_type(Entity,Fibre,Froz)

                Entity is the type of the entity (ML type token)
                Fibre is the type of the fibre (ML type token)
                Froz is the frozen equational definition used

          Provide real range given general frozen equations and the
           the real radius.

        find_fibre_range(Froz,Radius,L1,L2)

                Froz are the frozen equations
                Radius is the real radius of the original entity
                L1 is the real least limit
                L2 is the real greatest limit


*/


                        % Find fibre type

find_fibre_type(Entity,Fibre,Froz)
     :- shape_defn(Entity,Dim1,Eqns),
        freeze(Eqns,Froz),
        non_weird(Froz),
        lowerdim(Dim1,Dim),
        shape_defn(Fibre,Dim,Feqns),
        match_eqns(Feqns,Froz).



                        % Freeze a dimension in the equations

freeze(Eqns,Froz)
     :- sel(Eqns,A,B,Froz),
        friz(A,B).


friz(EqDefn,frozen(C,EqDefn))
     :- set_ineq(EqDefn,_,C,_).



                        % Check for weirdness.  Fibres that need
                        %  to be rotated are no good.

non_weird(Eqns)
     :- sel(Eqns,frozen(C,polar(X))),
        !,
        fail.

non_weird(_).



                        % Lower the dimension by one
```

```
lowerdim(D1,D)
      :- D1 > 0,
         D is D1 - 1.


                           % Matching two equational definitions

match_eqns(A&Rest1,B&Rest2)
      :- !,
         match(A,B),
         match_eqns(Rest1,Rest2).

match_eqns(A,B) :- match(A,B).


                           % Matching two coordinates

match(X,X) :- !.

match(X,Y) :- match2(X,Y), !.

match(X,Y) :- match2(Y,X), !.


match2(constant(C),frozen(C,_)).

match2(constant(C),cartesian(C=_)).

match2(cartesian(C=_),frozen(C,_)).



                           % Find the range over which the fibre can vary

find_fibre_range(Eqns,Radius,L1,L2)
      :- sel(Eqns,A),
         ffr(A,Radius,L1,L2),
         !.


ffr(frozen(C,EqDefn), Radius, L1, L2)
      :- set_ineq(EqDefn,E1,C,E2),
         fr(E1,Radius,L1),
         fr(E2,Radius,L2).


fr(-r,Radius,-Radius) :- !.

fr(r,Radius,Radius) :- !.

fr(X,_,X).


                           % Get the inequality from a coordinate
```

```
set_ineq(cartesian(E1 =< C =< E2),E1,C,E2).

set_ineq(polar(E1 =< C =< E2),E1,C,E2).
```

```
/* DEFN : Equational definitions of various shapes/objects.

        These routines generate equational definitions of certain simple
        regular objects about an origin. This is done by 'construction'
        from simpler objects (initially a point), using the operations
        of "displacement from an axis", "translation along an axis" and
        "rotation through an angle".
        The definitions are of the form:

                <defn> --> <coorddef> & <defn>
                        |  <coorddef>.

                <coorddef> --> constant( <dimsym> )
                            |  cartesian( <defn> )
                            |  polar( <defn> ).

                <defn> --> <num> =< <dimsym> =< <num>
                        |  <dimsym> = <num>
                        |  <dimsym> = <num> # <dimsym> = <num>.

                <dimsym> --> x | y | z.

                <num> --> 0 | -r | r | -pi | pi.

        Thus the atoms x,y,z stand for the three dimensions, their
        interpretation being either cartesian or polar. The origin of the
        coordinate system (Origin) will be such that:

                centre(Object,Origin)

        and the atom r stands for the radius (Radius):

                radius(Object,Radius)

        Given the regular nature of the objects generated these will always
        have reasonable interpretations.


*/


                        % How to construct various objects

construction(point,0,[]).
construction(line,1,[translate]).
construction(ring,1,[displace,rotate]).
construction(square,2,[translate,translate]).
construction(disc,2,[translate,rotate]).
construction(shell,2,[displace,rotate,rotate]).
construction(tube,2,[translate,displace,rotate]).
construction(cube,3,[translate,translate,translate]).
construction(sphere,3,[translate,rotate,rotate]).
construction(cylinder,3,[translate,translate,rotate]).


                        % Definition of the shape of an object

shape_defn(Object,Dim,Eqns2)
     :- construction(Object,Dim,Constr),
```

```
        point_defn(Eqns1),
        perform(Constr,0,Eqns1,Eqns2).


perform([],Rcount,Eqns1,Eqns2)
     :- correct(Rcount,Eqns1,Eqns2).

perform([OP|Rest],Rc1,Eqns1,Eqns3)
     :- add(OP,Eqns1,Eqns2),
        rchk(OP,Rc1,Rc2),
        perform(Rest,Rc2,Eqns2,Eqns3).


add(OP,Eqns1,Eqns2)
     :- sel(Eqns1,E1,E2,Eqns2),
        operation(OP,E1,E2),
        !.


                              % Routines to correct for rotations

rchk(rotate,N,N1) :- !, N1 is N + 1.

rchk(_,N,N).


correct(0,Eqns,Eqns) :- !.

correct(N1,A&Rest1,B&Rest2)
     :- restrict(A,B),
        N is N1 - 1,
        correct(N,Rest1,Rest2).

correct(N1,X&Rest1,X&Rest2)
     :- !,
        correct(N1,Rest1,Rest2).

correct(1,A,B)
     :- restrict(A,B).


                              % Basic operations

point_defn( constant(x) & constant(y) & constant(z) ).


operation(displace, constant(C), cartesian( C = r # C = -r ) ).

operation(translate, constant(C), cartesian( -r =< C =< r ) ).

operation(rotate, constant(C), polar( -pi =< C =< pi ) ).


restrict( cartesian( C=A # C=B ), cartesian( C=A ) ).

restrict( cartesian( -R =< C =< R ), cartesian( 0 =< C =< R ) ).

restrict( polar( -pi =< C =< pi ), polar( 0 =< C =< pi ) ).
```

```
/* SEL : Utilities for selecting elements from conjunctions */

                              % Select and construct

sel(A&Rest,A,B,B&Rest).

sel(X&Rest1,A,B,X&Rest2) :- !, sel(Rest1,A,B,Rest2).

sel(A,A,B,B).


                              % Select only

sel(A&Rest,A).

sel(X&Rest,A) :- !, sel(Rest,A).

sel(A,A).
```

```
/* LOAD : Routines for loading special inference rules etc */


                                % Entry point

load([]) :- !.

load([HD|TL])
        :- !,
           load(HD),
           load(TL).

load(File)
        :- seeing(Old),
           see(File),
           repeat,
           read(X),
           shove(X),
           !,
           seen,
           see(Old),
           ttynl, display(File), display(' loaded.'), ttynl.


                                % Examine and assert

shove(end_of_file).

shove({_}) :- !, fail.

shove(X)
        :- hit(X,X2,Where),
           ass(Where,X2),
           fail.


                                % Rewrite input into appropriate clauses

hit((T1->T2),typetreeu(T1,T2),type) :- !.

hit((Head<==Body),(Head:-Body),assertz) :- !.

hit((Head<--Body1),(Head:-Body2),assertz)
        :- cchit(Body1,Body2),
           !.

hit((Body1-->Head),(Head:-Body2),assertz)
        :- cchit(Body1,Body2),
           !.

hit(X,X,asserta).


cchit(A & B, (CCA,CCB))
        :- !,
           cchit(A,CCA),
           cchit(B,CCB).
```

```
cchit({X},X) :- !.

cchit(X, cc(X)).


                                   % Positional assert plus special handling
                                   %  for types.

ass(asserta,X) :- asserta(X).

ass(assertz,X) :- assertz(X).

ass(type,typetreeu(T1,T2))
     :- ( retract(typetreeu(T1,X)) ; true ),
        assertz(typetreeu(T1,T2)),
        !.
```

```
/* TEST : Random test routines */


                                   % Test shape_defn

dtest(Object)
      :- shape_defn(Object,_,Eqns),
         writef('\n%t\n%c',[Object,Eqns]),
         fail.

dtest(_).



                                   % Test find_fibre_type

ftest(Objtype)
      :- construction(Objtype,_,_),
         writef('\nFibres for a %t\n',[Objtype]),
         find_fibre_type(Objtype,Ftype,Froz),
         writef('\n  %t\n%c',[Ftype,Froz]),
         fail.

ftest(_).
```

```
/* QIK */

s :- save('scra:foo').

r :- restore(foo).
```

```
/* MOFI1.PRB : A moment of inertia problem */

problem(mofi1,'Radius of Gyration of a line\n\n',[]).

period(now).

line(l1).
centre(l1,midpt).
radius(l1,a).
mass_per_length(l1,m,now).

axis(ax).
meets(ax,l1,midpt).

rad_of_gyr(l1,ax,rg1,now).

given(a).
given(m).
sought(rg1).
```

Older problems & traces

```
/* MOFI2.PRB : 2nd Moments of Inertia problem */

problem(mofi2,'Radius of Gyration of a square\n\n',[]).


period(now).

square(sq).
centre(sq,midpt).
radius(sq,a).
mass_per_area(sq,m,now).

axis(ax).
meets(ax,sq,midpt).

rad_of_gyr(sq,ax,rg1,now).

given(a).
given(m).
sought(rg1).
```

```
/* MOFI4.PRB : 4th Moments of Inertia problem */

problem(mofi4,'Radius of Gyration of a Disc\n\n',[]).


period(now).

disc(disc1).
centre(disc1,midpt).
radius(disc1,a).
mass(disc1,m,now).

axis(ax).
meets(ax,disc1,midpt).

rad_of_gyr(disc1,ax,rg1,now).

given(a).
given(m).
sought(rg1).
```

```
                        % How to construct various objects

construction(point,0,[]).
construction(line,1,[translate]).
construction(ring,1,[displace,rotate]).
construction(square,2,[translate,translate]).
construction(disc,2,[translate,rotate]).
construction(shell,2,[displace,rotate,rotate]).
construction(tube,2,[translate,displace,rotate]).
construction(cube,3,[translate,translate,translate]).
construction(sphere,3,[translate,rotate,rotate]).
construction(cylinder,3,[translate,translate,rotate]).


                        % Basic operations

point_defn( constant(x) & constant(y) & constant(z) ).


operation(displace, constant(C), cartesian( C = r # C = -r ) ).

operation(translate, constant(C), cartesian( -r =< C =< r ) ).

operation(rotate, constant(C), polar( -pi =< C =< pi ) ).

restrict( cartesian( C=A # C=B )), cartesian( C=A ) ).

restrict( cartesian( -R =< C =< R ), cartesian( 0 =< C =< R ) ).

restrict( polar( -pi =< C =< pi ), polar( 0 =< C =< pi ) ).


                        % Freeze a dimension in the equations

freeze(Eqns,Froz)
     :- sel(Eqns,A,B,Froz),
        friz(A,B).


friz(EqDefn,frozen(C,EqDefn))
     :- set_ineq(EqDefn,_,C,_).


                        % Matching two coordinates

match(X,X).

match(constant(C),frozen(C,_)).

match(constant(C),cartesian(C=_)).

match(cartesian(C=_),frozen(C,_)).
```

```
Mecho Problem Solver
Prolog-10  version 3

| ?- restore(alan).

yes
| ?- input(mofi1).

Problem from file : mofi1.prb

Radius of Gyration of a line

 Let now be a new period
 Let l1 be a new line
 Note: a (of type length) was used in a radius definition (2)
 Note: m (of type mass) was used in a mass_per_length definition (2)
 Let ax be a new axis
 Note: rg1 (of type rofg) was used in a rad_of_gyr definition (3)

mofi1 problem read into data base.


yes
| ?- go.

Attempting to solve for [rg1] in terms of [a,m]


I am now trying to solve for rg1 without introducing any unknowns.

 Applicable formulae : [moment_of_inertia,parallel_axes]
 (try  moment_of_inertia)

New point fibre created, fibre1, for line l1.
 Let fibre1 be a new point
 Trying to apply strategy(moment_of_inertia,situation(l1,ax,fibre1,now))
 (try  parallel_axes)

No luck - I will now accept unknowns in solving for rg1.

 Applicable formulae : [moment_of_inertia,parallel_axes]
 (try  moment_of_inertia)

New point fibre created, fibre2, for line l1.
 Let fibre2 be a new point
 Trying to apply strategy(moment_of_inertia,situation(l1,ax,fibre2,now))
 Let mass1 be the mass of l1.
 Note: mass1 (of type mass) was used in a mass definition (2)
 Let mass2 be the mass of fibre2.
 Note: mass2 (of type mass) was used in a mass definition (2)
 Let rad_of_gyr1 be the rad_of_gyr of fibre2.
 Note: rad_of_gyr1 (of type rofg) was used in a rad_of_gyr definition (3)

 Equation-1 : mass1*rg1^2=integrate(mass2*rad_of_gyr1^2,-a,a,d2)
 formed by applying : strategy(moment_of_inertia,situation(l1,ax,fibre2,now))

 This equation solves for rg1 but introduces [mass1,mass2,rad_of_gyr1].

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

```
 So now I must solve for [mass1,mass2,rad_of_gyr1]
                   given [rg1,a,m]

I am now trying to solve for mass1 without introducing any unknowns.

 Applicable formulae : [mass_per_area,mass_per_length,moment_of_inertia,resol
 (try  mass_per_area)
 Trying to apply strategy(mass_per_area,situation(l1,now))
 (try  mass_per_length)
 Trying to apply strategy(mass_per_length,situation(l1,now))

 Equation-2 : mass1=2*a*m
 formed by applying : strategy(mass_per_length,situation(l1,now))

 This equation solves for mass1.

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

 So now I must solve for [mass2,rad_of_gyr1]
                   given [mass1,rg1,a,m]

I am now trying to solve for mass2 without introducing any unknowns.

 Applicable formulae : [mass_per_area,mass_per_length,moment_of_inertia,resol
 (try  mass_per_area)
 Trying to apply strategy(mass_per_area,situation(fibre2,now))
 (try  mass_per_length)
 Trying to apply strategy(mass_per_length,situation(fibre2,now))
 (try  moment_of_inertia)
 (try  resolve)

No luck - I will now accept unknowns in solving for mass2.

 Applicable formulae : [mass_per_area,mass_per_length,moment_of_inertia,resol
 (try  mass_per_area)
 Trying to apply strategy(mass_per_area,situation(fibre2,now))
 Let mass_per_area1 be the mass_per_area of fibre2.
 Note: mass_per_area1 (of type mass) was used in a mass_per_area definition (
 Let area1 be the area of fibre2.
 Note: area1 (of type area) was used in a area definition (2)

 Equation-3 : mass2=area1*mass_per_area1
 formed by applying : strategy(mass_per_area,situation(fibre2,now))

 This equation solves for mass2 but introduces [area1,mass_per_area1].

[ Unknowns allowed ]   Do you accept this equation ? no.

 Equation-3 rejected.

 (try  mass_per_length)
 Trying to apply strategy(mass_per_length,situation(fibre2,now))
 Let length1 be the length of fibre2.
 Note: length1 (of type length) was used in a length definition (2)

 Equation-4 : mass2=length1*m
 formed by applying : strategy(mass_per_length,situation(fibre2,now))

 This equation solves for mass2 but introduces [length1].
```

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

 So now I must solve for [rad_of_gyr1,length1]
                   given [mass2,mass1,rg1,a,m]

I am now trying to solve for rad_of_gyr1 without introducing any unknowns.

 Applicable formulae : [moment_of_inertia,parallel_axes]
 (try  moment_of_inertia)
 (try  parallel_axes)

No luck - I will now accept unknowns in solving for rad_of_gyr1.

 Applicable formulae : [moment_of_inertia,parallel_axes]
 (try  moment_of_inertia)
 (try  parallel_axes)

I am unable to solve for rad_of_gyr1.

I will go back to solve for mass2 again

 Equation-4 rejected.

 (try  moment_of_inertia)
 (try  resolve)

I am unable to solve for mass2.

I will go back to solve for rg1 again

 Equation-1 rejected.

 (try  parallel_axes)

I am unable to solve for rg1.

no
| ?- restore(alan).

yes
| ?- input(mofi2).

Problem from file : mofi2.prb

Radius of Gyration of a square

 Let now be a new period
 Let sq be a new square
 Note: a (of type length) was used in a radius definition (2)
 Note: m (of type mass) was used in a mass_per_area definition (2)
 Let ax be a new axis
 Note: rg1 (of type rofg) was used in a rad_of_gyr definition (3)

mofi2 problem read into data base.


yes
| ?- go.

Attempting to solve for [rg1] in terms of [a,m]
```

```
I am now trying to solve for rs1 without introducing any unknowns.

 Applicable formulae : [moment_of_inertia,parallel_axes]
 (try  moment_of_inertia)

New line fibre created, fibre1, for square sq.
 Let fibre1 be a new line
 Trying to apply strategy(moment_of_inertia,situation(sq,ax,fibre1,now))
 (try  parallel_axes)

No luck - I will now accept unknowns in solving for rs1.

 Applicable formulae : [moment_of_inertia,parallel_axes]
 (try  moment_of_inertia)

New line fibre created, fibre2, for square sq.
 Let fibre2 be a new line
 Trying to apply strategy(moment_of_inertia,situation(sq,ax,fibre2,now))
 Let mass1 be the mass of sq.
 Note: mass1 (of type mass) was used in a mass definition (2)
 Let mass2 be the mass of fibre2.
 Note: mass2 (of type mass) was used in a mass definition (2)
 Let rad_of_gyr1 be the rad_of_gyr of fibre2.
 Note: rad_of_gyr1 (of type rofg) was used in a rad_of_gyr definition (3)

 Equation-1 : mass1*rg1^2=integrate(mass2*rad_of_gyr1^2,-a,a,d2)
 formed by applying : strategy(moment_of_inertia,situation(sq,ax,fibre2,now))

 This equation solves for rg1 but introduces [mass1,mass2,rad_of_gyr1].

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

 So now I must solve for [mass1,mass2,rad_of_gyr1]
                  given [rg1,a,m]

I am now trying to solve for mass1 without introducing any unknowns.

 Applicable formulae : [mass_per_area,mass_per_length,moment_of_inertia,resol
 (try  mass_per_area)
 Trying to apply strategy(mass_per_area,situation(sq,now))

 Equation-2 : mass1=(2*a)^2*m
 formed by applying : strategy(mass_per_area,situation(sq,now))

 This equation solves for mass1.

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

 So now I must solve for [mass2,rad_of_gyr1]
                  given [mass1,rg1,a,m]

I am now trying to solve for mass2 without introducing any unknowns.

 Applicable formulae : [mass_per_area,mass_per_length,moment_of_inertia,resol
 (try  mass_per_area)
 Trying to apply strategy(mass_per_area,situation(fibre2,now))
 (try  mass_per_length)
 Trying to apply strategy(mass_per_length,situation(fibre2,now))
```

```
 (try  moment_of_inertia)
 (try  resolve)

No luck - I will now accept unknowns in solving for mass2.

 Applicable formulae : [mass_per_area,mass_per_length,moment_of_inertia,resol
 (try  mass_per_area)
 Trying to apply strategy(mass_per_area,situation(fibre2,now))
 Let radius1 be the radius of fibre2.
 Note: radius1 (of type length) was used in a radius definition (2)

 Equation-3 : mass2=2*radius1*d(d2)*m
 formed by applying : strategy(mass_per_area,situation(fibre2,now))

 This equation solves for mass2 but introduces [radius1].

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

 So now I must solve for [rad_of_gyr1,radius1]
                    given [mass2,mass1,rg1,a,m]

I am now trying to solve for rad_of_gyr1 without introducing any unknowns.

 Applicable formulae : [moment_of_inertia,parallel_axes]
 (try  moment_of_inertia)
 (try  parallel_axes)

No luck - I will now accept unknowns in solving for rad_of_gyr1.

 Applicable formulae : [moment_of_inertia,parallel_axes]
 (try  moment_of_inertia)
 (try  parallel_axes)

I am unable to solve for rad_of_gyr1.

I will go back to solve for mass2 again

 Equation-3 rejected.

 (try  mass_per_length)
 Trying to apply strategy(mass_per_length,situation(fibre2,now))
 Let mass_per_length1 be the mass_per_length of fibre2.
 Note: mass_per_length1 (of type mass) was used in a mass_per_length definiti

 Equation-4 : mass2=2*radius1*mass_per_length1
 formed by applying : strategy(mass_per_length,situation(fibre2,now))

 This equation solves for mass2 but introduces [radius1,mass_per_length1].

[ Unknowns allowed ]   Do you accept this equation ? no.

 Equation-4 rejected.

 (try  moment_of_inertia)
 (try  resolve)

I am unable to solve for mass2.

I will go back to solve for rg1 again
```

```
 Equation-1 rejected.

 (try  parallel_axes)

I am unable to solve for rs1.

no
| ?- halt.
```

```
yes
| ?- input(mofi4).

Problem from file : mofi4.prb

Radius of Gyration of a Disc

 Let now be a new period
 Let disc1 be a new disc
 Note: a (of type length) was used in a radius definition (2)
 Note: m (of type mass) was used in a mass definition (2)
 Let ax be a new axis
 Note: rg1 (of type rofg) was used in a rad_of_gyr definition (3)

mofi4 problem read into data base.


yes
| ?- go.
                          R
Attempting to solve for [rg1] in terms of [a,m]


I am now trying to solve for rg1 without introducing any unknowns.

 Applicable formulae : [moment_of_inertia,parallel_axes]
 (try  moment_of_inertia)

New ring fibre created, fibre1, for disc disc1.
 Let fibre1 be a new ring
 Trying to apply strategy(moment_of_inertia,situation(disc1,ax,fibre1,now))
 (try  parallel_axes)

No luck - I will now accept unknowns in solving for rg1.

 Applicable formulae : [moment_of_inertia,parallel_axes]
 (try  moment_of_inertia)

New ring fibre created, fibre2, for disc disc1.
 Let fibre2 be a new ring
 Trying to apply strategy(moment_of_inertia,situation(disc1,ax,fibre2,now))
 Let mass1 be the mass of fibre2.
 Note: mass1 (of type mass) was used in a mass definition (2)

 Equation-1 : m*rg1^2=integrate(mass1*(d2^2)^2,0,a,d2)
 formed by applying : strategy(moment_of_inertia,situation(disc1,ax,fibre2,now))

 This equation solves for rg1 but introduces [mass1].

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

 So now I must solve for [mass1]
                    given [rg1,a,m]

I am now trying to solve for mass1 without introducing any unknowns.

 Applicable formulae : [mass_per_area,mass_per_length,moment_of_inertia,resolve]
 (try  mass_per_area)
 Trying to apply strategy(mass_per_area,situation(fibre2,now))
```

```
  (try  mass_per_length)
  Trying to apply strategy(mass_per_length,situation(fibre2))
  (try  moment_of_inertia)
  (try  resolve)

No luck - I will now accept unknowns in solving for mass1.

  Applicable formulae : [mass_per_area,mass_per_length,moment_of_inertia,resolve]
  (try  mass_per_area)
  Trying to apply strategy(mass_per_area,situation(fibre2,now))
  Let mass_per_area1 be the mass_per_area of disc1.
  Note: mass_per_area1 (of type mass) was used in a mass_per_area definition (2)

  Equation-2 : mass1=2*pi*d2*d(d2)*mass_per_area1
  formed by applying : strategy(mass_per_area,situation(fibre2,now))

  This equation solves for mass1 but introduces [mass_per_area1].

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

  So now I must solve for [mass_per_area1]
                    given [mass1,rg1,a,m]

I am now trying to solve for mass_per_area1 without introducing any unknowns.

  Applicable formulae : [mass_per_area,mass_per_length,moment_of_inertia,resolve]
  (try  mass_per_area)
  Trying to apply strategy(mass_per_area,situation(disc1,now))

  Equation-3 : m=pi*a^2*mass_per_area1
  formed by applying : strategy(mass_per_area,situation(disc1,now))

  This equation solves for mass_per_area1.

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

  So now I must solve for []
                    given [mass_per_area1,mass1,rg1,a,m]


Equations extracted :
    m*rg1^2=integrate(mass1*(d2^2)^2,0,a,d2)
    mass1=2*pi*d2*d(d2)*mass_per_area1
    m=pi*a^2*mass_per_area1


yes
| ?- halt.
```

Moments of Inertia

Old - (original!)

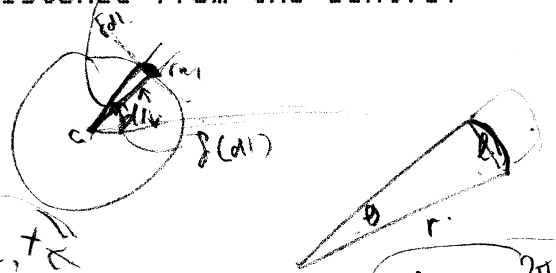
```
/* FORMUL : Formulae et al for the Moment of inertia problems */


{ problem_solving_rules }.
%--------------------------


relates(moment_of_inertia,[mass,rofg]).


prepare(moment_of_inertia,Q,rofg,rad_of_gyr(Obj,Axis,RG,Per),
                situation(Obj,Axis,Fibre,Per))
    <-- meets(Axis,Obj,Origin) &
        centre(Obj,Origin) &
        regular_fibre(Obj,Fibre).


isform(moment_of_inertia, situation(Obj,Axis,Fibre,Per),
                M*RG^2 = integrate(Mf*RGf^2, A,B,X))
    <-- mass(Obj,M,Per) &
        rad_of_gyr(Obj,Axis,RG,Per) &
        mass(Fibre,Mf,Per) &
        rad_of_gyr(Fibre,Axis,RGf,Per) &
        range(Fibre,A,B) &
        perp_dist(Fibre,Axis,X,Per).
```

```
/* MK1 : Meta level knowledge for the Moment of inertia problems */

{ meta_knowledge }.
%------------------

argstruct(basetype,2,
                [type,object],
                [arg,arg]).

argstruct(centre,2,
                [object,point],
                [arg,val]).

argstruct(dist_along,4,
                [object,point,point,length],
                [arg,arg,arg,arg]).

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

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

argstruct(meets,3,
                [axis,object,point],
                [arg,arg,arg]).

argstruct(radius,2,
                [object,length],
                [arg,val]).

argstruct(rad_of_gyr,4,
                [object,axis,rofg,time],
                [arg,arg,val,arg]).

argstruct(range,3,
                [fibre,length,length],
                [arg,arg,arg]).

argstruct(regular_fibre,2,
                [object,fibre],
                [arg,arg]).

argstruct(perp_dist,4,
                [object,object,length,time],
                [arg,arg,val,arg]).


defn( regular_fibre(Obj,Fibre), hedb,
                [ magic_fibre(Obj,Fibre), _ ],
                []  ).
```

```
/* INF1 : Inference rules for the Moment of inertia problems */


{ inference_rules }.
%--------------------


                                % The mass of a 1D body can be derived from its
                                %  mass per unit length.

mass(Body,M*L,Per)
     <-- body1d(Body) &
         length(Body,L,Per) &
         mass_per_length(Body,M,Per).


                                % The mass of a 0D fibre depends on the mass per unit
                                %  length of its 1D object and an infitesimal length
                                %  derived from its distance from the centre.

mass(Fibre,M*delta(D),Per)
     <-- fibre(Fibre) &
         regular_fibre(Body,Fibre) &
         body1d(Body) &
         mass_per_length(Body,M,Per) &
         centre(Body,C) &
         dist_along(Body,C,Fibre,D).
```

fibre m x dl m

length(fibre, x)



$\frac{l}{\theta} = \frac{2\pi r}{2\pi}$

$l = r.\delta(dl)$

```
                                % The length of a line is twice its radius

length(Line,2*R,Per)
     <-- line(Line) &
         radius(Line,R).


                                % The length of a ring based on its radius

length(Ring,2*pi*R,Per)
     <-- ring(Ring) &
         radius(Ring,R).
```

length (Arc, θ*R, Per) ←
arc(Arc), angle_subtended(Arc
radius(Arc, R)

```
                                % Since in the current repn an axis is defined
                                %  to be perpendicular to the whole object, the
                                %  perpendicular distance can be found in terms
                                %  of the distance along the object.

perp_dist(X,Axis,D,Per)
     <-- axis(Axis) &
         meets(Axis,Obj,Point) &
         dist_along(Obj,Point,X,D).


{ meta_hacks }.
```

(inf1 continued)

```
%--------------

                                 % The radius of gyration of an object may be known
                                 %  on the basis of its type. For fibres this means
                                 %  their base type.

rad_of_gyr(X,Axis,RG,Per)
     <== cc perp_dist(X,Axis,D,Per),
         type(Type,X),
         known_rofg(Type,X,D,RG).


known_rofg(point,_,D,D).

known_rofg(fibre,F,D,Ans)
     <== cc basetype(Btype,F),
         known_rofg(Btype,F,D,Ans).
```

```
/* MKINF2 : Additional meta knowledge and inference rules
            for dealing with 2D objects.

*/



{ meta_knowledge }.
%-----------------


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

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



{ inference_rules }.
%-----------------



                        % The mass of 2D body can be derived from its
                        %  mass per unit area.

mass(Body,M*A,Per)
    <-- body2d(Body) &
        area(Body,A,Per) &
        mass_per_area(Body,M,Per).



                        % The area of a square

area(Square,(2*A)^2,Per)
    <-- square(Square) &
        radius(Square,A).



                        % The area of a disc

area(Disc,pi*R^2,Per)
    <-- disc(Disc) &
        radius(Disc,R).
```

```
/* MAGIC : The magic routine which creates fibres */


                              % Any particular fibre only belongs to one object

magic_fibre(Obj,Fibre)
     :- nonvar(Fibre),
        !,
        ?(regfib(Obj,Fibre)).



                              % Every object, however, has an infinite number of
                              %  fibres, so examine the conditions and produce a
                              %  new fibre of the appropriate sort.

magic_fibre(Obj,Fibre)

     :- nonvar(Obj), var(Fibre),

        cc centre(Obj,C),
        cc radius(Obj,R),

        type(Shape,Obj),
        find_fibre_type(Shape,Fshape,Froz),
        find_fibre_range(Froz,R,L1,L2),

        gensym(fibre,Fibre),
        gensym(d,D),

        trace('\nNew %t fibre created, %t, for %t %t.\n',
                                [Fshape,Fibre,Shape,Obj], 2),

        (add fibre(Fibre) &
             regfib(Obj,Fibre) &
             basetype(Fshape,Fibre) &
             range(Fibre,L1,L2) &
             dist_along(Obj,C,Fibre,D) ).


                              % Add facts to data-base

add(Facts)
     :- sel(Facts,F),
        dbentry(F),
        fail.

add(_).


                              % 'regfib' is a hack, here is some more help

nokind(regfib,2).
```

```
/* FIND : Meta level reasoner about what sort of object would
           be a possible fibre for some entity.

        find_fibre_type(Entity,Fibre,Froz)

                Entity is the type of the entity (ML type token)
                Fibre is the type of the fibre (ML type token)
                Froz is the frozen equational definition used

          Provide real range given general frozen equations and the
           the real radius.

        find_fibre_range(Froz,Radius,L1,L2)

                Froz are the frozen equations
                Radius is the real radius of the original entity
                L1 is the real least limit
                L2 is the real greatest limit


*/


                            % Find fibre type

find_fibre_type(Entity,Fibre,Froz)
     :- shape_defn(Entity,Dim1,Eqns),
        freeze(Eqns,Froz),
        lowerdim(Dim1,Dim),
        shape_defn(Fibre,Dim,Feqns),
        match_eqns(Feqns,Froz).



                            % Freeze a dimension in the equations

freeze(Eqns,Froz)
     :- sel(Eqns,A,B,Froz),
        friz(A,B).


friz(linear(E1 =< C =< E2), frozen(C, E1 =< C =< E2) ).

friz(polar(E1 =< C =< E2), frozen(C, E1 =< C =< E2) ).



                            % Lower the dimension by one

lowerdim(D1,D)
     :- D1 > 0,
        D is D1 - 1.



                            % Matching two equational definitions

match_eqns(A&Rest1,B&Rest2)
     :- !,
        match(A,B),
```

```
        match_eqns(Rest1,Rest2).

match_eqns(A,B) :- match(A,B).


match(X,X).

match(constant(C),frozen(C,_)).

match(constant(C),linear(C=_)).



                              % Find the range over which the fibre can vary

find_fibre_range(Eqns,Radius,L1,L2)
     :- sel(Eqns,A),
        ffr(A,Radius,L1,L2),
        !.


ffr(frozen(X,E1 =< X =< E2), Radius, L1, L2)
     :- fr(E1,Radius,L1),
        fr(E2,Radius,L2).


fr(-r,Radius,-Radius) :- !.

fr(r,Radius,Radius) :- !.

fr(X,_,X).
```

/

Automatic generation of Equational Definitions.

OLD - (original!)

```
Utilities package
Prolog-10  version 3

| ?- [ ops2, sel, defn, test ].

ops2 consulted   20 words      0.11 sec.

sel consulted   112 words      0.19 sec.

defn consulted   706 words      1.03 sec.

test consulted   44 words      0.07 sec.

yes
| ?- dtest(_).

point

    constant(x)
    constant(y)
    constant(z)

line

    linear(-r=<x=<r)
    constant(y)
    constant(z)

ring

    linear(x=r)
    polar(0=<y=<2*pi)
    constant(z)

ring

    linear(x=r#x= -r)
    polar(0=<y=<pi)
    constant(z)

square

    linear(-r=<x=<r)
    linear(-r=<y=<r)
    constant(z)

disc

    linear(0=<x=<r)
    polar(0=<y=<2*pi)
    constant(z)

disc

    linear(-r=<x=<r)
    polar(0=<y=<pi)
    constant(z)

shell
```

```
     linear(x=r)
     polar(0=<y=<pi)
     polar(0=<z=<2*pi)

shell

     linear(x=r)
     polar(0=<y=<2*pi)
     polar(0=<z=<pi)

shell

     linear(x=r#x= -r)
     polar(0=<y=<pi)
     polar(0=<z=<pi)

tube

     linear(0=<x=<r)
     linear(y=r#y= -r)
     polar(0=<z=<2*pi)

tube

     linear(-r=<x=<r)
     linear(y=r)
     polar(0=<z=<2*pi)

tube

     linear(-r=<x=<r)
     linear(y=r#y= -r)
     polar(0=<z=<pi)

cube

     linear(-r=<x=<r)
     linear(-r=<y=<r)
     linear(-r=<z=<r)

sphere

     linear(0=<x=<r)
     polar(0=<y=<pi)
     polar(0=<z=<2*pi)

sphere

     linear(0=<x=<r)
     polar(0=<y=<2*pi)
     polar(0=<z=<pi)

sphere

     linear(-r=<x=<r)
     polar(0=<y=<pi)
     polar(0=<z=<pi)

cylinder
```

```
    linear(0=<x=<r)
    linear(-r=<y=<r)
    polar(0=<z=<2*pi)

cylinder

    linear(-r=<x=<r)
    linear(0=<y=<r)
    polar(0=<z=<2*pi)

cylinder

    linear(-r=<x=<r)
    linear(-r=<y=<r)
    polar(0=<z=<pi)

yes
| ?- halt
|
```

```
/* TEST : Random test routines */

                                   % Test share_defn

dtest(Object)
      :- share_defn(Object,_,Eqns),
         writef('\n%t\n%c',[Object,Eqns]),
         fail.

dtest(_).
```

```
/* DEFN : Equational definitions of various shapes/objects.

        These routines generate equational definitions of certain simple
        regular objects about an origin. This is done by 'construction'
        from simpler objects (initially a point), using the operations
        of "displacement from an axis", "translation along an axis" and
        "rotation through an angle".
        The definitions are of the form:

                <defn> --> <coorddef> & <defn>
                        |  <coorddef>.

                <coorddef> --> constant( <dimsym> )
                            |  linear( <ineq> )
                            |  polar( <ineq> ).

                <ineq> --> <num> =< <dimsym> =< <num>
                        |  <dimsym> = <num>
                        |  <dimsym> = <num> # <dimsym> = <num>.

                <dimsym> --> x | y | z.

                <num> --> 0 | r | -r | pi | 2*pi.

        Thus the atoms x,y,z stand for the three dimensions, their
        interpretation being either linear or polar. The origin of the
        coordinate system (Origin) will be such that:

                centre(Object,Origin)

        and the atom r stands for the radius (Radius):

                radius(Object,Radius)

        Given the regular nature of the objects generated these will always
        have reasonable interpretations.


*/


                             % How to construct various objects

construction(point,0,[]).
construction(line,1,[translate]).
construction(ring,1,[displace,rotate]).
construction(square,2,[translate,translate]).
construction(disc,2,[translate,rotate]).
construction(shell,2,[displace,rotate,rotate]).
construction(tube,2,[translate,displace,rotate]).
construction(cube,3,[translate,translate,translate]).
construction(sphere,3,[translate,rotate,rotate]).
construction(cylinder,3,[translate,translate,rotate]).


                             % Definition of the shape of an object

shape_defn(Object,Dim,Eqns2)
     :- construction(Object,Dim,Constr),
```

```
          point_defn(Eqns1),
          perform(Constr,0,Eqns1,Eqns2).


perform([],Rcount,Eqns1,Eqns2)
      :- correct(Rcount,Eqns1,Eqns2).

perform([OP|Rest],Rc1,Eqns1,Eqns3)
      :- add(OP,Eqns1,Eqns2),
         rchk(OP,Rc1,Rc2),
         perform(Rest,Rc2,Eqns2,Eqns3).


add(OP,Eqns1,Eqns2)
      :- sel(Eqns1,E1,E2,Eqns2),
         operation(OP,E1,E2),
         !.


                              % Routines to correct for rotations

rchk(rotate,N,N1) :- !, N1 is N + 1.

rchk(_,N,N).


correct(0,Eqns,Eqns) :- !.

correct(N1,A&Rest1,B&Rest2)
      :- restrict(A,B),
         N is N1 - 1,
         correct(N,Rest1,Rest2).

correct(N1,X&Rest1,X&Rest2)
      :- !,
         correct(N1,Rest1,Rest2).

correct(1,A,B)
      :- restrict(A,B).


                              % Basic operations

point_defn( constant(x) & constant(y) & constant(z) ).


operation(displace, constant(C), linear( C = r # C = -r ) ).

operation(translate, constant(C), linear( -r =< C =< r ) ).

operation(rotate, constant(C), polar( 0 =< C =< 2*pi ) ).


restrict( linear( A # B ), linear( A ) ).

restrict( linear( -R =< C =< R ), linear( 0 =< C =< R ) ).

restrict( polar( 0 =< C =< 2*pi ), polar( 0 =< C =< pi ) ).
```

```
/* SEL : Utilities for selecting elements from conjunctions */

                                        % Select and construct

sel(A&Rest,A,B,B&Rest).

sel(X&Rest1,A,B,X&Rest2) :- !, sel(Rest1,A,B,Rest2).

sel(A,A,B,B).


                                        % Select only

sel(A&Rest,A).

sel(X&Rest,A) :- !, sel(Rest,A).

sel(A,A).
```

Problems

```
/* MOFI1.PRB : A moment of inertia problem */

problem(mofi1,'Radius of Gyration of a line\n\n',[]).


period(now).

line(l1).
centre(l1,midpt).
radius(l1,a).
mass_per_length(l1,m,now).

axis(ax).
meets(ax,l1,midpt).

rad_of_gyr(l1,ax,rg1,now).

given(a).
given(m).
sought(rg1).



/* MOFI2.PRB : 2nd Moments of Inertia problem */

problem(mofi2,'Radius of Gyration of a square\n\n',[]).


period(now).

square(sq).
centre(sq,midpt).
radius(sq,a).
mass_per_area(sq,m,now).

axis(ax).
meets(ax,sq,midpt).

rad_of_gyr(sq,ax,rg1,now).

given(a).
given(m).
sought(rg1).



/* MOFI3.PRB : 3rd Moments of Inertia problem */

problem(mofi3,'Radius of Gyration of a ring\n\n',[]).


period(now).

ring(rng).
centre(rng,midpt).
radius(rng,a).
mass_per_length(rng,m,now).

axis(ax).
```

```
meets(ax,rng,midpt).

rad_of_gyr(rng,ax,rg1,now).

given(a).
given(m).
sought(rg1).
```

Old

handout at first

talk on this stuff

```
% How to construct various objects

construction(point,0,[]).
construction(line,1,[translate]).
construction(ring,1,[displace,rotate]).
construction(square,2,[translate,translate]).
construction(disc,2,[translate,rotate]).
construction(shell,2,[displace,rotate,rotate]).
construction(tube,2,[translate,displace,rotate]).
construction(cube,3,[translate,translate,translate]).
construction(sphere,3,[translate,rotate,rotate]).
construction(cylinder,3,[translate,translate,rotate]).


% Basic operations

point_defn( constant(x) & constant(y) & constant(z) ).


operation(displace, constant(C), cartesian( C = r # C = -r ) ).

operation(translate, constant(C), cartesian( -r =< C =< r ) ).

operation(rotate, constant(C), polar( -pi =< C =< pi ) ).


restrict( cartesian( C=A # C=B ), cartesian( C=A ) ).

restrict( cartesian( -R =< C =< R ), cartesian( 0 =< C =< R ) ).

restrict( polar( -pi =< C =< pi ), polar( 0 =< C =< pi ) ).


% Freeze a dimension in the equations

freeze(Eqns,Froz)
      :- sel(Eqns,A,B,Froz),
         friz(A,B).


friz(EqDefn,frozen(C,EqDefn))
      :- set_ineq(EqDefn,_,C,_).


% Matching two coordinates

match(X,X).

match(constant(C),frozen(C,_)).

match(constant(C),cartesian(C=_)).

match(cartesian(C=_),frozen(C,_)).
```

```
Utilities package
Prolog-10  version 3

| ?- [ ops2, sel, find, defn, test ].

ops2 consulted    20 words       0.11 sec.

sel consulted   112 words       0.19 sec.

find consulted   510 words       0.70 sec.

defn consulted   668 words       1.04 sec.

test consulted   112 words       0.16 sec.

yes
| ?- dtest(_).    % Generated definitions for various bodies

point

    constant(x)
    constant(y)
    constant(z)

line

    cartesian(-r=<x=<r)
    constant(y)
    constant(z)

ring

    cartesian(x=r)
    polar(-pi=<y=<pi)
    constant(z)

ring

    cartesian(x=r#x= -r)
    polar(0=<y=<pi)
    constant(z)

square

    cartesian(-r=<x=<r)
    cartesian(-r=<y=<r)
    constant(z)

disc

    cartesian(0=<x=<r)
    polar(-pi=<y=<pi)
    constant(z)

disc

    cartesian(-r=<x=<r)
    polar(0=<y=<pi)
    constant(z)
```

```
shell

     cartesian(x=r)
     polar(0=<y=<pi)
     polar(-pi=<z=<pi)

shell

     cartesian(x=r)
     polar(-pi=<y=<pi)
     polar(0=<z=<pi)

shell

     cartesian(x=r#x= -r)
     polar(0=<y=<pi)
     polar(0=<z=<pi)

tube

     cartesian(0=<x=<r)
     cartesian(y=r#y= -r)
     polar(-pi=<z=<pi)

tube

     cartesian(-r=<x=<r)
     cartesian(y=r)
     polar(-pi=<z=<pi)

tube

     cartesian(-r=<x=<r)
     cartesian(y=r#y= -r)
     polar(0=<z=<pi)

cube

     cartesian(-r=<x=<r)
     cartesian(-r=<y=<r)
     cartesian(-r=<z=<r)

sphere

     cartesian(0=<x=<r)
     polar(0=<y=<pi)
     polar(-pi=<z=<pi)

sphere

     cartesian(0=<x=<r)
     polar(-pi=<y=<pi)
     polar(0=<z=<pi)

sphere

     cartesian(-r=<x=<r)
     polar(0=<y=<pi)
     polar(0=<z=<pi)
```

```
cylinder

     cartesian(0=<x=<r)
     cartesian(-r=<y=<r)
     polar(-pi=<z=<pi)

cylinder

     cartesian(-r=<x=<r)
     cartesian(0=<y=<r)
     polar(-pi=<z=<pi)

cylinder

     cartesian(-r=<x=<r)
     cartesian(-r=<y=<r)
     polar(0=<z=<pi)

yes
```

```
| ?- ftest(_).    % Finding fibres by freezing and trying to find match
Fibres for a point

Fibres for a line

  point

    frozen(x,cartesian(-r=<x=<r))
    constant(y)
    constant(z)

Fibres for a ring

  point

    cartesian(x=r)
    frozen(y,polar(-pi=<y=<pi))
    constant(z)

Fibres for a square

  line

    cartesian(-r=<x=<r)
    frozen(y,cartesian(-r=<y=<r))
    constant(z)

Fibres for a disc

  ring

    frozen(x,cartesian(0=<x=<r))
    polar(-pi=<y=<pi)
    constant(z)

  line

    cartesian(-r=<x=<r)
    frozen(y,polar(0=<y=<pi))
    constant(z)
Fibres for a shell

  ring

    cartesian(x=r)
    polar(-pi=<y=<pi)
    frozen(z,polar(0=<z=<pi))

  ring

    cartesian(x=r#x= -r)
    polar(0=<y=<pi)
    frozen(z,polar(0=<z=<pi))

Fibres for a tube

  line
```

wrong!

this is not line
it is sector.

```
    cartesian(-r=<x=<r)
    cartesian(y=r)
    frozen(z,polar(-pi=<z=<pi))

Fibres for a cube

  square

    cartesian(-r=<x=<r)
    cartesian(-r=<y=<r)
    frozen(z,cartesian(-r=<z=<r))

Fibres for a sphere

  shell

    frozen(x,cartesian(0=<x=<r))
    polar(0=<y=<pi)
    polar(-pi=<z=<pi)

  shell

    frozen(x,cartesian(0=<x=<r))
    polar(-pi=<y=<pi)
    polar(0=<z=<pi)

  disc

    cartesian(0=<x=<r)
    polar(-pi=<y=<pi)
    frozen(z,polar(0=<z=<pi))

  disc

    cartesian(-r=<x=<r)
    polar(0=<y=<pi)
    frozen(z,polar(0=<z=<pi))

Fibres for a cylinder

  tube

    cartesian(-r=<x=<r)
    frozen(y,cartesian(0=<y=<r))
    polar(-pi=<z=<pi)

  square

    cartesian(-r=<x=<r)
    cartesian(-r=<y=<r)
    frozen(z,polar(0=<z=<pi))

yes
| ?- halt.
```