;; MOFI.SUB - All Alan's Moments of Inertia stuff

mofi.sub ;; This file
mofi. ;; Program load file
cumeas,
defn,
coord,
senkn,
form1,
form2,
infl,
infl,
mk1,
typ1,
pitch,
findb,
self,
aik,
mofi4.old ;; Problem files
  - mofi1.prb ;;
  - mofi2.prb ;;
  - mofi3.prb ;;
  - mofi4.prb ;;
  - mofi5.prb ;;
  - mofi6.prb ;;
  - mofi7.prb ;;
  - mofi8.prb ;;
  - mofi9.prb ;;
  - mofi10.prb ;;
  mofii.prb ;;
mofii.sol ;; Solution files (traces)
mofi2.sol ;;
mofi3.sol ;;
mofi4.sol ;;
mofi5.sol ;;
mofi6.sol ;;
mofi7.sol ;;
mofii.sol ;;
mofi9.sol ;;
mofii10.sol ;;
mofii11.sol ;;
/* MOFI : All the bits for the Moment of inertia Problems */

Updated: 10 February 81

:- [sel,
   % Utilities for conjunctions
tmeas,
   % Fibre Schema
defn,
   % Algebraic Shape Definitions
enkn,
   % Some general knowledge
fnbn,
   % Date base entry with function properties
patch,
   % Various temporary patches
gik,
   % Quick save and restore
'arith:polvis',
   % for poly_form
'press:misc',
   % for multi_occ
'arith:polpak',
   % for poly
'press:match',
   % for recomp
],

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

,
% continuous measure system already known

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

% make continuous measure system

cont_meas(Obj,X0,axis(Y),Fibre,A,B) :-
    set_defn(Obj,origin,Defn1),
    sel(Defn1, A =< X =< B, VAR=X0, Defn2),
    uniform(X,Obj),
    sensym(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

set_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),
    sensym(Shape,Obj),
    checklist(fn_dbentry,Relns),
    trace(\n % is new fibre defined by \n', [Obj,Relns], 2).

% Test for uniform fibre

uniform(X,_) :- distance_coord(X), !, % either parameter is not angle
uniform(_ ,Obj) :- ncc bodyId(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_orisn(Y,X,X0,Defn,Orsn,Defn,Orsn), % they are perpendicular
move_orisn(X,X,X0,Defn,0Orsn,Defn,N0Orsn,N0Orsn),
    sel(0Defn,X=X0,X=0,NDefn), % axis and fibre coord
    ncc_tangent(axis(X),Dir), % are the same
    define_orisn(X0,Dir,0Orsn,N0Orsn),
move_orisn(rr,C0,0Defn,0Orsn,Defn,N0Orsn,N0Orsn),
    ncc_tangent(axis(rr), [Alpha,O]),
    set_incr(C,0,Alpha,X0,Y0,RO),
    update_defns(X0,Y0,Defn,NDefn),
    define_orisn(RO,[Alpha,O],0Orsn,N0Orsn),

/* 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,Y0/tan(Alpha),Y0*Y0/sin(Alpha)),

/* Update definitions by replacing old coordinates by new */
update_defns(X0,Y0,OXinea & OYinea & Zinea, NXinea & NYinea & Zinea),
    update_inea(X0,OXinea,NXinea),
    update_inea(Y0,OYinea,NYinea),
update_inea(C0, C=V, C=V1),
    poly_form(V + (-1)*C0, V1),
update_inea(C0, A=C=C=B, A1=C=C=B1),
    poly_form(A + (-1)*C0, A1), poly_form(B + (-1)*C0, B1),

/* Assert definitions of new orisn */
define_orisn(RO,Dir,0Orsn,N0Orsn),
    sensym(orisn,N0Orsn),
    fn_dbentry( on(N0Orsn,axis(rr)) ),
    fn_dbentry( separation(0Orsn,N0Orsn,RO,Dir) ),
/* DEFN : Algebraic Definitions of Ideal Objects

Updated: 1st April 1981
*/

% sphere in cylindrical polars

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

% cylinder in cylindrical polars

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

% tube in cylindrical polars

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

% cone in cylindrical polars

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

% hollow sphere in cylindrical polars

is_defn(shell, Shl1, Orisin, r=sqrt(A^2-z^2) & 0=<theta=<2*Pi & -A=<z=<A,
    [isa(shell,Sph), centre(Sph,Orisin), radius(Sph,A)]) ),

% circular disc in cylindrical polar coordinates

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

% circular rings in cylindrical polar coordinates

is_defn(rings, Rns, Orisin, r=A & 0=<theta=<2*Pi & z=0,
    [isa(rings,Rns), centre(Rns,Orisin), radius(Rns,A),
    meets(axis(z),Rns,Orisin)]) ),

% radial line in spherical polar coordinates
% horizontal line in cartesian 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])] ) ,

% 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])] ).
\texttt{perp_dist(Origin, Right, B, [E-90, 0])}
/* MK1 : Meta level knowledge for the Moment of inertia problems

Updated: 10 February 81
*/

{ meta_knowledge },
%------------------

arsstruct(on,2,
    [point,object],
    [ars,ars]),

arsstruct(area,2,
    [object,area],
    [ars,val]),

arsstruct(basetype,2,
    [type,object],
    [ars,ars]),

arsstruct(centre,2,
    [object,point],
    [ars,val]),

arsstruct(dist_along,4,
    [object,point,point,length],
    [ars,ars,ars,ars]),

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

arsstruct(mass_per_vol,2,
    [object,mass],
    [ars,val]),

arsstruct(mass_per_area,2,
    [object,mass],
    [ars,val]),

arsstruct(mass_per_length,2,
    [line,mass],
    [ars,val]),

arsstruct(mass,2,
    [object,mass],
    [ars,val]),

arsstruct(meets,3,
    [line,object,point],
    [ars,ars,ars]),

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

arsstruct(height,2,
    [object,length],
    [ars,val])}
\begin{verbatim}
argsstruct(red_of_syr, 3, 
    [object, line, rof], 
    [ars, ars, val]).

argsstruct(perp_dist, 4, 
    [object, object, length, angle], 
    [ars, ars, val, val]).

argsstruct(separation, 4, 
    [Point, Point, length, angle], 
    [ars, ars, val, val]).

argsstruct(line_sys, 3, 
    [line, Point, Point], 
    [ars, val, val]).

argsstruct(quad_sys, 5, 
    [object, line, line, line], 
    [ars, val, val, val, val]).

argsstruct(cont_meas, 6, 
    [object, scalar, Point, object, scalar, scalar], 
    [ars, val, ars, ars, val, val]).
\end{verbatim}
*START* User BUNDY    HPS [400,405]   Job    TYP1 Seq. 8138 Date 02-Apr-81 14:51:11
File: DSKA;TYP1<005>[400,405;MYMOFI] Created: 02-Apr-81 11:46:07 Printed: 02-Apr

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

\{ types \},
\%

------

body0d -> object,
body1d -> object,
body2d -> object,
body3d -> object,

point -> body0d,
line -> body1d,
rings -> 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)
  \<-- rings(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(Shtl,4*pi*R^2)
  \<-- shell(Shtl) \& radius(Shtl,R),
% The surface area of a cylinder
areas(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,Axes,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,Axes,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(Obj,X,Axes,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.

sd_of_gyr(Pt,Axes,K)
  <-- point(Pt) & perp_dist(Pt,Axes,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\(\text{axis}(rr),\{\text{Alpha},0\}\),
\% 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( tansent(axis(x),[0,0]) ),
;— dbentry( tansent(axis(y),[90,0]) ),
;— dbentry( tansent(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,rofs]),

prepare(moment_of_inertia,Q,rofs,rad_of_syr(Obj,axis(X),RG),
        situation(Obj,axis(X))),

isform(moment_of_inertia, situation(Obj,axis(X)),
        M*RG^2 = integrate(M*RGf^2, A,B,X),
        \[ \text{mass}(Obj, M) \% \]
        \[ \text{rad_of_syr}(Obj, axis(X), RG) \% \]
        \{ cont_meas(Obj,Y0, axis(X),Fibre,A,B) \} \&
        \[ \text{mass}(Fibre,Mf) \% \]
        \[ \text{rad_of_syr}(Fibre, axis(X), RGf) \% \],

% Parallel Axes
relates(parallel_axes,[rofs]),

prepare(parallel_axes,Q,rofs,rad_of_syr(Obj,Axis,RG),
        situation(Obj,Axis,Newaxis))
        \[ \text{meets}(Axis,Obj,Axpt) \% \]
        \[ \text{centre}(Obj,Orisin) \% \]
        \{ diff(Axpt,Orisin) \} \&
        \[ \text{meets}(Newaxis,Obj,Orisin) \% \]
        \[ \text{Parallel}(Axis,Newaxis) \% \],

isform(parallel_axes, situation(Obj,RealAxis,CentreAxis),
        RG = RGC + A^2 )
        \[ \text{rad_of_syr}(Obj,RealAxis,RG) \% \]
        \[ \text{rad_of_syr}(Obj,CentreAxis,RGC) \% \]
        \[ \text{ParP_dist}(RealAxis,CentreAxis,A,Dir) \].
/* FNDB : Temporary stuff for doing function stuff on dbentry

Updated: 5 February 81

fn_dbentry(X)
  :- groundtest(X,sground),!
  trace(\rightarrow \%\n',[X],4),
  dbentry(X).

fn_dbentry(X)
  :- functor(X,F,A),
     arsstruct(Fred,N,Types,Fmap),
     X =.. [..!Ars],
     maksolid(Fmap,Ars,Types),
     trace(\rightarrow \%\n',[X],4),
     dbentry(X),!

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

   errmess(unsr,'Unsround database entry: \%\n'),

% Apply function properties

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

maksolid([F:Frest],[A:Arest],[T:Trest])
  :- maks1(F,A,T),
     maksolid(Frest,Arest,Trest).

maks1(ars,A,_) :- nonvar(A).

maks1(val,A,T) :- csensym(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('scratch'),

r :- restore(foo).
/* MOFI1.PR1 : A moment of inertia problem */

problem(mofii,'Radius of Gyration of a line',[],[]),

line(l1),
point(p1),
point(p2),
line_sys(l1,p1,p2),
separation(origin,p1,a,[0,0]),
separation(origin,p2,a,[0,0]),
on(origin,l1),
mass(l1,m),
rad_of_syr(l1,axis(y),k),
siven(a),
siven(m),
sought(k),
/* MOFI2.PRB : 2nd Moments of Inertia Problem */

problem(mofi2,'Radius of Gyration of a rectangle

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_syr(rect,axis(y),k),

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

% radius of gyration of a line

rad_of_syr(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 Inertias Problem */

Problem(mofi3,'Radius of Gyration of an inclined line

line(l1),
Point(l),
Point(r),
line_sys(l1,l,r),
separation(orisin,l,-a,[0,0]),
separation(orisin,r,a,[0,0]),
on(orisin,l1),
mass(l1,m),

line(axis(rr)),
on(orisin,axis(rr)),
tangent(axis(rr),[alpha,0]),
rad_of_syr(l1,axis(rr),k),
given(a),
given(alpha),
given(m),
sought(k).
/* MOFI4,PRB */ 4th Moments of Inertia Problem */

Problem(mofi4,'Radius of Gyration of a Parallelogram\n\n',[]),

parallelogram(paralmgm),
line(top),
line(bot),
line(left),
line(right),
quad_sys(paralmgm,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(paralmgm,m),

line(axis(rr)),
on(origin,axis(rr)),
tangent(axis(rr),[alpha,0]),
rad_of_syr(paralmgm,axis(rr),k),
siven(a),
siven(b),
siven(alpha),
siven(m),
sousht(k),

% radius of gyration of an inclined line
rad_of_syr(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(rings1),
centre(rings1,origin),
radius(rings1,a),
meets(axis(z),rings1,origin),
mass(rings1,m),

rad_of_gyr(rings1,axis(z),k),

given(a),
given(m),
sought(k),
/* MOFI6.PR6: 6th Moments of Inertia Problem */

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

  disc(disc1),
  centre(disc1,orisin),
  radius(disc1,a),
  meets(axis(z),disc1,orisin),
  mass(disc1,m),
  rad_of_gyr(disc1,axis(z),k),
  siven(a),
  siven(m),
  sousht(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),
/* M0FI7.PRBL : 7th Moments of Inertia Problem */

Problem(mofi7,'Radius of Gyration of a sphere

sphere(sphere1),
centre(sphere1,origin),
radius(sphere1,a),
mass(sphere1,m),
rad_of_gyr(sphere1,axis(z),k),
siven(a),
siven(m),
sousht(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),
/* MOFI8.PRB : 8th Moments of Inertia Problem */

problem(mofi8,'Radius of Gyration of a shell

shell(shell1),
centre(shell1,origin),
radius(shell1,a),
mass(shell1,m),
rad_of_syr(shell1,axis(z),k),
siven(a),
siven(m),
sought(k).

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

rad_of_syr(Rings,Axis,R)
  <-- rings(Rings) & centre(Rings,C) &
  meets(Axis,Rings,C) & radius(Rings,R),
/* 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_syr(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_syr(Disc,Axes,R/sqrt(2))
     <- disc(Disc) & centre(Disc,C) &
         meets(Axis,Disc,C) & radius(Disc,R).
/* MOFI10.PROB : 10th Moments of Inertia Problem */

Problem(mofi10,'Radius of Gyration of a cone

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

rad_of_syr(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_syr(Disc,Axix,R/sqrt(2))
  <-- disc(Disc) & centre(Disc,C) &
    meets(Axis,Disc,C) & radius(Disc,R),
/* MOFI11.PRZ : 11th Moments of Inertia Problem */

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

radius(tube1, a).
heisht(tube1, h).

mass(tube1, m).

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

given(a),
given(h),
given(m),
sousht(k).

rad_of_syr(Rings, Axis, R)
<-- rings(Rings) & centre(Rings, C) &
     meets(Axis, Rings, C) & radius(Rings, R).
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 rots) was used in a rad_of_gyr definition (3)

mofil problem read into data base.

yes
! ?- restore(save).

yes
! ?- input(mofil).

Problem from file: [mofil.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 rots) was used in a rad_of_gyr definition (3)

mofil problem read into data base.

yes
! ?- so.

** 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(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=\int integrate(d(r1)*mass_per_length1*(a*1+a*tr1+a*tr2+a*1+ax-1)^2,a*1+a*tr1+a*tr2+a*1+ax-1,a*1+a*tr1,a*1+a*tr2)

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_length 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(li))

Equation-2: m = \( (\text{mass}\_\text{per}\_\text{length}) \times \text{situation}(\text{li}) \) \times \text{mass}\_\text{per}\_\text{length}

This equation solves for mass_per_length.

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

So now I must solve for \( \text{mass}\_\text{per}\_\text{length} \),
given \( \text{mass}\_\text{per}\_\text{length}, \text{k}, \text{a}, \text{m} \)

Equations extracted:

\[
m = \int (1+a+\text{mass}\_\text{per}\_\text{length}(\text{situation}(\text{li}))) \times \text{mass}\_\text{per}\_\text{length}
\]

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 saving 993 words
0.51 sec. for 26 local shifts and 43 trail shifts
11.01 sec. runtime
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: [90,0] (of type angle) was used in a perp_dist definition (3)
Note: [0,90] (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: 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: -b (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 rofs) was used in a rad_of_curvature definition (3)

mofi2 problem read into data base.

yes

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 nerc3 (continue after error)

Let linel be a new line
Let point1 be a new point
Let point2 be a new point
Let typ_pt1 be a new point

linel is new fibre defined by
isa(line,linel)
linel Sys(line1,Point1,Point2)
isa(point,point1)
isa(point,point2)
isa(point,typ_pt1)
on(typ_pt1,linel)
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 mass\(_1\) be the mass of line\(_1\).

Note: mass\(_1\) (of type mass) was used in a mass definition (2)

Equation-1 : \(m^k=\int\text{mass}\_1*(x-(a\_lx+a\_r))/\text{sort}(\text{mass}\_1)*2,-b,by\)
formed by applying : strategy(mass, of_inertia,situation(rect, axis(y))

This equation solves for \(k\) but introduces [mass\(_1\)].

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

So now I must solve for [mass\(_1\)]
diven \([k,a,b,m]\)

I am now trying to solve for mass\(_1\) without introducing any unknowns.

Applicable formulae : [mass, 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(rect))
(try resolve)

No luck - I will now accept unknowns in solving for mass\(_1\).

Applicable formulae : [mass, 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(rect))
Let mass, per_area\(_1\) be the mass, per_area of rect.
Note: mass, per_area\(_1\) (of type mass, per_area) was used in a mass, per_area definition (2)

Equation-2 : mass\(_1\)=(x-(a\_lx+a\_r))/\text{sort}(\text{mass}\_1)*2,-b,by\)
formed by applying : strategy(mass, per_length, situation(rect))

This equation solves for mass\(_1\) but introduces [mass, per_area\(_1\)].

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

So now I must solve for [mass, per_area\(_1\)]
diven \([mass\(_1\),k,a,b,m]\)

I am now trying to solve for mass, per_area\(_1\) without introducing any unknowns.

Applicable formulae : [mass, 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 : \(m=(b-(b))*a-(a)*\text{mass, per_area}\)
formed by applying : strategy(mass, per_area, situation(rect))

This equation solves for mass, per_area\(_1\).

[ 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:

\[
m^k = \int \frac{mass1 \cdot ((a^k - 1 + a) / \text{sort}(3))^{2, -b + y}}{b + y} \]
\[
mass1 = (a^k - 1 + a - (a^k - 1 + a + a - 1)) \cdot (d(y) \cdot mass\_per\_area1) \]
\[
m = (b - (-b)) \cdot (a - (-a)) \cdot mass\_per\_area1
\]

yes

?- core 87552 (58368 lo-sec + 29184 hi-sec)
heap 53248 = 51256 in use + 1992 free
slocal 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
Problem from file: mofi3.prb

Radius of Gyration of an inclined line

Let \( l_1 \) 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: \([0,0]\\) (of type angle) was used in a separation definition (4)

Let \( \text{axis}(rr) \) be a new line
Let \( \text{typical-point4} \) be a new typical point
Note: \([\alpha,0]\\) (of type angle) was used in an incline definition (2)
Note: \(k\) (of type rofs) was used in a \(\text{rad_of_gyr}\\) definition (3)

mofi3 problem read into data base.

** 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: \([\text{parallel_axes, moment_of_inertia}]\\)
( try \(\text{parallel_axes}\\)
( try \(\text{moment_of_inertia}\\)
Trying to apply strategy(\(\text{moment_of_inertia, situation}(ll, \text{axis}(rr))\\))
Let \( \text{point1} \) be a new point
Note: \(r1\) (of type length) was used in a separation definition (3)

Point1 is new fibre defined by
\[ \text{isa(Point,Point1)}\]
\[ \text{separation(\text{origin},Point1,r1,[0,0])} \]

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

Equation-1 : \(m^2*2=\int\text{d}(r1)*\text{mass_per_length1}*(r1*\sin(\alpha-0))^2, a, a, r1\\)
formed by applying : strategy(\(\text{moment_of_inertia, situation}(ll, \text{axis}(rr))\\))

This equation solves for \(k\\) but introduces \([\text{mass_per_length1}]\\).
[Unknowns allowed] Do you accept this equation? Yes.

So now I must solve for \[\text{mass}\_\text{per}\_\text{length1}\]
siven \([k, a, \alpha, m]\)

I am now trying to solve for \(\text{mass}\_\text{per}\_\text{length1}\) without introducing any unknowns.

Applicable formulae: \([\text{moment}\_\text{of}\_\text{inertia}, \text{mass}\_\text{per}\_\text{vol}, \text{mass}\_\text{per}\_\text{area}, \text{mass}\_\text{per}\_\text{length}, \text{resolve}]\)

(try \text{moment}\_\text{of}\_\text{inertia})
(try \text{mass}\_\text{per}\_\text{vol})
(try \text{mass}\_\text{per}\_\text{area})
(try \text{mass}\_\text{per}\_\text{length})

Trying to apply strategy(\text{mass}\_\text{per}\_\text{length}, \text{situation}(11))

Equation-2:

\[m = (a - (-a)) \times \text{mass}\_\text{per}\_\text{length1}\]

This equation solves for \(\text{mass}\_\text{per}\_\text{length1}\).

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

So now I must solve for []
siven \([\text{mass}\_\text{per}\_\text{length1}, k, a, \alpha, m]\)

Equations extracted:

\[m \times k^2 \times \text{integrate}(d(r1) \times \text{mass}\_\text{per}\_\text{length1} \times (r1 \times \sin(\alpha - 0))^2, -a, a, r1)\]

\[m = (a - (-a)) \times \text{mass}\_\text{per}\_\text{length1}\]

yes

? core 93184 (64000 lo-ses + 29184 hi-ses)
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]
Radius of Gyration of a parallelogram

Let pagrm 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: [90,0J (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-90,0J (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)

mofi4 problem read into data base.

yes
! ?- so.

** 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(pagrm,axis(rr)))
Note: -1*x1/cos(alpha) (of type length) was used in a separation definition (3)
Note: [alpha,0J (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+y1*tan(alpha)^-1+atan(alpha)*y1*1 (of type length)
Note: [0,0J (of type angle) was used in a separation definition (4)
Note: a*sin(alpha)^-1+y1*tan(alpha)^-1+atan(alpha)*y1*1 (of type length) was used

line1 is new fibre defined by
isa(line,line1)
line_sys(line1,Point1,Point2)
isa(Point,Point1)
isa(Point,Point2)
isa(Point,origin2)
on(orisin2,line1)
sePar(orisin2,Point1,a*sin(alpha)-1+y1*tan(alpha)-i+tan(alpha)*y1*1)
sePar(orisin2,Point2,a*sin(alpha)-1+y1*tan(alpha)-i+tan(alpha)*y1*1)

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

Applicable formulae : \([\text{parallel}_\text{axes}, \text{moment}_\text{of}_\text{inertia}]\)
(try parallel_axes)
(try moment_of_inertia)
Trying to apply strategy(moment_of_inertia,situation(pagrm,axis(rr)))
Let mass1 be the mass of line1.
Note: mass1 (of type mass) was used in a mass definition (2)
Let \(-(a*sin(alpha)-1+y1*tan(alpha)-i+tan(alpha)*y1*1)\) be the separation of o
Note: \(-(a*sin(alpha)-1+y1*tan(alpha)-i+tan(alpha)*y1*1)\) (of type length) was

Equation-1 : m*k^2=integrate(mass1*((a*sin(alpha)-1+y1*tan(alpha)-1+tan(alpha)*y1*1)~1)
formed by applying : strategy(moment_of_inertia,situation(pagrm,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 : \([\text{moment}_\text{of}_\text{inertia}, \text{mass}_\text{per}_\text{vol}, \text{mass}_\text{per}_\text{area}, \text{mass}_\text{per}_\text{length}]\)
(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 : \([\text{moment}_\text{of}_\text{inertia}, \text{mass}_\text{per}_\text{vol}, \text{mass}_\text{per}_\text{area}, \text{mass}_\text{per}_\text{length}]\)
(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 pagrm.
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-\(\alpha sin(\))
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_length]

(try moment_of_inertia)
(try mass_per_vol)
(try mass_per_area)

Trying to apply strategy(mass_per_area, situation(pasrm))

Equation-3 : \( m = (b - (-b)) \times (a/\sin(\alpha) - (-a)/\sin(\alpha)) \times \text{mass}_{\text{per}_\text{area}} \)

formed by applying strategy(mass_per_area, situation(pasrm))

This equation solves for \( \text{mass}_{\text{per}_\text{area}} \).

[ No unknowns ]  Do you accept this equation? Yes,

So now I must solve for []

given [mass_per_area, mass1, k, a, b, alpha, m]

Equations extracted:

\[
\begin{align*}
\text{mass1} &= (a \cdot \sin(\alpha) \cdot -1 + y1 \cdot \tan(\alpha) \cdot -1 + \tan(\alpha) \cdot y1 \cdot -1) \\
\text{mass} &= (b - (-b)) \times (a/\sin(\alpha) - (-a)/\sin(\alpha)) \times \text{mass}_{\text{per}_\text{area}}
\end{align*}
\]

Yes

? core 93696 (64512 lo-ses + 29184 hi-ses)
heap 59392 = 57587 in use + 1805 free
local 1024 = 16 in use + 1008 free
trail 511 = 0 in use + 511 free

0.02 sec. for 1 GCs saving 353 words
0.33 sec. for 16 local shifts and 28 trail shifts
10.90 sec. runtime
Problem from file: mofi5,Prb

Radius of Gyrat i on of a ring

Let ringl 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 rofs) was used in a rad_of_smr definition (3)

mofi5 problem read into data base.

yes
! ?- so.

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(ringl, axis(z)))
Let pointl be a new point

Pointl is new fibre defined by
ise(pointl,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(ringl, axis(z)))
Let mass_per_lengthl be the mass_per_length of ringl.
Note: mass_per_lengthl (of type mass) was used in a mass_per_length definition (2)

Equation-1: m*k^2=integrate(d(theta1)*mass_per_lengthl*{((a^2)"number1",[1],[2])}^2*0,"pi",2)
    formed by applying: strategy(moment_of_inertia,situation(ringl, axis(z)))

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

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

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

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

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

Applicable formulae: [moment_of_inertia,mass_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_length
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 [+] given [mass_per_length1,k,a,m]

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

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

I am unable to solve for +.

I will go back to solve for k again

Equation-1 rejected.

I am unable to solve for k.

no

?- core  83456 (54272 lo-ses + 29184 hi-ses)
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 10 local shifts and 21 trail shifts
             4.90 sec. runtime
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 rofs) was used in a rad_of_gyr definition (3)

mofi6 problem read into data base.

yes
?

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(discl,axis(z)))

Let rinsi be a new ring
Note: r1 (of type length) was used in a radius definition (2)

rinsi is new fibre defined by
| isa(rinsi,rins1) |
| centre(rinsi,origin) |
| radius(rinsi,r1) |
| meets(axis(z),rinsi,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(discl,axis(z)))

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

Equation-1: m*k^2=\int_{r1}^{2a}(mass1*r1^2)dx
formed by applying: strategy(moment_of_inertia,situation(discl,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]
(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^2=lintegrate(mass1*r^2*0^d(r1))
mass1=2*pi*r1*(d(r1)*mass_per_area1)
m=pi*a^2*mass_per_area1
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 rota) was used in a rad_of_str definition (3)

mofi7 problem read into data base.

yes

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: sort(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,sort(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=interate(mass1*(sort(a^2-z1^2)/sort(2))~2,a^2,z1)
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)
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=p1*sort(a^2-z1^2)^2*(d(z1)*mass_per_vol)
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]
siven [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)
(try mass_per_area)
Trying to apply strategy(mass_per_vol,situation(sphere1))

Equation-3 : m=4/3*pi*a^3*mass_per_vol
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 []
siven [mass_per_vol1,mass1,k,a,m]

Equations extracted :

m*k^2=integer(mass1*(sort(a^2-z1^2)/sort(2))~2,~z,a,z)
mass1=p1*sort(a^2-z1^2)^2*(d(z1)*mass_per_vol1)
m=4/3*pi*a^3*mass_per_vol1

yes

0.01 sec. for 1 GCs freeing 312 words
0.17 sec. for 10 local shifts and 15 trail shifts
3.40 sec. runtime
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 rofs) was used in a rad_of_syr definition (3)

mofi8 problem read into data base.

yes

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,axes(z)))

Let rin1 be a new ring
Note: sort(a^2-z1^2) (of type length) was used in a radius definition (2)

rin1 is new fibre defined by
isa(rin1,rin1)
centre(rin1,origin1)
radius(rin1,sorta^2-z1^2))
meets(axis(z),rin1,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,axes(z)))
Let mass1 be the mass of rin1.
Equation-1: \( m \times k^2 = \int \text{mass} \times \sqrt{(a^2-z^2)^2-a^2 \times z^2} \)

This equation solves for \( k \) but introduces \([\text{mass}1]\).

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

So now I must solve for \( [\text{mass}1] \) given \([k,a,m]\).

I am now trying to solve for \( \text{mass}_1 \) without introducing any unknowns.

Applicable formulae: \([\text{moment of inertia}, \text{mass per vol}, \text{mass per area}, \text{mass per length}, \text{resolve}]\)

(try moment_of_inertia)
(try mass_per_vol)
(try \text{mass per area})
(try \text{mass per length})

Trying to apply strategy \( [\text{mass per length}, \text{situation (shell1)]} \)
(try resolve)

No luck - I will now accept unknowns in solving for \( \text{mass}_1 \).

Applicable formulae: \([\text{moment of inertia}, \text{mass per vol}, \text{mass per area}, \text{mass per length}, \text{resolve}]\)

(try moment_of_inertia)
(try mass_per_vol)
(try \text{mass per area})
(try \text{mass per length})

Trying to apply strategy \( [\text{mass per length}, \text{situation (shell1)]} \)

Let \( \text{mass per area}1 \) be the \( \text{mass per area} \) of shell1.

Note: \( \text{mass per area}1 \) (of type mass) was used in a \( \text{mass per area} \) definition (2).

Let \( \text{line sys}1 \) be the \( \text{line sys} \) of ring1 Let \( \text{line sys}2 \) be the \( \text{line sys} \) of ring1.

Equation-2: \( \text{mass}_1 = 2 \times \pi \times \sqrt{(a^2-z^2)^2-(d(z1)^2 \times \text{mass per area}1)} \)

This equation solves for \( \text{mass}_1 \) but introduces \([\text{mass per area}1]\).

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

So now I must solve for \( [\text{mass per area}1] \) given \([\text{mass}1,k,a,m]\).

I am now trying to solve for \( \text{mass per area}1 \) without introducing any unknowns.

Applicable formulae: \([\text{moment of inertia}, \text{mass per vol}, \text{mass per area}, \text{mass per length}, \text{resolve}]\)

(try moment_of_inertia)
(try mass_per_vol)
(try mass_per_area)
(try mass_per_length)

Trying to apply strategy \( [\text{mass per area}, \text{situation (shell1)]} \)

Equation-3: \( m = 4 \times \pi \times a^2 \times \text{mass per area}1 \)

This equation solves for \( \text{mass per area}1 \).

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

So now I must solve for \( \)
Equations extracted:

\[ m * k^2 = \text{integrate}(\text{mass1} * \text{sort}(a^2 - z1^2)^2, -a, a + z) \]
\[ \text{mass1} = 2 * \pi * \text{sort}(a^2 - z1^2) * (d(z1) * \text{mass per areal}) \]
\[ m = 4 * \pi * a^2 * \text{mass per areal} \]

Yes

! ?- core 87552 (58368 lo-ses + 29184 hi-ses)
heap 53248 = 51358 in use + 1890 free
slocal 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
yes
! ?- input(mofi9).

Problem from file: mofi9.prb

Radius of Gyraton 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 rofs) was used in a rad_of_gyr definition (3)

mofi9 problem read into data base.

yes
! ?- so.

** ERROR Type unknown  ( continue after error )

Attempts 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: \([\text{Parallel axes, moment of inertia}]\)

(try \text{Parallel axes})
(try \text{moment of inertia})

Trying to apply strategy(\text{moment of inertia}, \text{situation(cylinder1, axis(z))})

Let \(m_{1}\) be the mass of \(\text{discl}\),

Note: \(m_{1}\) (of type \text{mass}) was used in a mass definition (2)

Equation-1: \(m_{1} \cdot [k \cdot a / \sqrt{2}] \cdot r_{y}\)

formed by applying : strategy(\text{moment of inertia}, \text{situation(cylinder1, axis(z))})

This equation solves for \(k\) but introduces \([m_{1}]\).

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

So now I must solve for \([m_{1}]\)
given \([k, a, h, m]\)

I am now trying to solve for \(m_{1}\) without introducing any unknowns.

Applicable formulae: \([\text{moment of inertia, mass per vol, mass per area, mass per length}]\)

(try \text{moment of inertia})
(try \text{mass per vol})
(try \text{mass per area})
Trying to apply strategy(mass per area, situation(discl))
(try \text{mass per length})
(try resolve)

No luck - I will now accept unknowns in solving for \(m_{1}\).

Applicable formulae: \([\text{moment of inertia, mass per vol, mass per area, mass per length}]\)

(try \text{moment of inertia})
(try \text{mass per vol})
(try \text{mass per area})
Trying to apply strategy(mass per area, situation(discl))

Let \(m_{1}\) be the mass per vol of \(\text{cylinder1}\).

Note: \(m_{1}\) (of type \text{mass}) was used in a mass per vol definition (2)

Equation-2: \(m_{1} = \pi \cdot a^{2} \cdot d(z1) \cdot \text{mass per vol1}\)

formed by applying : strategy(mass per area, situation(discl))

This equation solves for \(m_{1}\) but introduces \([\text{mass per vol1}]\).

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

So now I must solve for \([\text{mass per vol1}]\)
given \([m_{1}, k, a, h, m]\)

I am now trying to solve for mass per vol1 without introducing any unknowns.

Applicable formulae: \([\text{moment of inertia, mass per vol, mass per area, mass per length}]\)

(try \text{moment of inertia})
(try \text{mass per vol})
Trying to apply strategy(mass per vol, situation(cylinder1))

Equation-3: \(m = \pi \cdot a^{2} \cdot h \cdot \text{mass per vol1}\)

formed by applying : strategy(mass per vol, situation(cylinder1))

This equation solves for \(\text{mass per vol1}\).

[ No unknowns ] Do you accept this equation? yes.
So now I must solve for \[\]
given \([\text{mass\_per\_voll}, \text{mass1}, \text{k}, \text{a}, \text{h}, \text{m}]\)

Equations extracted:
\[
m * k^2 = \text{integrate(mass1*(a/\text{sort}(2))^2, 0, h, z)}
\]
\[
\text{mass1} = \pi * a^2 * (d(z1) * \text{mass\_per\_voll})
\]
\[
m = \pi * a^2 * h * \text{mass\_per\_voll}
\]

yes
! ?- core 87552 (58368 lo-ses + 29184 hi-ses)
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
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 rofs) was used in a rad_of_gyr definition (3)

mofi10 problem read into data base.

yes  
  : ?- so.

** ERROR Type unknown ( continue after error )  

Attempts 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(disc1,disc)  
	centre(disc1,origin1)  
	radius(disc1,a*(h-z1)/h)  

Attemps to solve for [k] in terms of [a,h,m]
No luck - I will now accept unknowns in solving for \( k \).

**Applicable formulae:** 
- \([\text{parallel_axes}, \text{moment_of_inertia}]\)  
- \([\text{try parallel_axes}])\)  
- \([\text{try moment_of_inertia}])\)  

Trying to apply strategy\((\text{moment_of_inertia} \rightarrow \text{situation(cone, axis(z))})\)  

Let \( m_{\text{mass1}} \) be the mass of \( \text{discl} \).  

Note: \( m_{\text{mass1}} \) (of type mass) was used in a mass definition (2)  

**Equation-1:**  
\[
m_{\text{mass1}} = \int \left( m_{\text{mass1}} \right) \left( a \cdot h - z_1 \right) / h \sqrt{c_2} \right) \, dz \]  

formed by applying : strategy\((\text{moment_of_inertia} \rightarrow \text{situation(cone, axis(z))})\)  

This equation solves for \( k \) but introduces \([m_{\text{mass1}}]\).  

[**Unknowns allowed**] Do you accept this equation? Yes.  

So now I must solve for \([m_{\text{mass1}}]\)  
given \([k, a, h, m]\)  

I am now trying to solve for \( m_{\text{mass1}} \) without introducing any unknowns.  

**Applicable formulae:** 
- \([\text{moment_of_inertia}, \text{mass_per_vol}, \text{mass_per_area}, \text{mass_per_length, resolve}]\)  
- \([\text{try moment_of_inertia}])\)  
- \([\text{try mass_per_vol}])\)  
- \([\text{try mass_per_area}])\)  
- \([\text{try mass_per_length}]\)  
- \([\text{try resolve}])\)  

No luck - I will now accept unknowns in solving for \( m_{\text{mass1}} \).  

**Applicable formulae:** 
- \([\text{moment_of_inertia}, \text{mass_per_vol}, \text{mass_per_area}, \text{mass_per_length, resolve}]\)  
- \([\text{try moment_of_inertia}])\)  
- \([\text{try mass_per_vol}])\)  
- \([\text{try mass_per_area}])\)  
- \([\text{try mass_per_length}]\)  
- \([\text{try resolve}])\)  

Let \( m_{\text{mass_per_vol1}} \) be the mass\(_{\text{mass_per_vol1}} \) of cone1.  

Note: \( m_{\text{mass_per_vol1}} \) (of type mass) was used in a mass\(_{\text{mass_per_vol1}} \) definition (2)  

**Equation-2:**  
\[
m_{\text{mass1}} = \pi \cdot (a \cdot h - z_1) / h \sqrt{c_2} \cdot d(z_1) \cdot m_{\text{mass_per_vol1}} \]  

formed by applying : strategy\((\text{mass_per_area} \rightarrow \text{situation(discl)})\)  

This equation solves for \( m_{\text{mass1}} \) but introduces \([m_{\text{mass_per_vol1}}]\).  

[**Unknowns allowed**] Do you accept this equation? Yes.  

So now I must solve for \([m_{\text{mass_per_vol1}}]\)  
given \([m_{\text{mass1}}, k, a, h, m]\)  

I am now trying to solve for \( m_{\text{mass_per_vol1}} \) without introducing any unknowns.  

**Applicable formulae:** 
- \([\text{moment_of_inertia}, \text{mass_per_vol}, \text{mass_per_area}, \text{mass_per_length, resolve}]\)  
- \([\text{try moment_of_inertia}])\)  
- \([\text{try mass_per_vol}])\)  
- \([\text{try mass_per_area}])\)  

**Equation-3:**  
\[
m_{\text{mass1}} = \frac{1}{3} \pi \cdot a^2 \cdot h \cdot m_{\text{mass_per_vol1}} \]
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:
  \[ m^2 = \int \frac{\pi a (h - z)^2}{h} dz \]
  \[ \text{mass1} = \frac{\pi a (h - z)}{h} \]
  \[ \text{m} = \frac{1}{3} \pi a^2 h \text{mass_per_vol1} \]

Yes

! 7-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
Problem from file: mofill.prb

Radius of Gyration of a tube

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

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

mofill problem read into data base.

yes

** 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(tubel,axis(z)))
oisin andorisinl are separated by [zi,0,90] during always.
Let rinsl be a new ring

rinsl is new fibre defined by
   isa(rings,rinsl)
   centre(rinsl,orisinl)
   radius(rinsl,a)
   meets(axis(z),rinsl,orisinl)

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(tubel,axis(z)))
Let massl be the mass of rinsl.
Note: massl (of type mass) was used in a mass definition (2)

Equation-1 : m*k^2=integrate(massl*2+0+h,zi)
   formed by applying : strategy(moment_of_inertia,situation(tubel,axis(z)))

This equation solves for k but introduces [massl].

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

So now I must solve for [massl]
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_area be the mass_per_area of tube1.

Note: mass_per_area (of type mass) was used in a mass_per_area definition.

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_area
formed by applying: strategy(mass_per_length,situation(ring1))

This equation solves for mass1 but introduces [mass_per_area],

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

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

I am now trying to solve for mass_per_area 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_area
formed by applying: strategy(mass_per_area,situation(tube1))

This equation solves for mass_per_area.

[ 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:
mk^2 = integrate(mass1*a^2,0,h,z1)
mass1=2*pi*a*(d(z1))*mass_per_area
m=2*pi*a*h*mass_per_area

yes
:- core  88064 (58880 lo-ses + 29184 hi-ses)
<table>
<thead>
<tr>
<th>Memory Type</th>
<th>Size 1</th>
<th>Size 2</th>
<th>Size 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>heap</td>
<td>53760</td>
<td>51520</td>
<td>2240</td>
</tr>
<tr>
<td>global</td>
<td>1187</td>
<td>16</td>
<td>1171</td>
</tr>
<tr>
<td>local</td>
<td>1024</td>
<td>16</td>
<td>1008</td>
</tr>
<tr>
<td>trail</td>
<td>511</td>
<td>0</td>
<td>511</td>
</tr>
</tbody>
</table>

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)
       cush, % Current hacks
   ]).
/
* 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 : Formulæ et al for the Moment of Inertia problems */

{ problem_solving_rules },
----------------------------------

% Moment of Inertia
relates(moment_of_inertia,[mass,rofs]),

prepare(moment_of_inertia,Q,rofs,rad_of_syr(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_syr(Obj,Axis,RG,Per) &
    mass(Fibre,Mf,Per) &
    rad_of_syr(Fibre,Axis,RF,Rf,Per) &
    cont_meas(Obj,X,_,Fibre,A,B),

% Parallel Axes
relates(parallel_axes,[rofs]),

prepare(parallel_axes,Q,rofs,rad_of_syr(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_syr(Obj,RealAxis,RG,Per) &
    rad_of_syr(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

\[
\text{length(Line,2*}R,\text{Per)} \\
\quad \leftarrow \text{line(Line)} \land \\
\quad \text{radius(Line,R)}.
\]

% The length of a ring based on its radius

\[
\text{length(Ring,2*}\pi*R,\text{Per)} \\
\quad \leftarrow \text{ring(Ring)} \land \\
\quad \text{radius(Ring,R)}.
\]

% Hack rule for the area of a 1D fibre

\[
\text{area(Fibre,L*d(X),Per)} \\
\quad \leftarrow \text{bodyId(Fibre)} \land \\
\quad \text{length(Fibre,L,Per)} \land \\
\quad \text{cont_meas(Body,X,Origin,Fibre,A,B)}.
\]

% The area of a square

\[
\text{area(Square,(2*A)^2,Per)} \\
\quad \leftarrow \text{square(Square)} \land \\
\quad \text{radius(Square,A)}.
\]

% The area of a disc

\[
\text{area(Disc,}\pi*R^2,\text{Per)} \\
\quad \leftarrow \text{disc(Disc)} \land \\
\quad \text{radius(Disc,R)}.
\]
/* 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)
  \iff % 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)
  \iff % 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_syr(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

arsstruct(cont_meas, 6,
  [foo, foo, foo, foo, foo, foo],
  [ars, ars, ars, ars, ars, ars]).
/* MK1 : Meta level knowledge for the Moment of inertia Problems */

{ meta_knowledge },

%------------------

arsstruct(area,3,
[object,area,time],
[ars,val,ars]),

arsstruct(basetype,2,
[type,object],
[ars,ars]),

arsstruct(centre,2,
[object,point],
[ars,val]),

arsstruct(dist_along,4,
[object,point,point,length],
[ars,ars,ars,ars]),

arsstruct(length,3,
[line,length,time],
[ars,val,ars]),

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

arsstruct(mass_per_length,3,
[line,mass,time],
[ars,val,ars]),

arsstruct(meets,3,
[axis,object,point],
[ars,ars,ars]),

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

arsstruct(rad_of_syr,4,
[object,axis,rofs,time],
[ars,ars,val,ars]),

arsstruct(range,3,
[fibre,length,length],
[ars,ars,ars]),

arsstruct(regular_fibre,2,
[object,fibre],
[ars,ars]),

arsstruct(perp_dist,4,
[object,object,length,time],
[ars,ars,val,ars]).
/* TYP1 : Part of type hierarchy for Moments of Inertia problems */

{ types },
%---------

body0d -> object,
body1d -> object,
body2d -> object,
body3d -> object,

point -> body0d,
line -> body1d,
rings -> 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,
rofs -> scalar,
{ normal_form },
\¾------------------

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),
   centre(Obj,C),
   radius(Obj,R),
   type(Shape,Obj),
   find_fibre_type(Shape,Fshape,Froz),
   find_fibre_range(Froz,R,L1,L2),
   sensym(fibre,Fibre),
   sensym(d,D),
   trace('New 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,Eans),
       freeze(Eans,Froz),
       non_weird(Froz),
       lowerdim(Dim1,Dim),
       shape_defn(Fibre,Dim,Eans),
       match_eans(Eans,Froz).

% Freeze a dimension in the equations

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

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

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

non_weird(Eans)
    :- sel(Eans,frozen(C,Polar(X))),
    fail.

non_weird(_).

% Lower the dimension by one
lowerdim(D1,D),
\( \text{\textbf{\#} D1 > 0,} \)
\( D \text{ is } D1 - 1. \)

% Matching two equational definitions

\text{match_eans(A\&Rest1,B\&Rest2)}
\( \text{\textbf{\#} !,} \)
\( \text{match(A,B),} \)
\( \text{match_eans(Rest1,Rest2).} \)
\( \text{match_eans(A,B) \textbf{\#} match(A,B).} \)

% Matching two coordinates

\text{match(X,X) \textbf{\#} !.} 
\text{match(X,Y) \textbf{\#} match2(X,Y), !.} 
\text{match(X,Y) \textbf{\#} match2(Y,X), !.} 
\text{match2(constant(C),frozen(C,_)).} 
\text{match2(constant(C),cartesian(C=_)).} 
\text{match2(cartesian(C=_),frozen(C,_)).}

% Find the range over which the fibre can vary

\text{find_fibre_range(Eans,RADIUS,LI,L2)}
\( \text{\textbf{\#} sel(Eans,A),} \)
\( \text{ffr(A,RADIUS,LI,L2),} \)
\( !. \)

\text{ffr(frozen(C,EaDefn), RADIUS, LI, L2)}
\( \text{\textbf{\#} set_ineq(EaDefn,E1,C,E2),} \)
\( \text{fr(E1,RADIUS,LI);} \)
\( \text{fr(E2,RADIUS,L2),} \)

\text{fr(-r,RADIUS,-Radius) \textbf{\#} !,} 
\text{fr(r,RADIUS,Radius) \textbf{\#} !,} 
\text{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 (Orisin) will be such that:

centre(Object,Orisin)
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,Ens2)
    !- construction(Object,Dim,Constr),
Point_defn(Eans1),
perform(Constr,0,Eans1,Eans2).

perform([],Rcount,Eans1,Eans2)
!- correct(Rcount,Eans1,Eans2).

perform([OP|Rest],Rc1,Eans1,Eans3).
!- add(OP,Eans1,Eans2),
rchk(OP,Rc1,Rc2),
perform(Rest,Rc2,Eans2,Eans3).

add(OP,Eans1,Eans2)
!- sel(Eans1,E1,E2,Eans2),
operation(OP,E1,E2),
!.

% Routines to correct for rotations
rchk(rotate,N,N1) !- !, N1 is N + 1.
rchk(_,N,N).
correct(0,Eans,Eans) !- !.
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)
    :- seesing(Old),
       see(File),
       repeat,
       read(X),
       shove(X),
       !,
       seen,
       see(Old),
       ttyn1, display(File), display(‘ loaded.’), ttyn1.

% 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), typeftrreeu(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,assertz).

    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,_,Eans),
       writeln(’\n%t\n%c’, [Object,Eans]),
       fail.

dtest(_).

% Test find_fibre_type

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

ftest(_).
s :- save('scraffoo'),
r :- restore(foo),
problem(mofii1,'Radius of Gyration of a line',[]).

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,rs1,now).

given(a).
siven(m).
sousht(rs1).

Older problems & traces
/* MOFI2.PRBB : 2nd Moments of Inertia Problem */

problem(mofi2,'Radius of Gyration of a square

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,rs1,now),
given(a),
given(m),
sought(rs1),
/* MOFI4,PRB : 4th Moments of Inertia problem */

problem(mofi4,'Radius of Gyration of a Disc

period(now).

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

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

rad_of_syr(disc1,ax,rs1,now),
given(a),
given(m),
sought(rs1).
% How to construct various objects

construction(point,0,[]),
construction(line,1,[translate]),
construction(circle,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,translate]),
construction(sphere,3,[translate,rotate,rotate]),
construction(cylinder,3,[translate,translate,rotate,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 ) ),
restrict( cartesian( -R <= C <= R ) ),
restrict( polar( -pi <= C <= pi ) ),

% Freeze a dimension in the equations

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

friz(EaDefn,frozen(C,EaDefn))
   :- sel_ineq(EaDefn,-,C,-).

% Matching two coordinates

match(X,X),
match(constant(C),frozen(C,-)),
match(constant(C),cartesian(C=-)),
match(cartesian(C=__),frozen(C,-)).
Problem from file: mofil.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: rs1 (of type rofs) was used in a rad_of_syr definition (3)

mofil problem read into data base.

yes

I am now trying to solve for rs1 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 rs1.

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_syr1 be the rad_of_syr of fibre2.
Note: rad_of_syr1 (of type rofs) was used in a rad_of_syr definition (3)

Equation-1: mass1*rs1^2=integrate(mass2*rad_of_syr1^2,-a,a,d2)
formed by applying: strategy(moment_of_inertia,situation(l1,ax,fibre2,now))

This equation solves for rs1 but introduces [mass1,mass2,rad_of_syr1].

[ Unknowns allowed ] Do you accept this equation ? yes.
So now I must solve for \([\text{mass1, mass2, rad_of_gyr1}]\)
\[\text{given [rsl, a, m]}\]

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

Applicable formulae : \([\text{mass_per_area, mass_per_length, moment_of_inertia, resol (try mass_per_area)}]\)
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 \([\text{mass2, rad_of_gyr1}]\)
\[\text{given [mass1, rsl, a, m]}\]

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

Applicable formulae : \([\text{mass_per_area, mass_per_length, moment_of_inertia, resol (try mass_per_area)}]\)
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 : \([\text{mass_per_area, mass_per_length, moment_of_inertia, resol (try mass_per_area)}]\)
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 (1)
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 \([\text{area1, mass_per_area1}]\).

[ 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 \([\text{length1}]\).
[Unknowns allowed]  Do you accept this equation? yes.

So now I must solve for [rad_of_syr1,length1]
   given [mass2,mass1,rs1,a,m]

I am now trying to solve for rad_of_syr1 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_syr1.

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

I am unable to solve for rad_of_syr1.

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 rs1 again

Equation-1 rejected.

   (try  parallel_axes)

I am unable to solve for rs1.

no
! ?- restore(alan).

yes
! ?- input(mof2).

Problem from file: mof2.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: rs1 (of type rofs) was used in a rad_of_syr definition (3)

mof2 problem read into data base.

yes
! ?- go.

Attempting to solve for [rs1] in terms of [a,m]
I am now trying to solve for rsl1 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 rsl1.

**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_syr1 be the rad_of_syr of fibre2.
Note: rad_of_syr1 (of type rofs) was used in a rad_of_syr definition (3)

**Equation-1 :** mass1*rsl1^2=integrate(mass2*rad_of_syr1^2,-a,a,d2)
formed by applying : strategy(moment_of_inertia,situation(sq,ax,fibre2,now))

This equation solves for rsl1 but introduces [mass1,mass2,rad_of_syr1].

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

So now I must solve for [mass1,mass2,rad_of_syr1]
given [rsl1,ax,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_syr1]
given [mass1,rsl1,ax,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*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_syr1, radius1]
siven [mass2, mass1, rs1, a, m]

i am now trying to solve for rad_of_syr1 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_syr1.

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

I am unable to solve for rad_of_syr1.

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 rs1 again
Equation 1 rejected.
(try \texttt{parallel\_axes})

I am unable to solve for \texttt{r31}.

no
! \texttt{?- halt.}
Problem from file: mof14.prb
Radius of Gyration of a Disc

Let now be a new period
let disc1 be a new disc
Note: a (of tyre length) was used in a radius definition (2)
Note: m (of tyre mass) was used in a mass definition (2)
let ax be a new axis
Note: r1 (of tyre rofs) was used in a rad_of_gyr definition (3)

mof14 problem read into data base.

yes
1 ?- go.

Attempting to solve for [rs1] 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 ring fibre created, fibre1, for disc disc1.
Let fibre1 be a new ring
Trys to apply strategy(moment_of_inertia,situation(disc1,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 ring fibre created, fibre2, for disc disc1.
Let fibre2 be a new ring
Trys to apply strategy(moment_of_inertia,situation(disc1,ax,fibre2,now))
Let mass1 be the mass of fibre2.
Note: mass1 (of tyre mass) was used in a mass definition (2)

Equation-1: m*r1^2=integrate(mass1*(d2^2)*a^2,0,a,d2)
formed by applying : strategy(moment_of_inertia,situation(disc1,ax,fibre2,now))

This equation solves for rs1 but introduces [mass1].

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

So now I must solve for [mass1]
given [rs1,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)
Trys to apply strategy(mass_per_area,situation(fibre2,now))
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, rs1, 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, rs1, a, m]

Equations extracted:
  m*rs1^2=integrate(mass1*(d2^2)*2,0,a,d2)
  mass1=2*pi*d2*d(d2)*mass_per_area1
  a=pi*a^2*mass_per_area1

yes
! ?- halt.
/* FORMUL : Formulae et al for the Moment of inertia problems */

{ problem Solving rules },
%-----------------------------------
relates(moment_of_inertia,[mass,rofs]).

prepare(moment_of_inertia,Q,rofs,rad_of_syr(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_syr(Obj,Axis,RG,Per) &
               mass(Fibre,M,Per) &
               rad_of_syr(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 },

args struct(basetype,2,
    [type,object],
    [ars,ars]),

args struct(centre,2,
    [object,Point],
    [ars,val]),

args struct(dist_along,4,
    [object,Point,Point,length],
    [ars,ars,ars,ars]),

args struct(length,3,
    [line,length,time],
    [ars,val,ars]),

args struct(mass_per_length,3,
    [line,mass,time],
    [ars,val,ars]),

args struct(meets,3,
    [axis,object,Point],
    [ars,ars,ars]),

args struct(radius,2,
    [object,length],
    [ars,val]),

args struct(rad_of_syr,4,
    [object,axis,rofs,time],
    [ars,ars,val,ars]),

args struct(range,3,
    [fibre,length,length],
    [ars,ars,ars]),

args struct(regular_fibre,2,
    [object,fibre],
    [ars,ars]),

args struct(perp_dist,4,
    [object,object,length,time],
    [ars,ars,val,ars]),

defn( regular_fibre(Obj,Fibre), heeb,
    [ 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, Mass, Per) <- bodyid(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 infinitesimal length
% derived from its distance from the centre.

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

% 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).

% Since in the current rep 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 },
% The radius of gyration of an object may be known
% on the basis of its type. For fibres this means
% their base type.

\texttt{rad_of_gyr(X,Axis,RG,Per)}
  \texttt{<= cc perp_dist(X,Axis,D,Per),}
  \texttt{type(Type,X),}
  \texttt{known_rofs(Type,X,D,RG),}

\texttt{known_rofs(point,_,D,D),}

\texttt{known_rofs(fibre,F,D,Ans)}
  \texttt{<= cc basetype(Btype,F),}
  \texttt{known_rofs(Btype,F,D,Ans).}
/* MKINF2 : Additional meta knowledge and inference rules for dealing with 2D objects. */

{ meta_knowledge }, %-----------------------

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

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

{ 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),
       !,
       ?(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),
       sensym(fibre,Fibre),
       sensym(d,D),
       trace('\nNew %t fibre created, %t, for %t %t,\n',
              [Fshape,Fibre,Shape,Obj], 2),
       (add fibre(Fibre) &
        resfib(Obj,Fibre) &
        basetype(Shape,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(_),

% '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,Eans),
     freeze(Eans,Froz),
     lowerdim(Dim1,Dim),
     shape_defn(Fibre,Dim,Feans),
     match_eans(Feans,Froz).

% Freeze a dimension in the equations

freeze(Eans,Froz)
  :- sel(Eans,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_eans(A&Rest1,B&Rest2)
  :- !,
     match(A,B),


match_eons(Rest1, Rest2).
match_eons(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(Eons, Radius, L1, L2)
  :- sel(Eons, 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).
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)

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)

yes
; ?- halt
; .
/* TEST : Random test routines */

% Test shape_defn

dtest(Object)
   :- shape_defn(Object,_,Eans),
      writef(\n%t\n%c,[Object,Eans]),
      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( <inea> )
  ; polar( <inea> ).

<inea> --> <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 (Orisin) will be such that:

centre(Obj ect,Orisin)

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

radius(Obj ect,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,transla te,translate]),
construction(sphere,3,[translate,rotate,rotate]),
construction(cylinder,3,[translate,translate,rotate]).

% Definition of the shape of an object

shape_defn(Object,Dim,Means2)
  ; - construction(Object,Dim,Constr),
Point_defn(Eans1),
    Perform(Constr,0,Eans1,Eans2).

Perform([],Rcount,Eans1,Eans2)
  :- correct(Rcount,Eans1,Eans2).

Perform([OP:Rest],Rc1,Eans1,Eans3)
  :- add(OP,Eans1,Eans2),
     rchk(OP,Rc1,Rc2),
     Perform(Rest,Rc2,Eans2,Eans3).

add(OP,Eans1,Eans2)
  :- sel(Eans1,E1,E2,Eans2),
     operation(OP,E1,E2),
     !.

% Routines to correct for rotations
rchk(rotate,N,N1) :- !, N1 is N + 1.
rchk(_,N,N).
correct(0,Eans,Eans) :- !.
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).
/* 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_syr(l1,ax,rs1,now),
given(a),
given(m),
sought(rs1).

/* 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_syr(sq,ax,rs1,now),
given(a),
given(m),
sought(rs1).

/* MOFI3.PRB : 3rd Moments of Inertia problem */

Problem(mofi3,'Radius of Gyration of a ring\n\n',[]).

Period(now),

rings(rns),
centre(rns,midpt),
radius(rns,a),
mass_per_length(rns,m,now),
axis(ax),
meets(ax, rns, midpt),

rad_of_syr(rns, ax, rs1, now),

given(a),
given(m),
sought(rs1),
% 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

\texttt{point\_defn(}\ \texttt{constant}(x) \& \texttt{constant}(y) \& \texttt{constant}(z)\texttt{)}.

\texttt{operation}(\texttt{displace}, \texttt{constant}(C), \texttt{cartesian}(C = r \& C = -r)).

\texttt{operation}(\texttt{translate}, \texttt{constant}(C), \texttt{cartesian}(C = r \& C = -r)).

\texttt{operation}(\texttt{rotate}, \texttt{constant}(C), \texttt{polar}(C = \pi)).

\texttt{restrict}(\texttt{cartesian}(A \& \texttt{cartesian}(B)), \texttt{cartesian}(A)).

\texttt{restrict}(\texttt{cartesian}(C = r \& C = R), \texttt{cartesian}(C = r \& C = R)).

\texttt{restrict}(\texttt{polar}(C = \pi)), \texttt{polar}(C = \pi)).

% Freeze a dimension in the equations

\texttt{freeze(Eqndef,Froz)}
  \quad \texttt{?- sel(Eqndef,A,B,Froz),}
  \quad \texttt{friz(A,B)}.

\texttt{friz(Eqndef,frozen(C,Eqndef))}
  \quad \texttt{?- set\_ineq(Eqndef,\_C,\_)}.

% Matching two coordinates

\texttt{match(X,X)}.

\texttt{match(constant(C),frozen(C,\_))}.

\texttt{match(constant(C),cartesian(C=\_))}.

\texttt{match(cartesian(C=\_),frozen(C,\_))}.  

handout at first
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(-π≤z≤π)

cylinder

cartesian(-r≤x≤r)
cartesian(0≤y≤r)
polar(-π≤z≤π)

cylinder

cartesian(-r≤x≤r)
cartesian(-r≤y≤r)
polar(0≤z≤π)

yes
f test(_). % Finding fibres by freezing and trying to find match

Fibres for a point

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

ing

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

ing

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
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.