~~JUNK~~
Directory Listing

MISCELLANEOUS

JUNK

PROBS

METHODS

AXIOMS

PACKAGES

```
;; PRESS.SUB : The Press System
;;
;;                                                                    Updat
;;
;;          ** marks files not included in FILIN
;;
;;          001   -   30 May 82        (before current organis
;; >>

press:press.sub             ;; **  This file

press:solve.                ; TOP LEVEL
press:simeq.
press:sim.
press:ineq.
press:ident.                ;  **

press:isolat.               ; METHODS
press:poly.
press:chunk.
press:collec.
press:attrac.
press:trig.fac
press:nasl.
press:homog.top
press:homog.trs
press:log.
press:nasty.

press:simp.ax               ; AXIOMS
press:isolat.ax
press:ineqis.ax
press:collec.ax
press:attrac.ax
press:homog.rew
press:facts.
press:init.
press:init.mec              ;  **

press:match.                ; PACKAGES
press:int.
press:diff.
press:polpak.
press:polyis.
press:weaknf.
press:prover.
press:sprint.               ;  **

press:words.                ; MISCELLANEOUS
press:sportr.
press:misc.
press:odds.
press:homog.msc

press:runex.                ; PROBLEMS
press:demo.
press:goals                 ;; **
press:mecho.prb             ;; **
press:lewis.prb             ;; **
press:testex.prb            ;; **
```

```
press:exam.                  ;; **
press:exam.inq               ;; **
press:score.                 ;; **
press:failed.                ;; **
press:fixed                  ;; **
press:probs.tid              ;; **

press:facile.                ; JUNK
press:press.ops              ;; **
press:press.mic              ;; **
press:press.def              ;; **
press:filin.                 ;; **
```

```
Prolog ToolKit version 1 (7 December 82)

| ?- count.
Next file: filin
press:chunk.          8 clauses   5 predicates.
press:collec.        28 clauses  13 predicates.
press:attrac.        10 clauses   5 predicates.
press:simp.ax        32 clauses   1 predicates.
press:match.         26 clauses   9 predicates.
press:int.          152 clauses  52 predicates.
press:diff.          28 clauses   4 predicates.
press:polpak.        93 clauses  41 predicates.
press:poltid.         8 clauses   5 predicates.
press:odds.          19 clauses  13 predicates.
press:weaknf.         7 clauses   3 predicates.
press:words.         17 clauses   7 predicates.
press:sportr.        28 clauses   9 predicates.
press:misc.          56 clauses  27 predicates.
press:solve.         29 clauses  13 predicates.
press:simeq.         12 clauses   8 predicates.
press:sim.           45 clauses  22 predicates.
press:ineq.          13 clauses   8 predicates.
press:isolat.        19 clauses   7 predicates.
press:factor.         5 clauses   3 predicates.
press:poly.          38 clauses  17 predicates.
press:trig.fac       96 clauses  36 predicates.
press:nasl.          14 clauses   3 predicates.
press:homog.top      44 clauses  19 predicates.
press:homog.trs     152 clauses  58 predicates.
press:log.           22 clauses   8 predicates.
press:nasty.        159 clauses  69 predicates.
press:isolat.ax      63 clauses   1 predicates.
press:ineqis.ax      21 clauses   0 predicates.
press:collec.ax      17 clauses   1 predicates.
press:attrac.ax      12 clauses   1 predicates.
press:homog.rew      63 clauses   4 predicates.
press:facts.         10 clauses   7 predicates.
press:init.           8 clauses   8 predicates.
press:prover.        22 clauses   8 predicates.
press:manip.          6 clauses   4 predicates.
press:homog.msc      80 clauses  37 predicates.
press:runex.         16 clauses  13 predicates.
press:demo.          13 clauses  10 predicates.
press:facile.         7 clauses   6 predicates.
mec:top.pl            3 clauses   3 predicates.
filin.             1503 clauses 570 predicates.
Next file: util:util
util:util.ops         0 clauses   0 predicates.
util:arith.ops        0 clauses   0 predicates.
util:files.pl         7 clauses   6 predicates.
util:writef.pl       62 clauses  21 predicates.
util:trace.pl        11 clauses   7 predicates.
util:readin.pl       24 clauses  12 predicates.
util:listro.pl       27 clauses  14 predicates.
util:setrou.pl       18 clauses   8 predicates.
util:applic.pl       20 clauses   8 predicates.
util:multil.pl       12 clauses   7 predicates.
util:flagro.pl        1 clauses   1 predicates.
util:struct.pl       18 clauses   6 predicates.
util:cmisce.pl        4 clauses   3 predicates.
```

```
util:long.pl          261 clauses  83 predicates.
util:tidy.pl           90 clauses  25 predicates.
util:edit.pl            8 clauses   4 predicates.
util:invoca.pl         19 clauses  12 predicates.
util:imisce.pl         14 clauses   8 predicates.
util:util.            597 clauses 226 predicates.
Next file:
Grand total:         2100 clauses 796 predicates.

yes
| ?-
PRESS  (7 Dec 82)

| ?- exam.

[Consulting press:exam]

press:exam consulted   5605 words      2.60 sec.

yes
| ?- aeb(1).

Solving  sec(2 *  x) +  tan(2 *  x) = 3 for  x

Applying substitution
     x * 2 =  x1

   to    :
     sec( x * 2) +  tan( x * 2) = 3

   gives :
     sec( x1) +  tan( x1) = 3


Rewriting equation in terms of  sec( x1)
gives  sec( x1) + ( sec( x1) ^ 2 - 1) ^ (1 / 2) = 3

Substituting     x2 for  sec( x1) gives
  x2 + ( x2 ^ 2 + -1) ^ (1/2) = 3

Trying to isolate  x2 ^ 2 + -1
 in  x2 + ( x2 ^ 2 + -1) ^ (1/2) = 3

     x2 ^ 2 + -1 = ( x2 * -1 + 3) ^ 2
        (by Isolation)

Polynomial  x2 ^ 2 + ( x2 * -1 + 3) ^ 2 * -1 + -1 becomes

 x2 * 6 + -10 when in normal form

Applying substitution
     x2 =  sec( x1)

   to    :
     x2 = (5/3)

   gives :
     sec( x1) = (5/3)
```

```
                              **********************************
                              * PROLOG CROSS REFERENCE LISTING *
                              **********************************

                                PRESS Equation Solving System

   PREDICATE                    FILE           CALLED BY



\=/2                         utility        findtype/2 guess_list/2 reduced_term/3

absent/2                     PRESS:COLLEC.  exp_match1/5 absent/2

absol/2                      PRESS:HOMOG.MSC
                                            rew_rule/5 form/3

ac_decomp/4                  PRESS:MATCH.   decomp/2 ac_decomp/4

ac_op/5                      PRESS:MATCH.   decomp/2 ac_decomp/4 recomp/2 ac_recomp
                                            match/2

ac_recomp/3                  PRESS:MATCH.   recomp/2 ac_recomp/3

action/6                     PRESS:HOMOG.TRG
                                            anaz1/6 hyper_find/6

action1/5                    PRESS:HOMOG.TRG
                                            action/6

acute/1                      PRESS:INT.     <user>

add_angle/10                 PRESS:TRIG.FAC
                                            trigsolve/5 checkpairs/5

add_poly/3                   PRESS:POLPAK.  poly/3 add_poly/3 times_poly/3

add_power/3                  PRESS:POLPAK.  map_add_power/3

additive_angles/3            PRESS:TRIG.FAC
                                            apcheck1/3 additive_angles/3

all_are_contained/2          PRESS:INT.     int_apply/3 all_are_contained/2

allowed_guess/2              PRESS:POLPAK.  guess_list/2

anaz/6                       PRESS:HOMOG.TOP
                                            homog1/8

anaz1/6                      PRESS:HOMOG.TRG
                                            anaz/6

anaz2/3                      PRESS:HOMOG.TRG
                                            anaz1/6

andtodot/2                   PRESS:MISC.    pick_xeqn/4 listsolve/5 maximum/2
```

```
angle/3                 PRESS:INIT.   classify/2

angle_size/4            PRESS:HOMOG.TRG
                                      <user> anaz/6 findangle/3 anaz1/6

angle_size1/4           PRESS:HOMOG.TRG
                                      angle_size/4 angle_size1/4

anti_symmetric/2        PRESS:POLPAK. odd_anti_symmetric/1
                                      even_anti_symmetric/1 anti_symmetric/2

apcheck/4               PRESS:TRIG.FAC
                                      trigmethod/3

apcheck1/3              PRESS:TRIG.FAC
                                      apcheck/4

apcheck2/2              PRESS:TRIG.FAC
                                      apcheck1/3 apcheck2/2

append/3                utility       pick_xeqn/4 collect_ans/3 findrhs/2
                                      anaz/6 nasty_act/5 find_symbols1/4
                                      attract_list/3 strip/3 get_dist/4
                                      symmetric/2 anti_symmetric/2

applicable/3            PRESS:COLLEC. collect/3 attract/3

apply_sim2/6            PRESS:SIM.    sim1/4 apply_sim2/6

arbint/1                PRESS:MISC.   <user>

arctrigf/1              PRESS:NASTY.  trig_nasty/2 expon/2 good_fun/1 pta/1

associative/1           PRESS:FACTS.

assumed_positive/1      PRESS:INT.    make_assumption_positive/1

at_least_occ/3          PRESS:MISC.   at_least_occ/3 least_dom/2

atom_num/1              PRESS:HOMOG.MSC
                                      expcase1/5

attract/3               PRESS:ATTRAC. solve2/4 attract/3

attract_list/3          PRESS:NASTY.  find_attract_list/4 attract_list/3

attrax/3                PRESS:ATTRAC.AX
                                      attract/3

below/2                 PRESS:INT.    disjoint/2

bigger/2                PRESS:PROVER. <user> maximum1/2 smaller/2

binary_to_list/5        PRESS:MISC.   andtodot/2 ortodot/2 binary_to_list/5
                                      least_dom/2

binomial/3              PRESS:POLPAK. poly/3 binomial/3

break/4                 PRESS:HOMOG.MSC
                                      rew_rule/5
```

```
breakup_bnds/3          PRESS:INT.    calc/3 breakup_bnds/3

build_red/4             PRESS:POLPAK. sym_transform/2 build_red/4

bye/0                   PRESS:FACILE. <user>

calc/3                  PRESS:INT.    find_int2/2 limits/5

calc_coeff/3            PRESS:HOMOG.TRG
                                      coeff2/4

cart_prod/5             PRESS:INT.    cartesian_product/4

cartesian_product/4     PRESS:INT.    make_regions/3 cart_prod/5

cc/2                    PRESS:HOMOG.TRG
                                      <user> findtype_trig/2

cch/2                   PRESS:HOMOG.TRG
                                      <user> findtype_hyper/2

change_the_variable/5   PRESS:HOMOG.TOP
                                      homog1/8

changeunknown/3         PRESS:CHUNK.  solve2/4

changevar/4             PRESS:CHUNK.  solve2/4

check_tan/2             PRESS:HOMOG.TRG
                                      taneqn/3 check_tan/2

check_tan1/2            PRESS:HOMOG.TRG
                                      check_tan/2

checklist/2             utility       sim2/6 trigmethod/3 findtype/2 anaz/4
                                      half_angle/5 findtype_trig/2
                                      findtype_hyper/2 action1/5
                                      find_log_base/2 maximum1/2 onetest/2
                                      signed/2

checkpairs/5            PRESS:TRIG.FAC
                                      trigsolve/5 checkpairs/5

checkpt/1               PRESS:NASTY.  attract_list/3 checkpt/1

checkpts/1              PRESS:NASTY.  attract_list/3 checkpts/1

checksin_cos1/1         PRESS:TRIG.FAC
                                      checksin_cos1/1

checktrivial_set/2      PRESS:SIM.    sim1/4 checktrivial_set/2

classify/2              PRESS:INT.    find_simple_int/2

clean_up/2              PRESS:INT.    find_limits/4

closeness/3             PRESS:ATTRAC.

coeff1/4                PRESS:HOMOG.TRG
```

```
                                              cosexp/4

      coeff2/4                PRESS:HOMOG.TRG
                                              tanexp_num/4 tanexp_denom/4

      coeff_exp/3             PRESS:HOMOG.MSC
                                              anaz/6

      coeff_list/2            PRESS:POLPAK. gcd_coeffs/2 coeff_list/2

      collax/3                PRESS:COLLEC.AX
                                              collect/3

      collect/3               PRESS:COLLEC. solve2/4 collect/3

      collect_ans/3           PRESS:INEQ.   findmax/3 collect_ans/3

      collect_intervals/3     PRESS:INT.    interval/3 collect_intervals/3

      collect_multipliers/6   PRESS:TRIG.FAC
                                              unattract_distribute/3
                                              collect_multipliers/6

      com_ass_idn/2           PRESS:MISC.   least_dom/2

      comb/2                  PRESS:INT.    less_than/2 calc/3

      combine/3               PRESS:INT.    gen_combine/3

      commutative/1           PRESS:FACTS.

      compatible/2            PRESS:COLLEC. absent/2 list_compatible/2

      concavity/2             PRESS:INIT.   interval/3

      cond_poly_print/4       PRESS:POLY.   poly_solve/4

      cond_print/3            PRESS:MISC.   process_input/4

      cond_trace/0            PRESS:NASTY.  looping1/1

      constant/2              PRESS:POLPAK. guess_list/2

      contains/2              PRESS:MISC.   <user> pick_xeqn/4 collect/3 attract/3
                                              linear_sin_cos/2 mod_angsize1/4 naslok/
                                              angle_size1/4 suit1/3 root_nasty/2
                                              exp_nasty/2 trig_nasty/2 rew_rule/5
                                              exactly_one_arg/4 filter/4 at_least_occ
                                              least_dom/4 laura1/3 report_subs/2

      convert_functor/8       PRESS:TRIG.FAC
                                              trigsolve/5

      correct/2               PRESS:INT.    split1/3

      correct_cos/5           PRESS:TRIG.FAC
                                              add_angle/10

      correct_cos1/3          PRESS:TRIG.FAC
                                              correct_cos/5
```

```
correct_sin/6           PRESS:TRIG.FAC
                                      add_angle/10

correct_sin1/4          PRESS:TRIG.FAC
                                      correct_sin/6

correspond/4            PRESS:MISC.   find_common/4 correspond/4

correspond1/7           PRESS:SIM.    listsolve1/5 correspond1/7

correspond2/6           PRESS:SIM.    correspond1/7 correspond2/6

corresponding_arguments/4
                        PRESS:MATCH.  collect/3 attract/3
                                      corresponding_arguments/4

cosatt/2                PRESS:NASTY.  nag_rule/3 cosatt/2 tanatt/2

cosecfind/1             PRESS:HOMOG.TRG
                                      <user> anaz2/3

cosechp/3               PRESS:HOMOG.TRG
                                      <user> action1/5

cosecp/3                PRESS:HOMOG.TRG
                                      <user> action1/5

cosexp/4                PRESS:HOMOG.TRG
                                      expcc/4 cosexp/4 sinexp/4

cosfind/1               PRESS:HOMOG.TRG
                                      <user> anaz2/3

coshp/3                 PRESS:HOMOG.TRG
                                      <user> action1/5

cosp/3                  PRESS:HOMOG.TRG
                                      <user> action1/5

cothp/3                 PRESS:HOMOG.TRG
                                      <user> action1/5

cs/2                    PRESS:HOMOG.TRG
                                      <user> findtype_trig/2

csh/2                   PRESS:HOMOG.TRG
                                      <user> findtype_hyper/2

decomp/2                PRESS:MATCH.  collect/3 matchup/3 attract/3 factorise
                                      linear_sin_cos/2 trig_normal_form/3
                                      unattract_distribute/3 multiply_through
                                      prepd/2 mulbag_to_list/2 match/2
                                      exp_distrib/2 mul_distrib/2 weaknf/3

default_interval/1      PRESS:INT.    find_int2/2 find_simple_int/2 clean_up/

delete/3                PRESS:MISC.   delete/3

denorm/2                PRESS:POLPAK. poleval/3
```

```
denorm1/3               PRESS:POLPAK. denorm/2 denorm1/3

derive/7                PRESS:TRIG.FAC
                                      trigsolve/5

diff/2                  utility       anaz2/3 exp_nasty/2

diffwrt/3               PRESS:DIFF.   findmax/3

discriminant/4          PRESS:POLY.   poly_method/4

disguised_linear/1      PRESS:POLY.   poly_method/4

disj_solve/4            PRESS:SOLVE.  solve1/4

disj_solve_list/4       PRESS:SOLVE.  disj_solve/4 disj_solve_list/4

disjoint/2              PRESS:INT.    overlap/2

disjunction/1           PRESS:FACTS.  solve1/4

dist/2                  PRESS:NASTY.  nas_rule/3 multiply_through/4

dist1/2                 PRESS:NASTY.  dist/2 dist1/2

dist_multiply/3         PRESS:TRIG.FAC
                                      unattract_distribute/3 dist_multiply/3

distribute/3            PRESS:SIMEQ.  simsolve1/3 distribute/3 simsolve2/3

div_lin/5               PRESS:POLPAK. factor_out/3 div_lin/5

div_list/4              PRESS:FACTOR. factorise/4 div_list/4

div_power/3             PRESS:POLPAK. map_div_power/3

dl_modparse/3           PRESS:SIM.    modparse/3 dl_modparse/3

dl_parse/3              PRESS:HOMOG.TOP
                                      parse/3 dl_parse/3

dl_parse2/3             PRESS:HOMOG.MSC
                                      parse2/3 dl_parse2/3

dl_parse4/4             PRESS:NASTY.  parse4/4 dl_parse4/4

domult/3                PRESS:NASTY.  mult/3

domult/4                PRESS:NASTY.  domult/3 domult/4

dottoand/2              PRESS:MISC.   pick_xeqn/4 listsolve/5 maximum/2

dottoor/2               PRESS:MISC.   disj_solve/4 solve1/4 dottoor_set/2

dottoor_set/2           PRESS:SIM.    sim1/4

dx/3                    PRESS:DIFF.   diffwrt/3 dx/3

error/3                 utility       find_int2/2
```

```
eval/1                  utility       modcall/1 warn_if_complex/1 add_angle/1
                                      convert_functor/8 correct_sin1/4
                                      correct_cos1/3 expss/4 expsc/4 expcs/4
                                      find_bases1/2 postidy/2 root_nasty/2
                                      exp_nasty/2 least/3 good_fun/1
                                      set_nasty_type/3 expon_exp/3
                                      expon_inv_exp/3 exp_member/4 isolax/4
                                      isolax/4 rew_rule/5 match/2 order/4
                                      less_than_eval/3 poly/3 add_poly/3
                                      denorm1/3 div_lin/5 odd/1 even/1
                                      least_el/2 great_el/2 lessone/1 moreone
                                      neg22/1

eval/2                  utility       remove_neg_powers/4 poly_method/4
                                      trigsolve/5 add_angle/10 sumdiff/10
                                      derive/7 apcheck2/2 checkpairs/5 anaz/6
                                      half_angle/5 expss/4 expsc/4 expcs/4
                                      expcc/4 cosexp/4 coeff1/4 sinexp/4
                                      exptt/4 tanexp_num/4 tanexp_denom/4
                                      calc_coeff/3 find_bases1/2 postidy/2
                                      set_nasty_type/3 neg_exp/3 domult/4
                                      free_mult/3 merge/2 rew_rule/5 match/2
                                      calc/3 poly/3 timesingl/4 binomial/3
                                      denorm1/3 factor_out/3 div_lin/5
                                      poleval1/4 guess_list/2 factors_of/3
                                      sym_transform/2 build_red/4 symmetric/2
                                      anti_symmetric/2 gcd/3 lcm/3
                                      rational_gcd/3 rational_gcd_list/2 ral/
                                      fact/3 half_angle_check1/2 powered/3
                                      absol/2 break/4

even/1                  PRESS:ODDS.   <user> action1/5 check_tan1/2 isolax/4
                                      find_int2/2 even_symmetric/1
                                      even_anti_symmetric/1

even_anti_symmetric/1   PRESS:POLPAK. poly_method/4

even_symmetric/1        PRESS:POLPAK. poly_method/4

exactly_one_arg/3       PRESS:DIFF.   dx/3

exactly_one_arg/4       PRESS:DIFF.   exactly_one_arg/3 exactly_one_arg/4

exp_distrib/2           PRESS:MANIP.  poly/3

exp_distrib_list/3      PRESS:MANIP.  exp_distrib/2 exp_distrib_list/3

exp_match/5             PRESS:COLLEC. matchup/3

exp_match1/5            PRESS:COLLEC. exp_match/5

exp_member/4            PRESS:NASTY.  domult/4 exp_member/4

exp_nasty/2             PRESS:NASTY.  nasty/2 exp_nasty_list/3

exp_nasty_list/3        PRESS:NASTY.  try_nasty_method/3 exp_nasty_list/3

expcase1/5              PRESS:HOMOG.MSC
                                      <user> anaz/6
```

```
expcase2/3             PRESS:HOMOG.MSC
                                     <user> anaz/6

expcc/4                PRESS:HOMOG.TRG
                                     expsc/4 rew_rule/5

expcs/4                PRESS:HOMOG.TRG
                                     expss/4 expcs/4 rew_rule/5

expon/2                PRESS:NASTY.  find_symbols1/4

expon_exp/3            PRESS:NASTY.  nas_rule/3

expon_exp1/3           PRESS:NASTY.  expon_exp/3

expon_inv_exp/3        PRESS:NASTY.  nas_rule/3

expon_inv_exp1/3       PRESS:NASTY.  expon_inv_exp/3

expp/1                 PRESS:HOMOG.TOP
                                     <user> findtype/2

expp1/1                PRESS:HOMOG.TOP
                                     hypexp/1 split_case1/3

expsc/4                PRESS:HOMOG.TRG
                                     expsc/4 rew_rule/5

expss/4                PRESS:HOMOG.TRG
                                     expss/4 expcs/4 rew_rule/5

exptt/4                PRESS:HOMOG.TRG
                                     rew_rule/5

extreme_term/3         PRESS:MISC.   changeunknown/3 reduced_term/3

extreme_term/5         PRESS:MISC.   extreme_term/3 extreme_term/5

fact/2                 PRESS:ODDS.   coeff1/4 calc_coeff/3

fact/3                 PRESS:ODDS.   fact/2 fact/3

fact_solve/4           PRESS:SOLVE.  solve1/4 fact_solve/4

factor_out/3           PRESS:POLPAK. poly_method/4

factorise/4            PRESS:FACTOR. solve1/4

factors_of/3           PRESS:POLPAK. allowed_guess/2 factors_of/3

filter/4               PRESS:WEAKNF. weaknf/3 filter/4

find1/2                PRESS:POLY.   linear_method/2

find2/3                PRESS:POLY.   find_coeffs/4

find_attract_list/4    PRESS:NASTY.  nasty_act/5 find_attract_list/4

find_bases/2           PRESS:LOG.    find_log_base/2
```

```
find_bases1/2           PRESS:LOG.     find_bases/2 find_bases1/2

find_coeffs/4           PRESS:POLY.    poly_method/4

find_common/4           PRESS:HOMOG.TRG
                                       anaz1/6

find_int/2              PRESS:INT.     vet/2 positive/1 negative/1 non_neg/1
                                       non_pos/1 non_zero/1 acute/1 obtuse/1
                                       non_reflex/1 find_int2/2

find_int2/2             PRESS:INT.     find_int/2 find_int2/2 find_int_args/4

find_int_args/3         PRESS:INT.     find_int2/2

find_int_args/4         PRESS:INT.     find_int_args/3 find_int_args/4

find_limits/4           PRESS:INT.     int_apply/3

find_log_base/2         PRESS:LOG.     suitable/3

find_simple_int/2       PRESS:INT.     find_int2/2

find_symbols/4          PRESS:NASTY.   try_nasty_method/3 find_symbols/4

find_symbols1/4         PRESS:NASTY.   find_symbols/4

findangle/3             PRESS:HOMOG.TRG
                                       anaz/6

findbnd/3               PRESS:INEQ.    <user> solveineq/3

findmax/3               PRESS:INEQ.    findbnd/3

findrhs/2               PRESS:NAS1.    findrhs/2

findtype/2              PRESS:HOMOG.TOP
                                       homog1/8

findtype_hyper/2        PRESS:HOMOG.TRG
                                       hyper_find/6

findtype_trig/2         PRESS:HOMOG.TRG
                                       anaz1/6

fixvar/2                undefined      solveineq/3

flag/3                  utility        cond_trace/0

form/3                  PRESS:HOMOG.MSC
                                       anaz/6 findangle/3

form1/3                 PRESS:HOMOG.MSC
                                       anaz/6 form/3

form2/3                 PRESS:HOMOG.MSC
                                       half_angle/5

form4/3                 PRESS:HOMOG.MSC
```

```
                                      anaz/6 coeff1/4 coeff2/4

free_mult/3             PRESS:NASTY.  multiply_through/4 free_mult/3

freeof/2                PRESS:MISC.   solve1/4 safe_divisor/2 linear_sin_cos/
                                      collect_multipliers/6 trigtype/3
                                      mod_angsize1/4 correct_sin/6
                                      correct_cos/5 naglok/4 dl_parse/3
                                      angle_size1/4 suit1/3 dl_parse4/4
                                      rew_rule/5 dx/3 is_poly/2 poly/3
                                      contains/2 freeof/3 expcase1/5 expcase2
                                      laura/4 dl_parse2/3

freeof/3                PRESS:MISC.   freeof/2 freeof/3

frequent_words/2        PRESS:WORDS.  <user>

gcd/3                   PRESS:ODDS.   gcd_poly/3 lcm/3 gcd1/3 rational_gcd/3

gcd_coeffs/2            PRESS:POLPAK. guess_list/2

gcd_list/2              PRESS:ODDS.

gcd_poly/3              PRESS:POLPAK. gcd_powers/2 gcd_poly/3

gcd_powers/2            PRESS:POLPAK. poly_hidden/3

gcd1/3                  PRESS:ODDS.   gcd_list/2 gcd1/3

gen_combine/2           PRESS:INT.    int_apply/3 interval/3

gen_combine/3           PRESS:INT.    gen_combine/2 gen_combine/3

genpolcase/3            PRESS:HOMOG.MSC
                                      <user> anaz/6

genpoly/2               PRESS:HOMOG.TOP
                                      <user> findtype/2

gensym/2                utility       dx/3 arbint/1 identifier/1

get_bnd/3               PRESS:INT.    get_bnds/4

get_bnds/4              PRESS:INT.    limits/5 get_bnds/4

get_coeff/2             PRESS:TRIG.FAC
                                      <user> apcheck/4

get_dist/4              PRESS:NASTY.  prepd1/2 get_dist/4

get_members/3           PRESS:HOMOG.MSC
                                      coeff_exp/3 get_members/3

get_nasty_type/3        PRESS:NASTY.  attract_list/3

get_ops/3               PRESS:NASTY.  pos1/4

givesmes/3              PRESS:INEQ.   <user> findmax/3

go/0                    PRESS:FACILE. <user>
```

```
good_eqn/2              PRESS:SIM.    try_sort/3

good_fun/1              PRESS:NASTY.  nice_list/1

good_subterm/2          PRESS:CHUNK.  good_subterm/4 good_subterm/3

good_subterm/3          PRESS:CHUNK.  good_subterm/2 good_subterm/3

good_subterm/4          PRESS:CHUNK.  <user> changeunknown/3

great_el/2              PRESS:HOMOG.MSC
                                      find_common/4 onetest/2 great_el/2

guess_list/2            PRESS:POLPAK. poly_method/4

half_angle/5            PRESS:HOMOG.TRG
                                      anaz1/6

half_angle_check1/2     PRESS:HOMOG.MSC
                                      <user> half_angle/5

half_angle_check2/2     PRESS:HOMOG.MSC
                                      <user> half_angle/5

homog/6                 PRESS:HOMOG.TOP
                                      solve2/4

homog1/8                PRESS:HOMOG.TOP
                                      sim2/6 homog/6

hyper_find/6            PRESS:HOMOG.TRG
                                      anaz/6

hyperf/1                PRESS:HOMOG.TOP
                                      <user> findtype/2 hypexp/1 split_case1/
                                      dl_parse2/3

hypexp/1                PRESS:HOMOG.TOP
                                      <user> findtype/2

ident/1                 undefined     <user>

ident_operators/2       PRESS:COLLEC. applicable/3

identifier/1            PRESS:MISC.   sim2/6 changevar/4 change_the_variable/

in/2                    PRESS:INT.    <user>

incline/3               PRESS:INIT.   classify/2

initialize_loop_check/0 PRESS:SOLVE.  solve/4

insert_word/3           PRESS:WORDS.  scan_term/3 insert_word/3

int_apply/3             PRESS:INT.    find_int2/2 int_apply_all/3

int_apply_all/3         PRESS:INT.    int_apply/3 int_apply_all/3

integral/1              PRESS:NASTY.  subintegral/2
```

```
intermediate/1          PRESS:INIT.   <user> intermediates_in/2

intermediates_in/2      PRESS:INEQ.   findbnd/3

interval/3              PRESS:INT.    classify/2 collect_intervals/3

inv_trigtype/2          PRESS:TRIG.FAC
                                      trigsolve/5 checkpairs/5

invert/2                PRESS:ISOLAT. maneuver_sides/3

invert1/2               PRESS:ISOLAT. invert/2

is_poly/2               PRESS:POLPAK. solve1/4 is_poly/2 simplify/2

isolate/3               PRESS:ISOLAT. solve1/4 solve2/4 isolate/4 try_isolate

isolate/4               PRESS:POLY.   poly_method/4

isolate1/3              PRESS:ISOLAT. isolate/3 isolate1/3

isolax/4                PRESS:INEQIS.AX


isolax/4                PRESS:ISOLAT.AX
                                      isolate1/3

last/2                  utility       remove_neg_powers/4 poly_method/4
                                      attract_list/3 constant/2

last_equation/1         PRESS:FACILE. show/0 <user> redo/0

laura/4                 PRESS:HOMOG.MSC
                                      <user> anaz/6

laura1/3                PRESS:HOMOG.MSC
                                      <user> anaz/6

lcm/3                   PRESS:ODDS.   lcm1/3

lcm_list/2              PRESS:ODDS.

lcm1/3                  PRESS:ODDS.   lcm_list/2 lcm1/3

least/3                 PRESS:NASTY.  member_match/3

least_dom/2             PRESS:MISC.   collect/3 attract/3

least_dom/4             PRESS:MISC.   least_dom/2 least_dom/4

least_el/2              PRESS:HOMOG.MSC
                                      find_log_base/2 find_bases/2 onetest/2
                                      least_el/2

less_than/2             PRESS:INT.    positive/1 negative/1 non_neg/1 non_pos
                                      non_zero/1 acute/1 obtuse/1 non_reflex/
                                      sub_int/2 below/2 split1/3

less_than_eval/3        PRESS:INT.    less_than/2
```

```
lessone/1               PRESS:HOMOG.MSC
                                      <user> onetest/2

limits/5                PRESS:INT.    find_limits/4

linear/1                PRESS:POLY.   poly_method/4

linear_method/2         PRESS:POLY.   poly_method/4

linear_sin_cos/2        PRESS:TRIG.FAC
                                      solve2/4 linear_sin_cos/2

list_compatible/2       PRESS:COLLEC. compatible/2 list_compatible/2

list_to_binary/3        PRESS:MISC.   dottoand/2 dottoor/2 list_to_binary/3

listsolve/5             PRESS:SIM.    sim1/4 listsolve/5 listsolve1/5

listsolve1/5            PRESS:SIM.    listsolve/5

listtoset/2             utility       modparse/3 dottoor_set/2 factorise/4
                                      parse/3 remove_subsumed/2 onetest/2
                                      parse2/3

logf/1                  PRESS:HOMOG.TOP
                                      <user> findtype/2

logmethod/4             PRESS:LOG.    solve2/4 logmethod/4

logocc/4                PRESS:HOMOG.MSC
                                      anaz/6

looping/2               PRESS:NASTY.  nasty_method/3

looping1/1              PRESS:NASTY.  looping/2

make_arblist/2          PRESS:NASTY.  remove_arbs1/3

make_arblist1/3         PRESS:NASTY.  make_arblist/2 make_arblist1/3

make_assumption_positive/1
                        PRESS:INT.    find_simple_int/2

make_poly/3             PRESS:POLPAK. cond_poly_print/4 remove_neg_powers/4
                                      simplify/3

make_regions/3          PRESS:INT.    int_apply/3

make_subl/3             PRESS:HOMOG.MSC
                                      remove_arbs1/3 rew/5 make_subl/3

makenice/2              PRESS:HOMOG.TRG
                                      anaz1/6

maneuver_sides/3        PRESS:ISOLAT. isolate/3

map_add_power/3         PRESS:POLPAK. remove_neg_powers/4 poly_method/4
                                      map_add_power/3
```

```
map_div_power/3         PRESS:POLPAK. poly_method/4 map_div_power/3

map_reify/3             PRESS:POLPAK. make_poly/3 map_reify/3

mapand/3                utility       solveineq/3

maplist/3               utility       findmax/3 apcheck/4 findtype/2 anaz/6
                                      findangle/3 anaz1/6 find_common/4
                                      action1/5 rew/5

mapmodparse/3           PRESS:SIM.    sim/3 mapmodparse/3

maptrigtype/3           PRESS:TRIG.FAC
                                      trig_normal_form/3 maptrigtype/3

mapunattract_distribute/3
                        PRESS:TRIG.FAC
                                      unattract_distribute/3
                                      mapunattract_distribute/3

marker_flip/2           PRESS:INT.    sub_int/2 split1/3

match/2                 PRESS:MATCH.  cond_poly_print/4 applicable/3
                                      exp_match/5 exp_match1/5 trigtype/3
                                      try_factorize/4 try_factorize/3
                                      mod_angsize1/4 angle_size1/4 anaz1/6
                                      tantype1/2 check_tan1/2 postidy/2
                                      rem_sub/3 member_match/3 expon_exp1/3
                                      expon_inv_exp1/3 neg_exp/3
                                      neg_exp_match/4 sinatt/2 cosatt/2
                                      tanatt/2 trig_inv/3 exp_member/4 merge/
                                      rew_rule/5 match/2 match_arguments/3
                                      cond_print/3 expcase1/5 expcase2/3
                                      break/4

match_arguments/3       PRESS:MATCH.  match/2 match_arguments/3

matchup/3               PRESS:COLLEC. applicable/3

maximum/2               PRESS:PROVER. solveineq/3

maximum1/2              PRESS:PROVER. maximum/2 maximum1/2

measure/2               PRESS:INIT.   find_simple_int/2 classify/2

member/2                utility       trivial_set/2 poly_method/4 checkpairs/
                                      trigf/1 logf/1 hyperf/1 half_angle/5
                                      anaz2/3 arctrigf/1 sinatt/2 cosatt/2
                                      tanatt/2 logocc/4

member_match/3          PRESS:NASTY.  rem_sub/3 member_match/3

memberchk/2             utility       findtype_trig/2 findtype_hyper/2 comb/2

merge/2                 PRESS:NASTY.  sinatt/2 cosatt/2 tanatt/2

min/3                   PRESS:INEQ.

mod_angsize/4           PRESS:TRIG.FAC
                                      trigtype/3
```

```
nod_angsize1/4          PRESS:TRIG.FAC
                                     convert_functor/8 mod_angsize/4
                                     mod_angsize1/4

mod_trace/1             PRESS:ISOLAT. isolate/3

modcall/1               PRESS:ISOLAT. isolate1/3 modcall/1

modparse/3              PRESS:SIM.    mapmodparse/3

mono/3                  PRESS:INT.    int_apply/3

moreone/1               PRESS:HOMOG.MSC
                                     <user> onetest/2

mul_distrib/2           PRESS:MANIP.

mul_distrib_list/3      PRESS:MANIP.  mul_distrib/2 mul_distrib_list/3

mulbag/1                PRESS:FACTS.  solve1/4 exp_distrib/2

mulbag_to_list/2        PRESS:NASTY.  domult/3

mult/3                  PRESS:NASTY.  multiply_through/4 mult/3

mult_occ/2              PRESS:MISC.   collect/3 attract/3 simplify/2

multiple_offenders_set/3
                        PRESS:HOMOG.TOP
                                     solve2/4

multiply_through/4      PRESS:NASTY.  try_nasty_method/3

nas1/3                  PRESS:NAS1.   solve2/4

nas1ok/4                PRESS:NAS1.   nas1/3

nas_rule/3              PRESS:NASTY.  nasty_act/5

nasty/2                 PRESS:NASTY.  subnasty/3

nasty_act/5             PRESS:NASTY.  try_nasty_method/3

nasty_method/3          PRESS:NASTY.  solve2/4

natnum/1                PRESS:ODDS.

neg22/1                 PRESS:HOMOG.MSC
                                     <user> signed/2

neg_exp/3               PRESS:NASTY.  nas_rule/3 neg_exp/3

neg_exp_match/4         PRESS:NASTY.  neg_exp/3

negation/2              PRESS:PROVER. verify/2

negation1/2             PRESS:PROVER. negation/2

negative/1              PRESS:INT.    givesneg/3 isolax/4
```

```
newform/4               PRESS:COLLEC. collect/3 attract/3

newtype/2               PRESS:HOMOG.REW
                              rew/5

nice/1                  PRESS:NASTY.  nasty_act/5 nice/1

nice_list/1             PRESS:NASTY.  nice/1 nice_list/1

nmember/3               utility       nasty_act/5 posl/4

nocc/3                  PRESS:HOMOG.MSC
                              <user> find_common/4

non_add/2               PRESS:TRIG.FAC
                              apcheck1/3 non_add/2

non_neg/1               PRESS:INT.    prove/1

non_pos/1               PRESS:INT.    <user>

non_reflex/1            PRESS:INT.    <user>

non_trivial/1           PRESS:HOMOG.TOP
                              split_case/3 non_trivial/1

non_zero/1              PRESS:INT.    modcall/1 safe_divisor/2 non_zero/1
                                      prove/1

normstore/3             PRESS:NASTY.  looping/2

number/1                utility       good_subterm/2 mod_angsize1/4
                                      correct_sin/6 correct_cos/5 dl_parse/3
                                      genpoly/2 angle_size1/4 angz1/6
                                      find_bases1/2 postidy/2 root_nasty/2
                                      exp_nasty/2 expon/2 rem_sub/3 good_fun/
                                      rew_rule/5 match/2 find_int2/2 poly/3
                                      coeff_list/2 scan_term/3 position/3
                                      term_size/2 expcase1/5 expcase2/3
                                      laura1/3 dl_parse2/3 powered/3 lessone/
                                      moreone/1 break/4

numeric/1               PRESS:HOMOG.MSC
                              atom_num/1

obtuse/1                PRESS:INT.    <user>

occ/3                   utility       changeunknown/3 good_subterm/4
                                      single_occ/2 mult_occ/2 nocc/3

odd/1                   PRESS:ODDS.   trigsolve/5 odd_symmetric/1
                                      odd_anti_symmetric/1

odd_anti_symmetric/1    PRESS:POLPAK. poly_method/4

odd_symmetric/1         PRESS:POLPAK. poly_method/4

oddnum/1                PRESS:ODDS.
```

```
onetest/2               PRESS:HOMOG.MSC
                                      anaz/6

oops/0                  PRESS:FACILE. <user>

oper/3                  PRESS:GPORTR. prin/2

oper/4                  PRESS:GPORTR. oper/3

ops_list/2              PRESS:COLLEC. ops_to_find/2 ops_list/2

ops_to_find/2           PRESS:COLLEC. exp_match1/5 ops_list/2

order/4                 PRESS:INT.    combine/3

ortodot/2               PRESS:MISC.   disj_solve/4 sim1/4

overlap/2               PRESS:INT.    vet/2

parse/3                 PRESS:HOMOG.TOP
                                      dl_modparse/3 multiple_offenders_set/3

parse2/3                PRESS:HOMOG.MSC
                                      action/6 taneqn/3

parse4/4                PRESS:NASTY.  try_nasty_method/3

partition/2             PRESS:INIT.   interval/3

perform_rewrites/6      PRESS:HOMOG.TOP
                                      homog1/8

perm2/4                 utility       invert/2 matchup/3

pick_xeqn/4             PRESS:SIMEQ.  simsolve1/3

plusbag/1               PRESS:FACTS.  mul_distrib/2

poleval/3               PRESS:POLPAK. root/2

poleval1/4              PRESS:POLPAK. poleval/3 poleval1/4

poly/3                  PRESS:POLPAK. poly_norm/3 poly/3

poly_hidden/3           PRESS:POLY.   poly_method/4

poly_method/4           PRESS:POLY.   poly_solve/4 poly_method/4

poly_norm/3             PRESS:POLPAK. good_eqn/2 poly_solve/4 simplify/3

poly_solve/4            PRESS:POLY.   solve1/4 poly_solve/4 poly_method/4

poly_tidy/2             PRESS:POLTID. poly_solve/4 simplify/3 poly_tidy/2

polytype/2              PRESS:SIM.    <user> sim2/6

portray/1               PRESS:GPORTR.

portray_number/1        utility       prin/2
```

```
position/3             PRESS:MISC.    solve1/4 nasty_act/5 position/4

position/4             PRESS:MISC.    position/3 position/4

positive/1             PRESS:INT.     isolax/4 prove/1

posl/4                 PRESS:NASTY.   find_symbols1/4 posl/4

postidy/2              PRESS:LOG.     logmethod/4

powered/3              PRESS:HOMOG.MSC
                                      rew_rule/5

prepd/2                PRESS:NASTY.   dist/2 re_dist/3

prepd1/2               PRESS:NASTY.   prepd/2

prin/2                 PRESS:GPORTR.  portray/1 prin/2 prna/3 printail/1

prin/3                 PRESS:GPORTR.  prin/2 prin/3

print_the_answer/2     PRESS:SOLVE.   process_answer/3

printail/1             PRESS:GPORTR.  prin/2 printail/1

prna/3                 PRESS:GPORTR.  prin/2 prin/3

prnf/3                 PRESS:GPORTR.  prin/2

prnp/4                 PRESS:GPORTR.  prin/2 prnf/3

process_answer/3       PRESS:SOLVE.   solve/4

process_input/4        PRESS:SOLVE.   solve/4

prove/1                PRESS:PROVER.  bigger/2 verify/2

pt/1                   PRESS:NASTY.   pta/1 checkpt/1

pta/1                  PRESS:NASTY.   checkpta/1

quad/4                 PRESS:INT.     interval/3

quadratic/1            PRESS:POLY.    poly_method/4

quantity/1             PRESS:INIT.    find_simple_int/2

rational_gcd/3         PRESS:ODDS.    guess_list/2

rational_gcd_list/2    PRESS:ODDS.    findtype/2 anaz/6 gcd_coeffs/2
                                      coeff_exp/3 form/3

re_dist/3              PRESS:NASTY.   prepd1/2

recomp/2               PRESS:MATCH.   collect/3 newform/4 attract/3
                                      unattract_distribute/3 multiply_through
                                      set_dist/4 domult/4 match/2 make_poly/3
                                      exp_distrib/2 mul_distrib/7

redo/0                 PRESS:FACILE.  <user>
```

```
reduced_term/3          PRESS:HOMOG.MSC
                                      anaz/6

reify/3                 PRESS:POLPAK. map_reify/3

rem_sub/3               PRESS:NASTY.  remove_subsumed/2 rem_sub/3

remove_args/2           PRESS:NASTY.  normstore/3

remove_args1/3          PRESS:NASTY.  remove_args/2

remove_dis_dups/2       undefined     simsolve/3

remove_neg_powers/4     PRESS:POLY.   poly_solve/4

remove_subsumed/2       PRESS:NASTY.  try_nasty_method/3

reorder_eqn/3           PRESS:SIM.    sim1/4

report/0                PRESS:HOMOG.MSC
                                      report_subs/2 report_on/0 report_off/0

report_off/0            PRESS:HOMOG.MSC
                                      <user>

report_on/0             PRESS:HOMOG.MSC
                                      <user>

report_subs/2           PRESS:HOMOG.MSC
                                      change_the_variable/5

report_subs1/1          PRESS:HOMOG.MSC
                                      report_subs/2 report_subs1/1

rew/5                   PRESS:HOMOG.REW
                                      perform_rewrites/6

rew1/5                  PRESS:HOMOG.REW
                                      <user> rew/5 rew1/5

rew_rule/5              PRESS:HOMOG.REW
                                      rew1/5 rew_rule/5

rgl/4                   PRESS:ODDS.   rational_gcd_list/2 rgl/4

root/2                  PRESS:POLPAK. poly_method/4

root_nasty/2            PRESS:NASTY.  nasty/2

roots/6                 PRESS:POLY.   poly_method/4

safe_divisor/2          PRESS:FACTOR. div_list/4

scan_list/3             PRESS:WORDS.  scan_term/3 scan_list/3

scan_term/3             PRESS:WORDS.  wordsin/2 frequent_words/2 scan_list/3

secfind/1               PRESS:HOMOG.TRG
                                      <user> anaz2/3
```

```
sechp/3                 PRESS:HOMOG.TRG
                                      <user> action1/5

secp/3                  PRESS:HOMOG.TRG
                                      <user> action1/5

seen_eqn/1              PRESS:NASTY.  looping1/1

select/3                utility       pick_xeqn/4 matchup/3 exp_match1/5
                                      list_compatible/2

select_letter/2         PRESS:POLTID. simplify/2

show/0                  PRESS:FACILE. bye/0

signed/2                PRESS:HOMOG.MSC
                                      anaz/6

sim/1                   PRESS:SIM.    <user>

sim/2                   PRESS:SIM.    <user>

sim/3                   PRESS:SIM.    <user> sim/1 sim/2

sim1/4                  PRESS:SIM.    sim/3

sim2/6                  PRESS:SIM.    apply_sim2/6

simple/1                undefined     freeof/2

simplify/2              PRESS:POLTID. simplify_ans/2 singleton_method/3
                                      sumdiff/10 poly_tidy/2 prove/1

simplify/3              PRESS:POLTID. simplify/2

simplify_ans/2          PRESS:SOLVE.  process_answer/3 simplify_ans/2

simplify_axiom/2        PRESS:SIMP.AX

simsolve/1              PRESS:SIMEQ.  <user>

simsolve/2              PRESS:SIMEQ.  <user>

simsolve/3              PRESS:SIMEQ.  <user> simsolve/2 simsolve/1 sim1/4

simsolve1/3             PRESS:SIMEQ.  simsolve/3 simsolve2/3 sim1/4

simsolve2/3             PRESS:SIMEQ.  simsolve1/3 simsolve2/3

sinatt/2                PRESS:NASTY.  nas_rule/3 sinatt/2

sincos/2                PRESS:TRIG.FAC
                                      <user> trigmethod/3

sinexp/4                PRESS:HOMOG.TRG
                                      expss/4

sinfind/1               PRESS:HOMOG.TRG
                                      <user> anaz2/3
```

```
single_occ/2            PRESS:MISC.   solve1/4 good_eqn/2

singleton_method/3      PRESS:POLY.   poly_method/4

sinhp/3                 PRESS:HOMOG.TRG
                                      <user> action1/5

slope/2                 PRESS:INIT.   interval/3

smaller/2               PRESS:PROVER. <user> maximum1/2

solve/1                 PRESS:SOLVE.  <user> so/0 redo/0

solve/2                 PRESS:SOLVE.  <user>

solve/3                 PRESS:SOLVE.  <user> simsolve1/3 findbnd/3 findmax/3

solve/4                 PRESS:SOLVE.  solve/1 solve/2 solve/3

solve1/4                PRESS:SOLVE.  solve/4 disj_solve_list/4 fact_solve/4
                                      solve2/4 subst_solve/5

solve2/4                PRESS:SOLVE.  solve1/4

solveineq/3             PRESS:INEQ.   <user> min/3

some/2                  utility       anaz2/3 maximum1/2

special_atom/1          PRESS:FACTS.  positive/1 non_neg/1 non_zero/1

split/4                 PRESS:INT.    make_regions/3 split/4

split1/3                PRESS:INT.    split/4

split_case/3            PRESS:HOMOG.TOP
                                      anaz/6

split_case1/3           PRESS:HOMOG.TOP
                                      split_case/3 split_case1/3

split_two_ways/3        PRESS:MATCH.  match/2 split_two_ways/3

st/2                    PRESS:HOMOG.TRG
                                      <user> findtype_trig/2

sth/2                   PRESS:HOMOG.TRG
                                      <user> findtype_hyper/2

strip/3                 PRESS:NASTY.  nasty_act/5

strip_num/2             PRESS:WORDS.  wordsin/2 frequent_words/2 strip_num/2

sub_int/2               PRESS:INT.    in/2 all_are_contained/2 split1/3

subintegral/2           PRESS:NASTY.  remove_arbs/2 subintegral/2

sublist/3               utility       pick_xeqn/4 intermediates_in/2 findmax/
                                      linear_sin_cos/2 trigmethod/3
                                      report_subs/2
```

```
subnasty/3              PRESS:NASTY.  try_nasty_method/3 subnasty/3

subs1/3                 PRESS:HOMOG.MSC
                                      perform_rewrites/6 remove_arbs1/3 subs1

subst/3                 utility       subst2/4 change_the_variable/5
                                      nasty_act/5 normstore/3 dx/3 subst_mess
                                      subs1/3

subst2/4                PRESS:INEQ.   <user> findmax/3

subst_mess/3            PRESS:MISC.   solve2/4 distribute/3 sim2/6
                                      subst_solve/5 givesneg/3 changevar/4

subst_solve/5           PRESS:SIM.    listsolve1/5

subtract/3              utility       pick_xeqn/4

suit1/3                 PRESS:LOG.    suitable/3 suit1/3

suitable/3              PRESS:LOG.    logmethod/4

sumdiff/10              PRESS:TRIG.FAC
                                      add_angle/10

sym_transform/2         PRESS:POLPAK. poly_method/4

symmetric/2             PRESS:POLPAK. odd_symmetric/1 even_symmetric/1
                                      symmetric/2

takelog/3               PRESS:LOG.    logmethod/4 takelog/3

tanatt/2                PRESS:NASTY.  nag_rule/3

taneqn/3                PRESS:HOMOG.TRG
                                      anaz1/6

tanexp_denom/4          PRESS:HOMOG.TRG
                                      exptt/4 tanexp_denom/4

tanexp_num/4            PRESS:HOMOG.TRG
                                      exptt/4 tanexp_num/4

tanhp/3                 PRESS:HOMOG.TRG
                                      <user> action1/5

tantype/2               PRESS:HOMOG.TRG
                                      anaz1/6 tantype/2

tantype1/2              PRESS:HOMOG.TRG
                                      tantype/2

term_size/2             PRESS:MISC.   extreme_term/3 extreme_term/5 term_size

term_size/4             PRESS:MISC.   term_size/2 term_size/4

tidy/2                  utility       solve2/4 process_answer/3 tidy/3 sim/3
                                      sim1/4 listsolve/5 try_sort/3
                                      dottoor_set/2 solveineq/3 subst2/4
                                      isolate/3 cond_poly_print/4
                                      remove_neg_powers/4 linear_method/2
                                      discriminant/4 roots/6 poly_method/4
                                      collect/3 attract/3
                                      unattract_distribute/3 dist_multiply/3
                                      trigsolve/5 try_factorize/4
                                      try_factorize/3 add_angle/10 sumdiff/10
                                      derive/7 convert_functor/8 trigsolve1/5
                                      mod_angsize1/4 correct_sin1/4
                                      correct_cos1/3 change_the_variable/5
                                      postidy/2 nasty_method/3
                                      try_nasty_method/3 nasty_act/5 nas_rule/
                                      sinatt/2 cosatt/2 tanatt/2 trig_inv/3
                                      multiply_through/4 normstore/3 merge/2
                                      isolax/4 isolax/4 rew_rule/5 zero/1
                                      diffwrt/3 dx/3 add_poly/3 timesing1/4
                                      div_lin/5 reify/3 trans/2 poly_tidy/2
                                      weaknf/3 zero_rhs/2 subst_mess/3 form1/
                                      form2/3

tidy/3                  PRESS:SOLVE.  process_input/4

tidy_ops/2              PRESS:COLLEC. exp_match1/5 tidy_ops/2

times_poly/3            PRESS:POLPAK. poly/3 times_poly/3 binomial/3

timesing1/4             PRESS:POLPAK. times_poly/3 timesing1/4

trace/2                 utility       print_the_answer/2 poly_method/4
                                      trigsolve1/5 apcheck/4 anaz/6 anaz1/6
                                      looping1/1 div_lin/5 report_subs/2
                                      report_on/0 report_off/0

trace/3                 utility       disj_solve_list/4 fact_solve/4 solve2/4
                                      process_input/4 print_the_answer/2
                                      simsolve/3 simsolve2/3 sim1/4 sim2/6
                                      min/3 solveineq/3 findbnd/3 modcall/1
                                      mod_trace/1 cond_poly_print/4
                                      remove_neg_powers/4 roots/6
                                      warn_if_complex/1 poly_method/4 attract
                                      div_list/4 trigsolve/5 try_factorize/4
                                      try_factorize/3 convert_functor/8 homog
                                      homog1/8 try_nasty_method/3 nasty_act/5
                                      isolax/4 rew_rule/5
                                      make_assumption_positive/1 diffwrt/3
                                      arbint/1 cond_print/3 subst_mess/3
                                      report_subs1/1

trans/2                 PRESS:POLPAK. sym_transform/2

tree_list/4             PRESS:WORDS.  wordsin/2 frequent_words/2 tree_list/4

tree_size/3             PRESS:ATTRAC. closeness/3 tree_size/5

tree_size/5             PRESS:ATTRAC. tree_size/3 tree_size/5

trig_fac/3              PRESS:TRIG.FAC
                                      solve2/4

trig_inv/3              PRESS:NASTY.  sinatt/2 cosatt/2 tanatt/2 trig_inv/3
```

```
trig_nasty/2            PRESS:NASTY.  nasty/2

trig_normal_form/3      PRESS:TRIG.FAC
                              trig_fac/3

trigf/1                 PRESS:HOMOG.TOP
                              trigtype/3 <user> findtype/2 trig_nasty
                              expon/2 attract_list/3 dl_parse2/3

trigmethod/3            PRESS:TRIG.FAC
                              trig_fac/3

trigsolve/5             PRESS:TRIG.FAC
                              trig_fac/3 trigsolve/5 trigsolve1/5

trigsolve1/5            PRESS:TRIG.FAC
                              convert_functor/8

trigtype/3              PRESS:TRIG.FAC
                              maptrigtype/3

trivial_set/2           PRESS:SIM.    checktrivial_set/2

try_factorize/3         PRESS:TRIG.FAC
                              trigsolve/5

try_factorize/4         PRESS:TRIG.FAC
                              trigsolve/5

try_isolate/3           PRESS:NASTY.  nasty_act/5

try_nasty_method/3      PRESS:NASTY.  nasty_method/3

try_sort/3              PRESS:SIM.    reorder_eqn/3 try_sort/3

ttyprint/1              utility       show/0

unattract_distribute/3  PRESS:TRIG.FAC
                              trig_normal_form/3
                              mapunattract_distribute/3

union/3                 utility       suitable/3 suit1/3

updown_flip/3           PRESS:INT.    set_bnds/4

verify/2                PRESS:PROVER. solve1/4

vet/2                   PRESS:INT.    vet/2

warn_if_complex/1       PRESS:POLY.   roots/6

weaknf/3                PRESS:WEAKNF. solve1/4 good_eqn/2 multiply_through/4

wordsin/2               PRESS:WORDS.  listsolve1/5 intermediates_in/2 neg_exp
                              remove_arbs/2 simplify/2 numeric/1

writef/1                utility       try_isolate/3 trans/2

z_norm/2                PRESS:POLPAK. poly_method/4 poly_norm/3 z_norm/2
```

```
zero/1                  PRESS:FACTS.  solve1/4

zero_rhs/2              PRESS:WEAKNF. weaknf/3
```

```
/* PRESS.DEF :

                                                    Bernard Silver
                                                    Updated: 27 July 82
*/

cross_ref_file(xref).
title('PRESS Equation Solving System').
width(80).
globals_file(no).
update_globals(no).

called(solve(Eqn,X,Ans)).
called(solve(Eqn,X)).
called(solve(Eqn)).
called(sim(Eqns,Unks,Ans)).
called(sim(Eqns,Unks)).
called(sim(Eqns)).
called(simsolve(Eqns,Unks,Ans)).
called(simsolve(Eqns,Unks)).
called(simsolve(Eqns)).
called(solveineq(Eqns,Unks,Ans)).
called(go).
called(oops).
called(bye).
called(redo).
called(report_on).
called(report_off).
called(frequent_words(Exp,Ans)).

applies(isolax(Posn,Old,New,Cond),Cond).
```

```
/* PRESS.OPS : Operator declarations for Press

                                                  Updated: 12 August 82
*/

%% The following are now in UTIL:ARITH.OPS and are loaded into UTIL

   :- op(500,yfx,[++,--]).
   :- op(400,yfx,[div,mod]).
   :- op(300,xfy,[:,^]).


%% Since there is nothing else this file is not currently used (in FILIN)
```

```
/* FACILE : Some conveniences for PRESS

                                                     Lawrence
                                                     Updated: 3 April 81

*/

%% Run Interpreted %%



                           % Go from the terminal

go    :- ttynl, display('Equation: '), ttyflush,
         read(Equation),
         asserta( last_equation(Equation) ),
         solve(Equation).



                           % Show all the equations

show :- ttynl, display('Equations:'), ttynl, ttynl,
         last_equation(Equation),
         ttyprint(Equation), ttynl,
         fail.

show.



                           % Redo last equation

redo :- call( last_equation(Equation) ),
         !,
         solve(Equation).



                           % Remove record of last equation

oops :- retract( last_equation(_) ),
         display('(Ok, I''ve forgotten it!)'), ttynl,
         !.



                           % Leave Press, showing all the equations

bye   :- log,
         show, ttynl,
         display('Goodbye'), ttynl,
         halt.
```

```
/* INIT. :  Add dummy definitions from MECHO database

Used to allow better use of unknown(_,trace).

                                                     Bernard Silver
                                                     Updated: 31 May 82

*/

measure(_,_) :- fail.

const(_) :- fail.

quantity(_) :- fail.

incline(_,_,_):- fail.

slope(_,_) :- fail.

concavity(_,_) :- fail.

angle(_,_,_) :- fail.

partition(_,_) :- fail.

sought(_) :- fail.

intermediate(_) :- fail.

given(_) :- fail.

last-equation(-) :- fail
```

```
/* TIME : Time some bits of Press

                                                    Lawrence
                                                    Updated: 7 April 81
*/

  :- public     timetest/1.


                           % Do the tests

timetest(1)
     :- statistics(runtime,[Start|_]),
        t1(10000),
        statistics(runtime,[Finish|_]),
        Time is Finish-Start,
        ttynl, display('Time for test 1 is '),
        display(Time), display(' milliseconds'), ttynl.

timetest(2)
     :- statistics(runtime,[Start|_]),
        t2(10000),
        statistics(runtime,[Finish|_]),
        Time is Finish-Start,
        ttynl, display('Time for test 2 is '),
        display(Time), display(' milliseconds'), ttynl.

timetest(3)
     :- statistics(runtime,[Start|_]),
        t3(10000),
        statistics(runtime,[Finish|_]),
        Time is Finish-Start,
        ttynl, display('Time for test 3 is '),
        display(Time), display(' milliseconds'), ttynl.

timetest(4)
     :- statistics(runtime,[Start|_]),
        t4(10000),
        statistics(runtime,[Finish|_]),
        Time is Finish-Start,
        ttynl, display('Time for test 4 is '),
        display(Time), display(' milliseconds'), ttynl.


                           % The things to be timed

t1(0) :- !.

t1(N) :- N1 is N-1, t1(N1).


t2(0) :- !.

t2(N) :- N1 is N-1, task2(N), t2(N1).


task2(N) :- call(oddnum(N)), !.
task2(N).
```

```
t3(0) :- !.

t3(N) :- N1 is N-1, task3(N), t3(N1).


task3(N) :- eval(odd(N)), !.
task3(N).


t4(0) :- !.

t4(N) :- N1 is N-1, task4(N), t4(N1).


task4(N) :- mylocalodd(N), !.
task4(N).


mylocalodd(N) :- 1 is N mod 2.
```

from MEC:

```
; PRESS.MIC  -  Load Press      '<silence>
;
;        This Junk allows for automatic loading believe it or not
;
;        Call as:        /press          - to load press (normal use)
;                        /press auto     - used by MAKSYS
;
.on error:backto death
.error ?
.on operator:backto death
.operator !
.goto cont
death::
*^C
*^C
.if ($a = "auto") .let e1 = "error"
! PRESS.MIC  HALTED
.mic return
cont::
.let y = $date.["-",20], d = $date.[1,"-"]+" "+$y.[1,"-"]+" "+$y.["-",4]
.if ($d.[1] = "0") .let d = $d.[2,20]
;
.run util[400,444]  '<revive>                    ; Must use UTIL
* :- [filin].
* :- version(''Press Algebra System  ('d)
*Copyright (C) 1981 Dept. Artificial Intelligence. Edinburgh'').
* :- asserta( version_date('''d'') ).
* :- ok.
.save press[400,444]
```

```
/*                  SOLVE                       19.2.81 */
/*                                               5.4.81 poly_solve */
/*weaknf added 27.4.81  loss added 19.8.81 nasty added 4.9.81*/
/*                                    Updated: 27 June 82
/*********************************************************
        SOLVE ONE EQUATION OR INEQUALITY
*********************************************************/

/* Top Level Solve Procedure */

solve(Eqn,X,Ans) :-
    fixvar(Eqn,X),
    trace('\nSolving %t for %t\n',[Eqn,X],1),
    tidy(Eqn,Eqn1),
    abolish(seen_eqn,1),                            %Initialize Looping Check
    assert((seen_eqn(_) :- fail)),
    cond_print(Eqn,Eqn1),
    solve1(Eqn1,X,Ans1),
    remove_dis_dups(Ans1,Ans2),                     % Remove duplicate answers
    poly_form(Ans2,Ans),                            % in disjunctive solutions
    trace('\nAnswer is : %e\n',[Ans],1),
    !.




/*equation does not contain X*/
solve1(Eqn,X,Soln) :- freeof(X,Eqn), !,
  simplify(Eqn,Soln).


/* Deal with disjunction */
solve1(Exp1#Exp2,X,Ans1#Ans2) :-
        !,
        solve1(Exp1,X,Ans1), solve1(Exp2,X,Ans2).


/* See if eqn is factorizable*/
solve1(V1*V2=0, X, Ans) :-
        trace('\nFactorising\n\n (%t)*(%t)\ninto\n%t and %t.\n',[V1,V2,V1,V2],1),
        factsolve(V1,V2,X,Ans),
        !.

factsolve(V1,V2,X,Ans) :- freeof(X,V1),
        non_zero(V1),
        trace('\nSolving %t = 0\n',[V2],1),
        solve1(V2=0,X,Ans),
        !.
factsolve(V2,V1,X,Ans) :- freeof(X,V1),
        non_zero(V1),
        trace('\nSolving %t = 0\n',[V2],1),
        solve1(V2=0,X,Ans),
        !.
factsolve(V1,V2,X,Ans) :- trace('\nSolving %t = 0\n',[V1],1),
        solve1(V1=0,X,Ans1),
        trace('\nSolving %t = 0\n',[V2],1),
        solve1(V2=0,X,Ans2),
        tidy(Ans1#Ans2,Ans3),
        remove_false(Ans3,Ans),
        !.
```

```
/* If single occurence of unknown then Isolate */

solve1(Exp,X,Ans) :-
        singleocc(X,Exp),
        !,
        position(X,Exp,Posn),
        isolate(Posn,Exp,Ans1),
        remove_false(Ans1,Ans2), %Hack for false
        tidy(Ans2,Ans).

/* Special Polynomial Method */

solve1(L=R,X,Ans) :-
        poly_norm(X,L+(-1)*R,Plist),
        !,
        make_poly(X,Plist,Pol1),
        tidy(Pol1,Pol),
        cond_print(L=R,Pol = 0),
        poly_solve(X,Plist,Ans).

poly_solve(X,Plist # Qlist, PAns # QAns) :-
        !,
        make_poly(X,Plist,Pol1),
        make_poly(X,Qlist,Qol1),
        tidy(Pol1,Pol2),
        tidy(Qol1,Qol2),
        trace('\nFactorising into two polynomial equations\n',1),
        trace('\n %t = 0 \n and \n %t = 0\n',[Pol2,Qol2],1),
        poly_solve(X,Plist,PAns),
        poly_solve(X,Qlist,QAns).

poly_solve(X,Plist,Ans) :-
        poly_method(X,Plist,Ans1),
        !,
        remove_false(Ans1,Ans),
        trace('\n%t is a solution\n',[Ans],1).


/* Convert equation to weak normal form,all terms containing the
unknown are put on the left,all constants on the right   */

solve1(Eqn,X,Ans) :- weaknf(Eqn,X,Eqn1),
        solve2(Eqn1,X,Ans),
        !.


/* Try to Change the unknown to simplify equation   */

solve2(Eqn,X,Ans) :- changeunknown1(Eqn,X,Term),!,
        changevar(X,Term,Eqn,Ans),
        !.

/* Apply Collection to reduce occurrences of unknown */

solve2(Exp=Rhs,X,Ans) :-
        collect(X,Exp,New),
        !,
        trace('\n%t = %t\n',[New,Rhs],1),
        solve1(New=Rhs,X,Ans).
```

```
/* Apply Attraction to move occurrences of unknown closer together */

solve2(Exp=Rhs,X,Ans) :-
        closeness(X,Exp,EC),
        attract(X,Exp,New),
        closeness(X,New,NC),
        EC>NC,
        !,
        trace('\n%t = %t\n',[New,Rhs],1),
        solve1(New=Rhs,X,Ans).

/* Trig factorization method */

solve2(Eqn,X,Ans) :-
        trig_fac(Eqn,X,Neweqn),
        solve1(Neweqn,X,Ans),
        !.

/* Try to remove dominating functor  */

solve2(Eqn,X,Ans) :- nag1(Eqn,X,Posn),
        isolate(Posn,Eqn,New),
        findrhs(New,List),
        checklist(freeof(X),List),
        solve1(New,X,Ans),
        !.

/* Try homogenization     */

solve2(Eqn,X,Ans) :-
        homog(Eqn,X,Neweqn,Term,V),
        tidy(Neweqn,Neweqn1),
        solve1(Neweqn1,V,Vans),
        subst_mess(V=Term,Vans,Uans),
        solve1(Uans,X,Ans),
        !.

/* Try to take logs if equation is in suitable form  */

solve2(Eqn,X,Ans) :- logmethod(Eqn,X,New,Base),
        tidy(New,New1),
        trace('\nTaking logs, base %t, gives \n\n%t\n',[Base,New1],1),
        solve1(New1,X,Ans),!.

/* Try to eliminate Nasty Functions */

solve2(Eqn,X,Ans) :- nasty_method(Eqn,X,Neweq),
        tidy(Neweq,Neweqn),
        solve1(Neweqn,X,Ans),
        !.


/* One and two argument solve clauses for easy type-in. */

solve(Exp) :- solve(Exp,x,A).
solve(Exp,Unk) :- solve(Exp,Unk,Ans).
```

```
/* SIMEQ. :          May 1981

                                                  Alan Bundy
                                                  Updated: 31 May 82

        Simultaneous Equations Routines        */

/*simultaneous solution with messages*/
simsolve(Eqns,Us,Ans)
        :- trace('Simultaneously solving : %cFor %t.\n',[Eqns,Us],1),
        simsolve1(Eqns,Us,Ans1),
        remove_dis_dups(Ans1,Ans),
        trace('\nFinal Answers are : %e', [Ans],1),
           !.


/* Solve conjunction of equations */
simsolve1(EqnsA & EqnsB,[X|Unks], Ans1) :- !,
        pick_xeqn(EqnsA & EqnsB,X,XEqn,Rest),
        solve(XEqn,X,Ans),
        distribute(Ans,Rest,Eqns1),
        simsolve2(Eqns1,Unks,Ans1).



/*single equation*/
simsolve1(A=B, [U], Ans) :- !, solve(A=B,U,Ans).


/*basis case*/
simsolve1(true,[],true) :- !.

/*Pick equation to solve for x, and return the remainder */
pick_xeqn(EqnC,X,XEqn,RestC) :-  !,
        andtodot(EqnC,EqnL),
        sublist(contains(X),EqnL,XEqnL),
        subtract(EqnL,XEqnL,NonXRestL),
        select(XEqn,XEqnL,XRestL),
        append(XRestL,NonXRestL,RestL),
        dottoand(RestL,RestC).

/* Distribute Or over And */
distribute(Sub1 # Sub2, Exp, Ans1 # Ans2) :- !, % disjunction case
        distribute(Sub1,Exp,Ans1),
        distribute(Sub2,Exp,Ans2).

distribute(Sub, Exp, Sub & Ans) :- !,    % conjunction or single equation case
        subst_mess(Sub,Exp,Ans).


/* Call simsolve1 recursively and substitute back */
simsolve2(Eqns1 # Eqns2, Unks, Ans1 # Ans2) :- !,        % Solve disjunction
        simsolve2(Eqns1,Unks,Ans1),
        simsolve2(Eqns2,Unks,Ans2).

simsolve2(X=Ans1 & Eqns, Unks, Ans3) :- !,       % Discount already solved equatio
        simsolve1(Eqns, Unks, Ans2),
        trace('Substituting back in %t solution\n',[X],1),
        distribute(Ans2,X=Ans1,Ans3).
```

```
% Clauses for easy type-in
simsolve(Eqns,Unks) :- simsolve(Eqns,Unks,Ans).

simsolve(Eqns) :- simsolve(Eqns,[x,y],Ans).

/* Problems
        2. Return particular solutions; alternates on backtracking.
        4. Reject silly answers as required by Cardan. (??)
*/
```

```
 %                 SIM
 % Simplify simultaneous equations using homogenization
 % Bernard Silver 12.9.81
 % Updated: 31 May 82

 % Top level
 % Find the offending terms in each unknown

sim(Eqns1,Unks,Ans) :- tidy(Eqns1,Eqns),
        mapmodparse(Eqns,Unks,Offends),
        sim1(Eqns,Unks,Offends,Ans),
        !.

 % If all the offending sets are empty or contain only the unknown
 % use normal method (simsolve)

sim1(Eqns,Unks,Offends,Ans) :- checktrivial_set(Unks,Offends),
        simsolve(Eqns,Unks,Ans),
        !.

 % Otherwise try to use homogenization

sim1(Eqns,Unks,Offends,Ans) :-
        trace('Simultaneously solving : %c For %t.\n',[Eqns,Unks],1),
        apply_sim2(Eqns,Unks,Offends,New,Vs,Terms),
        !,
        tidy(New,New1),
        reorder_eqn(Vs,New1,New2),
        simsolve1(New2,Vs,Ans1),
        ortodot(Ans1,Dislist),
        listsolve(Dislist,Vs,Terms,Unks,Ans2),
        dottoor_set(Ans2,Ans),
        trace('\nFinal Answers are : %e ',[Ans],1).


 % If homogenization fails try simsolve
sim1(Eqns,Unks,_,Ans) :- simsolve1(Eqns,Unks,Ans1),
        tidy(Ans1,Ans),
        trace('\nFinal Answers are : %e',[Ans],1),
        !.

apply_sim2(Eqns,[],[],Eqns,[],[]) :- !.
apply_sim2(Eqns,[H|T],[O1|T1],New,[V1|T2],[Term1|T3]) :-
        sim2(Eqns,H,O1,New1,V1,Term1),
        apply_sim2(New1,T,T1,New,T2,T3),
        !.

 % sim2(Eqns,Unknown,Newequation,Identifier,Reduced_Term) applies
 % homogenization to the set of equations,homogenizing in Unknown

sim2(Eqns,X,[],Eqns,X,X) :- !. %Eqns do not contain X
sim2(Eqns,X,[X],Eqns,X,X) :- !. %Eqns is already homogeneous in X
sim2(Eqns,X,Y,Eqns,X,X) :- checklist(polytype(X),Y),!.
 %Only Polynomials

 % Change of Unknown case
sim2(Eqns,X,[A],New,V,A) :- identifier(V),subst_mess(A=V,Eqns,New),!.

 % Homogenize
sim2(Eqns,X,Off,New,V,Term) :- homog1(Eqns,X,New,Term,V,Off,Hom,sim),
        trace('\nHomogenizing equations in %t\n gives %c\n',[X,Hom],1),
        trace('\nSubstituting %t = %t gives %c\n',[V,Term,New],1),
        !.

 % listsolve(ListofAns,Newunks,Reducedterms,Oldunks,Newans)
 % ListofAns is the list of answers in the Newunks returned by simsolve1.
 % listsolve now solves the substitution equations (of the form
 % Newunk1=Ans1 & Reducedterm1=Newunk1) in terms of the Oldunks to give
 % Newans

listsolve([],_,_,_,[]) :- !.
listsolve([A|T],X,Y,Z,[A1|T1]) :- andtodot(A,A2),
        listsolve1(A2,X,Y,Z,A3),
        dottoand(A3,A4),
        tidy(A4,A1),
        listsolve(T,X,Y,Z,T1),
        !.

listsolve1([],_,_,_,[]) :- !.
listsolve1([H|T],Vs,Terms,Unks,[Ans|Tail]) :-
        wordsin(H,Words),
        correspond1(Words,Vs,Terms,Unks,Id,Term,Unk),
        subst_solve(Id,Term,H,Unk,Ans),
        listsolve(T,Vs,Terms,Unks,Tail),
        !.

 % Solve substitution equation

 %No substitution needed
subst_solve(X,X,Unk=Ans,Unk,Unk=Ans) :- !.

 % General case
subst_solve(Id,Term,H,Unk,Ans) :- subst_mess(Id=Term,H,New),
        solve1(New,Unk,Ans),
        !.

 % The offending set is trivial,ie it is empty or contains just the unknown
checktrivial_set(_,[]) :- !.
checktrivial_set(X,[H|T]) :- trivial_set(X,H),checktrivial_set(X,T),!.

trivial_set(_,[]) :- !.
trivial_set(Unklist,[X]) :- member(X,Unklist),!.


 %Reorder equations so nicest occurs first
reorder_eqn([X|_],Old,New) :- try_sort(X,Old,New).

 % Equation to be solved first should have only one 'easy' occurrence of X
try_sort(X,First&Rest,First&Rest) :- good_eqn(X,First),!.
try_sort(X,F&Rest,New) :- try_sort(X,Rest,New1),tidy(New1&F,New).
try_sort(X,F,F) :- !.

 %Occurrence is easy if it is a first order polynomial
good_eqn(X,Eqn) :- singleocc(X,Eqn),
        weaknf(Eqn,X,Lhs=Rhs),
        poly_norm(X,Lhs,[polyand(1,_)|_]),
        !.

 % Multilist version of correspond/4
 % correspond1(List,L1,L2,L3,T1,T2,T3)
 % List, L1,L2,L3 are lists,T1 is a member of List that also occurs in L1.
 % T2 and T3 occur in the same position in L2 and L3 as T1 does in L1

correspond1([],_,_,_,_,_,_) :- !,fail.
correspond1([H|_],L1,L2,L3,H,T2,T3) :-
        correspond2(H,L1,L2,L3,T2,T3),
        !.
correspond1([_|H],L1,L2,L3,T1,T2,T3) :-
        correspond1(H,L1,L2,L3,T1,T2,T3),
        !.

correspond2(H,[H|_],[H1|_],[H2|_],H1,H2) :- !.
correspond2(H,[_|T],[_|T1],[_|T2],H1,H2) :-
        correspond2(H,T,T1,T2,H1,H2),
        !.

 % Modified parser,deals with & and =,and also reorders the expression
mapmodparse(_,[],[]) :- !.
mapmodparse(X,[H|T],[H1|T1]) :- modparse(X,H,H1),mapmodparse(X,T,T1),!.

modparse(A&B,X,Off) :- !,modparse(A,X,O1),modparse(B,X,O2),union(O1,O2,Off).
modparse(A=B,X,Off) :- !,modparse(A,X,O1),modparse(B,X,O2),union(O1,O2,Off).
modparse(A,X,Off) :- parse(A,Off,X),!.

 % These are needed to deal with disjunctive solutions from simsolve1
dottoor_set(List,Ans) :- listtoset(List,L1),dottoor(L1,Ans1),tidy(Ans1,Ans),!.

 % Equation doesn't need homogenization in X
polytype(X,X) :- !.
polytype(X,X^N) :- integer(N),!.

 %Clauses for easy type in

sim(Eqns) :- sim(Eqns,[x,y],Ans).

sim(Eqns,Unks) :- sim(Eqns,Unks,Ans).
```

```
/*               INEQ       19.2.81  */

:- public
                 findbnd/3,
                 givesneg/3,
                 subst2/4.

/*******************************
        MULTIPLE INEQUALITIES
*******************************/

/*FIND MINIMUM VALUE OF X FOR WHICH EXP IS TRUE*/
min(Exp,X,Minval) :- solveineq(Exp,X,X>=Minval),
  trace('Hence minimum value of %c is %c\n',[X,Minval],1),
  !.

/*SOLVE INEQUALITY CONJUNCTION*/
solveineq(Exp,X,Ans) :-
  trace('Trying to solve %c\n',[Exp],1),
  tidy(Exp,Exp1),
  fixvar(Exp1,X), mapand(findbnd(X),Exp1,Ansset),
  trace('Isolating %t on the lhs gives %c\n',[X,Ansset],1),
  trace('Trying to find maximum of : %c',[Ansset],3),
  !,
  maximum(Ansset,Ans1), tidy(Ans1,Ans),
  trace('%t dominates the other inequalities.\n',[Ans],1).


/*SOLVE INEQUALITY*/
findbnd(X,true,true) :-!.

findbnd(X,Ineq,Ans) :-
  solve(Ineq,X,Ans1),  Ans1=..[Prop,X,Bnd1],
  (intermediates_in(Bnd1,[Y]) -> findmax(Bnd1,Y,[Bnd]);
    Bnd1=Bnd), Ans=..[Prop,X,Bnd],
  !.

findbnd(X,Ineq,Ans) :-
  trace('Unable to find bounds for %t in %t.\n',[X,Ineq],2), !, fail.




/*GET LIST OF INTERMEDIATES IN EXP*/
intermediates_in(Exp,Inters) :-
  wordsin(Exp,Words), sublist(intermediate,Words,Inters),
  !.

/*FIND MAXIMUM VALUES OF EXPRESSION*/
findmax(Exp,X,Maxvals) :-
  diffwrt(Exp,Exp2,X),
  solve(Exp2=0,X,Soln),
  collect_ans(X,Soln,Anslist),
  diffwrt(Exp2,Exp3,X),
  sublist(givesneg(X,Exp3),Anslist,Maxargs),
  maplist(subst2(X,Exp),Maxargs,Maxvals),
  !.

/*special substitution to suit maplist*/
subst2(X,Exp,Arg,Val) :- subst(X=Arg,Exp,Val1), tidy(Val1,Val), !.
```

```
/*MAKE LIST OF ALTERNATIVE ANSWERS*/
collect_ans(X,true, [X]) :- !.

collect_ans(X,false, []) :- !.

collect_ans(X,X=Ans,[Ans]) :- !.

collect_ans(X,Exp1#Exp2,Anslist) :-
  collect_ans(X,Exp1,Anslist1), collect_ans(X,Exp2,Anslist2),
  append(Anslist1,Anslist2,Anslist),
  !.

/*SUBSTITUTING ANS FOR X IN EXP GIVES NEGATIVE RESULT*/
givesneg(X,Exp,Ans) :-
  subst_mesg(X=Ans,Exp,Exp1), negative(Exp1),
  !.
```

```
/* IDENT. :                Prove identities with PRESS
Written 1.11.1981
                                                          Bernard Silver
                                                          Updated: 12 May 82
*/

/* Top level X is the possible identity */
identity(X) :- trace('\nTrying to prove that\n%t\nis an identity\n',[X],1),
        tidy(X,Y),
        cond_print(X,Y),
        abolish(seen_eqn,1),
        ident(Y),
        !.

/* Recursive call top level */
identity1(X) :- tidy(X,Y),cond_print(X,Y),ident(Y),!.

/* Base cases */
ident(false) :- trace('\nExpression is not an identity\n',1).
ident(true) :- trace('\nExpression is an identity\n',1).
ident(A=A) :-  trace('\nIdentically true\n',1).  %unifies

/* Find words in expression */
ident(X) :- wordsin(X,Words),ident1(X,Words),!.

/* No words remaining,so fail */
ident1(_,[]) :- trace('\nCannot show identity\n',1),!,fail.
/* Try to solve as an equation with unknown X */
ident1(X,[H|_]) :- ident2(X,H),!.
/* Try next word, if any */
ident1(X,[_|T]) :- ident1(X,T),!.

/* Put expression in weak normal form and try PRESS methods */
ident2(X,Unk) :- weaknf(X,Unk,New),ident3(New,Unk),!.

ident3(A,Unk) :- occ(Unk,A,1),  %isolation
        position(Unk,A,Posn),
        isolate(Posn,A,New),
        tidy(New,New1),
        cond_print(New,New1),
        terminate_ident(New),
        !.
ident3(L=R,X) :- poly_norm(X,L+(-1)*R,Plist), %polynomial
        !,
        make_poly(X,Plist,Pol),
        cond_print(L=R,Pol=0),
        poly_solve(X,Plist,Ans),
        ident(Ans).
ident3(Old=Rhs,Unk) :- mult_occ(Old,Unk), %collection
        collect(Unk,Old,New1),
        tidy(New1=Rhs,New),
        trace('\n%t\n',[New],1),
        identity1(New),
        !.
ident3(Old=Rhs,X) :-  mult_occ(Old,X), %attraction
        closeness(X,Old,EC),
        attract(X,Old,New1),
        closeness(X,New1,NC),
        EC>NC,
        !,
        tidy(New1=Rhs,New),
        trace('%c\n',[New],1),
        identity1(New).
ident3(A=B,Unk) :- occ(Unk,A,N), %change of unkown
        eval(N>1),
        setof(T,good_subterm(A,Unk,N,T),Tset),
        extreme_term(Tset,>,T),
        identifier(New),
        subst_mess(T=New,A=B,Neweq),
        identity1(Neweq),
        !.
ident3(Old,Unk) :- trig_fac(Old,Unk,New), %trig methods
        trace('\n%t\n',[New],1),
        identity1(New),
        !.
ident3(Old,Unk) :- mult_occ(Old,Unk), %homogenization
        homog(Old,Unk,New,_,_),
        identity1(New),
        !.
ident3(Eqn,X) :-  mult_occ(Eqn,X), %nas1
        nas1(Eqn,X,Posn),
        isolate(Posn,Eqn,New),
        findrhs(New,List),
        checklist(freeof(X),List),
        identity1(New),
        !.
ident3(Eqn,X) :- logmethod(Eqn,X,New,Base), %logmethods
        trace('\nTaking logs, base %t, gives\n\n%t\n',[Base,New],1),
        identity1(New),
        !.
ident3(Eqn,X) :- nasty_method(Eqn,X,Neweq),identity1(Neweq),!. %nasties

/* Examine result of isolation  */
terminate_ident(true) :- trace('\nExpression is identity\n',1),!.
terminate_ident(_) :- trace('\nExpression is not an identity\n',i),!.
```

```
/* ISOLAT. :

                                                                19.2.81
                                                             Modified 19.9.81
                                                          Updated: 7 September 82
*/

:- public
                  isolate/3.


/* ISOLATION ROUTINES*/

isolate([N|Posn],Exp,Ans) :-
        maneuver_sides(N,Exp,NewExp),
        isolate1(Posn,NewExp,Inter),
        tidy(Inter,Ans),
        mod_trace(Ans).

/*set term to be isolated on Rhs */

maneuver_sides(1,Exp,Exp) :- !.

maneuver_sides(2,Exp,NewExp) :-
        !,
        Exp=..[Sym,Lhs,Rhs],
        invert(Sym,Sym1),
        NewExp=..[Sym1,Rhs,Lhs].

%% Perform the Isolation %%

/*trivial boolean cases*/

isolate1(Posn,false,false).
isolate1(Posn,true,true).

/*deal with each disjunct*/

isolate1(Posn,Eqn1#Eqn2,Ans1#Ans2) :-
        !,
        isolate1(Posn,Eqn1,Ans1),
        isolate1(Posn,Eqn2,Ans2).


/*expression is already isolated*/

isolate1([],Ans,Ans) :- !.

/*expression can have isolax rule applied*/

isolate1([N|Posn],Old,Ans) :- !,
        isolax(N,Old,New,Condition),
        modcall(Condition),    %Hack for non_zero
        isolate1(Posn,New,Ans).

/* Inversion of Predicates */

invert(S1,S2) :- perm2(S1,S2,S3,S4), invert1(S3,S4), !.

invert1(=,=) :- !.
```

```
invert1(>,<) :- !.
invert1(>=,=<) :- !.

/* Overcoming non_zero, etc. condition */

modcall(A&B) :- !,modcall(A),modcall(B).
modcall(non_zero(X)) :- non_zero(X),!.
modcall(non_zero(X)) :- eval(X=0),!,fail.
modcall(non_zero(X)) :- trace('\nAssuming %t is non-zero\n',[X],1),!.
modcall(X) :- call(X),!.

/* Output result */

mod_trace(false) :- !. % Hack for false case
mod_trace(Exp) :- trace('%c      (by Isolation)\n',[Exp],1),!.
```

```
/*                 POLY                          19.2.81

                              Written as per note 82 in Mecho folder
                                                 1.5.81  Leon
                                           Updated: 8 September 82
*/

% Poly_solve is only called when it has been determined that the
% equation is a polynomial equation.
%  i.e. a precondition that the method is called is that is_poly is true

poly_solve(Eqn1#Eqn2,X,Soln1#Soln2,Rules-Diff) :-
        poly_solve(Eqn1,X,Soln1,Rules-Inter),
        poly_solve(Eqn2,X,Soln2,Inter-Diff).

poly_solve(Lhs=Rhs,X,Soln,[Infer,Mult|Rules]-Diff) :-
        poly_norm(Lhs + -1*Rhs,X,Plist),
        poly_tidy(Plist,Qlist),
        cond_poly_print(Lhs + -1*Rhs,X,Qlist,Infer),
        remove_neg_powers(X,Qlist,Poly,Mult),          % Remove negative powe
        poly_method(X,Poly,Soln,Rules-Diff).

cond_poly_print(Poly,X,Plist,tidy(Pol1)) :-
        make_poly(X,Plist,Pol1),
        tidy(Poly,Pol2),
        not match(Pol1,Pol2),
        !,
        trace('\nPolynomial %t becomes \n\n%t when in normal form',
                                                          [Pol2,Pol1],1).
cond_poly_print(_,_,_,_).

remove_neg_powers(X,Plist,Qlist,multiply(Mult)) :-
        last(polyand(N,_),Plist),
        N < 0,
        !,
        eval(-N,N1),
        map_add_power(N1,Plist,Qlist),
        make_poly(X,Qlist,Poly),
        tidy(X^N1,Mult),
        trace('\nMultiply through by %t to get \n\n%t = 0',[Mult,Poly],1).

remove_neg_powers(_,Plist,Plist,nomult).

/*****************************************/
/* ROUTINES FOR POLYNOMIAL EQUATIONS */
/*****************************************/

/* Identities and unsatisfiable equations */

poly_method(_,[],true,[ident|Diff]-Diff) :- !.  % The polynomial has simplifie

poly_method(X,[Pterm],Ans,[single_term|Diff]-Diff) :-    % Polynomial simplifie
        !,                                                % to a single term
        singleton_method(Pterm,X,Ans).

singleton_method(polyand(0,A),_,true) :-
        simplify(A,B),
        B = 0,
        !.
```

```
singleton_method(polyand(0,_),_,false) :- !.

singleton_method(polyand(_,_),X,X = 0) :- !.

/* LINEAR EQUATIONS */

poly_method(X,Poly,X=Ans,[linear|Diff]-Diff) :-
          linear(Poly),
          !,
          linear_method(Poly,Ans).

linear([polyand(1,_)|_]) :- !.

linear_method([polyand(_,A)|T],Ans) :-   % Handles disguised linear also
          find1(T,B),
          tidy(-B/A,Ans).

find1([polyand(0,B)],B) :- !.
find1([],0) :- !.                     % Shouldn't be needed

%* QUADRATIC EQUATIONS*/

poly_method(X,Poly,Soln,[quadratic|Diff]-Diff) :-
          quadratic(Poly),
          !,
          trace('\nUsing quadratic equation formula\n',1),
          find_coeffs(Poly,A,B,C),
          discriminant(A,B,C,Discr),
          roots(X,A,B,C,Discr,Soln).

quadratic([polyand(2,_)|_]) :- !.

find_coeffs([polyand(2,A)|T],A,B,C) :- find2(T,B,C).

discriminant(A,B,C,Discr) :- tidy(B^2 - 4*A*C,Discr).

roots(X,A,B,_,0,X = Root) :-                              % Only 1 root
          !,
          tidy(-B/(2*A),Root),
        trace('\nThe discriminant is zero, so the single solution is %t = %t\n',
                                                                    [X,Root],1).

roots(X,A,B,C,Discr,X = Root1 # X = Root2) :-
          warn_if_complex(Discr),
          tidy((-B + Discr^(1/2))/(2*A),Root1),
          tidy((-B - Discr^(1/2))/(2*A),Root2),
          trace('\nSolutions are %t = %t and %t = %t\n',[X,Root1,X,Root2],1).

warn_if_complex(Discr)  :-
          eval(Discr < 0),
          !,
          trace('\nRoots are complex',_,1).

warn_if_complex(_).

find2([polyand(1,B),polyand(0,C)],B,C) :- !.
find2([polyand(1,B)],B,0) :- !.
find2([polyand(0,C)],0,C) :- !.
%         find2([],0,0) :- !.     Shouldn't be needed
```

```
/* Polynomial divisible by an integral power of the unknown */

poly_method(X,Plist,X = 0 # Ans,[divide(X^N)|Rules]-Diff) :-
        last(polyand(N,_),Plist),
        N > 0,
        !,
        eval(-N,M),
        map_add_power(M,Plist,Qlist),
        poly_method(X,Qlist,Ans,Rules-Diff).

/* Disguised Linear */

poly_method(X,Poly,Soln,[linear|Rules]-Diff) :-
        disguised_linear(Poly),
        !,
        linear_method(Poly,Ans),
        isolate([1,1],X^N=Ans,Soln,Rules-Diff).

disguised_linear([polyand(_,_),polyand(0,_)]).

/* Disguised polynomial equations  */

poly_method(X,Plist,Ans,Rules-Diff) :-
        poly_hidden(X,Plist,N),         % Disguised polynomial in X^N
        trace('\nThis is a hidden polynomial in %t\n',[X^N],1),
        !,
        map_div_power(N,Plist,Qlist),
        poly_method(X^N,Qlist,Inter,Rules-Laws),
        isolate([1,1],Inter,Ans,Laws-Diff). % Maybe needs poly_isolate

poly_hidden(X,Poly,Gcd) :-
        gcd_powers(Poly,Gcd),
        Gcd > 1,
        !.

/* Special methods for reciprocal polynomial equations
        i.e. those that remain unchanged (w.r.t. roots)
        when unknown is replaced by 1/unknown           */

poly_method(X,Poly,X = -1 # Ans,[divide(X + 1)|Rules]-Diff) :-
        odd_symmetric(Poly),
        trace('\nPolynomial is odd-symmetric so %t + 1 is a factor\n',[X],1),
        !,
        factor_out(Poly,1,Plist),
        z_norm(Plist,Qoly),
        poly_method(X,Qoly,Ans,Rules-Diff).

poly_method(X,Poly,X = 1 # Ans,[divide(X - 1)|Rules]-Diff) :-
        odd_anti_symmetric(Poly),
        trace('\nPolynomial is odd anti-symmetric so %t - 1 is a factor\n',
                                                       [X],1),
        !,
        factor_out(Poly,-1,Plist),
        z_norm(Plist,Qoly),
        poly_method(X,Qoly,Ans,Rules-Diff).

poly_method(X,Poly,X = 1 # X = -1 # Ans,[divide(X^2 - 1)|Rules]-Diff) :-
        even_anti_symmetric(Poly),
trace('\nPolynomial is even anti-symmetric so %t - 1 and %t + 1 are both facto
                                                       [X],1),
        !,
        factor_out(Poly,-1,Plist),
        factor_out(Plist,1,Qlist),
        z_norm(Qlist,Qoly),
        poly_method(X,Qoly,Ans,Rules-Diff).

poly_method(X,Poly,Ans,Rules-Diff) :-
        even_symmetric(Poly),
        sym_transform(Poly,NewPoly),
        trace('\nPolynomial is symmetric\n',1),
        !,
        poly_method(X+1/X,NewPoly,Soln,Rules-Inter),
        tidy(Soln,NewEqn),
        poly_solve(NewEqn,X,Ans,Inter-Diff).

/* Guess a root,using integers between 9 and -9  */

poly_method(X,Poly,X = Root # Ans,[divide(X - Root)|Rules]-Diff) :-
        guess_list(Poly,Candidates),
        member(Root,Candidates),
        root(Poly,Root),
        !,
        trace('\nBy inspection %t = %t is a solution\n',[X,Root],1),
        eval(-Root,A),
        factor_out(Poly,A,Plist),
        z_norm(Plist,Qoly),
        poly_method(X,Qoly,Ans,Rules-Diff).


% isolate hack until code is reformed
isolate(Posn,Eqn,New,[isolate|L]-L) :- isolate(Posn,Eqn,New).
```

```
%	Press:Chunk.				Updated: 30 August 82
%	Clause removed 19.2.81, modified 28.4.81, 26.5.81, 10.9.81.
%	subst_mess moved to Misc, rest made compilable 12.9.81.

:- public	changeunknown/3,
		changevar/4,
		good_subterm/4.	%	Just so that 'setof' can find it.

:- mode		changeunknown(+, +, -),
		changeunknown1(+, +, -),
		changevar(+, +, +, -),
		good_subterm(+, +, +, -),
		    good_subterm(+, -),
			good_subterm(+, +, -).


/*	There is a non-trivial BUG:
	change of unknown sometimes fails when it should apparently succeed,
	e.g. when solving for x in the equation
		y + x*(x+1)^(-1)*6 + (y+4)*x*(x+1)^(-1)*(-3) = 1
	(this is problem  d2hard  in the Lewis set.)  The problem is due to
	the lack of associativity in the simple matcher, so that the subterm
	x*(x+1)^(-1) actually appears only once in this equation.  Fixing
	this will require extensive reworking of good_subterm/subterm.
*/

%	changeunknown(Eqn, Var, Ans) determines whether there is a suitable subter
%	(Term) of Eqn (which contains the unknown Var) for changing the unknown.
%	The equation is assumed to be in weak normal form.

changeunknown(Lhs=Rhs, Var, Term) :-
	occ(Var, Lhs, N), N > 1,
	setof(Term, good_subterm(Lhs, Var, N, Term), TermSet),
	extreme_term(TermSet, >, Term),!.

%	changevar generates a new variable NewVar and performs the relevant
%	substitution.

changevar(Term, Eqn, New, NewEqn) :-
	identifier(New),
	subst_mess(Term=New, Eqn, NewEqn).

%	find good subterms for the change of unknown method.

good_subterm(Exp, Var, N, Term) :-
	good_subterm(Exp, Term),
	occ(Var, Term, M), M > 0,
	occ(Term, Exp, L), L > 1,
	N is L*M.

	%	good_subterm(Term, Exp) is true when Term is a non-atomic subterm
	%	of Exp.  This enables us to drop the "Term \= Var" requirement in
	%	good_subterm/4.

	good_subterm(Exp, Term) :-
		(	atomic(Exp) ; number(Exp)	), !, fail.
	good_subterm(Exp, Term) :-
		functor(Exp, _, N),
		good_subterm(N, Exp, Term).
```

```
%   good_subterm(N,E,T) <- T is a good subterm of Exp's Nth argumen

    good_subterm(0, Exp, Term) :- !, Term = Exp.
    good_subterm(N, Exp, Term) :-
            arg(N, Exp, Arg),
            good_subterm(Arg, Term).
    good_subterm(N, Exp, Term) :-
            M is N-1, !,
            good_subterm(M, Exp, Term).
```

```
/* FACTOR : Method for factorising equations

                                                Leon
                                                Updated: 8 September 82
*/

% factorise assumes that the conditions necessary for factorisation
% have been met, namely the right hand side of the equation is zero,
% and the left-hand side is a multiplication term.

factorise(Expr,X,Factors,Proof) :-
        decomp(Expr,[*|List]),
        listtoset(List,List1),          % Remove duplicate equations
        div_list(List1,X,Factors,Proof).

div_list([],_,[],[]) :- !.

div_list([Lhs|L],X,Factors,[div(Lhs)|D]) :-
        safe_divisor(X,Lhs),
        !,
        trace('\nDividing through by %t',[Lhs],1),
        div_list(L,X,Factors,D).

div_list([Lhs|L],X,[Lhs|Factors],D) :-
        div_list(L,X,Factors,D).

safe_divisor(X,Term) :-
        free_of(X,Term),
        non_zero(Term),
        !.

zero(Rhs) :- tidy(Rhs,0).

mulbag(A*B).
```

```
/* TRIG.FAC :
First Created: 13.9.81
                                                         Bernard Silver
                                                         Updated: 2 September 82
*/

!- public
                 set_coeff/2,
                 sincos/2,
                 trig_fac/3.


/* Try to solve trig equations of the form A=0,where A contains only
sin and cos and terms in linear form  */

/* Also solves a*cos(x) + b*sin(x) =c ,see  comments on derive/7 */


/* Top Level */
trig_fac(A=C,X,New) :-
        linear_sin_cos(A,X),
        trig_normal_form(X,A,List),
        trigmethod(X,List,Type),
        trigsolve(X,List,C,Type,New),
        !.

linear_sin_cos(A=B,X) :- !,linear_sin_cos(A,X).

linear_sin_cos(A+B,X) :- !,linear_sin_cos(A,X),linear_sin_cos(B,X).

linear_sin_cos(A*B,X) :- !,
        decomp(A*B,[*|List]),
        sublist(contains(X),List,[New]),
        linear_sin_cos(New,X).

linear_sin_cos(Z,X) :- freeof(Z,X),!.

linear_sin_cos(sin(_),_) :- !.

linear_sin_cos(cos(_),_) :- !.


trig_normal_form(X,A,List) :-
        unattract_distribute(X,A,New),
        decomp(New,[+|NewList]),
        maptrigtype(X,NewList,List).

unattract_distribute(X,A,New) :-
        decomp(A,[*|List]),
        !,
        collect_multipliers(X,List,1,Mults,[],[Rest]),
        tidy(Mults,NewMult),
        dist_multiply(NewMult,Rest,New).

unattract_distribute(X,A,New) :-
        decomp(A,[+|New1]),
        !,
        mapunattract_distribute(X,New1,New2),
        recomp(New,[+|New2]).
```

```
unattract_distribute(_,A,A).

mapunattract_distribute(_,[],[]).

mapunattract_distribute(X,[H|New1],[H1|New2]) :-
        unattract_distribute(X,H,H1),
        mapunattract_distribute(X,New1,New2).

collect_multipliers(_,[],Acc,Acc,Acc1,Acc1) :- !.

collect_multipliers(X,[H|T],Acc,Ans,Acc1,Ans1) :-
        freeof(X,H),
        !,
        collect_multipliers(X,T,Acc*H,Ans,Acc1,Ans1).

collect_multipliers(X,[H|T],Acc,Ans,Acc1,Ans1) :-
        collect_multipliers(X,T,Acc,Ans,[H|Acc1],Ans1).


dist_multiply(A,B+C,D+E) :- !,dist_multiply(A,B,D),dist_multiply(A,C,E).

dist_multiply(A,B,C) :- tidy(A*B,C),!.


/* Put each trig term into the form tf(Fun,Mult,Ang,Rest,Coeff)
where Fun is the functor,Ang the angle.Ang is of the form
Coeff*Rest, where Rest contains the unknown,and Coeff is a number.
Mult is the coeff of the trig term.eg 2*sin(3*a*x) becomes
tf(sin,2,3*a*x,a*x,3)   */

maptrigtype(_,[],[]) :- !.
maptrigtype(X,[H|T],[H1|T1]) :- trigtype(X,H,H1),maptrigtype(X,T,T1),!.

trigtype(Unk,X,tf(Fun,1,Ang,Rest,Coeff)) :- trigf(X),
        functor(X,Fun,1),
        arg(1,X,Ang),
        mod_anglesize(Unk,Rest,X,Coeff),!.

trigtype(Unk,A,tf(Fun,Y,Ang,Rest,Coeff)) :- match(A,X*Y),
        trigf(X),
        freeof(Unk,Y),
        functor(X,Fun,1),
        arg(1,X,Ang),
        mod_anglesize(Unk,Rest,X,Coeff),!.


/* Classify the equation into three types: Does it contain only two terms
or does it contain only sin (or cos) terms whose angles are in A.P.,or
is it a mixture of sines and cosines */
trigmethod(_,List,two(norm)) :- length(List,2),!.
trigmethod(X,List,ap(A,D)) :-  checklist(sincos(Type),List),
        apcheck(X,List,A,D),
        !.

trigmethod(_,List,mixed(Sins,Cos)) :- sublist(sincos(sin),List,Sins),
        sublist(sincos(cos),List,Cos),
        length(Cos,M),
        M>0,
        length(Sins,N),
        N>0,
        M+N>2,
        !.


/* Eqn is A=0 where A is C*sin(X) + C*sin(Y),or C*cos(X) + C*cos(Y),
or C*sin(X)-C*sin(Y),or C*cos(X)-C*cos(Y)        */

trigsolve(_,[tf(Y,N,Ang1,R1,Co1),tf(Y,M,Ang2,R2,Co2)],0,two(X),Newform=0) :-
        (M=N;eval(-M,N)),
        add_angle(Y,N,M,Ang1,Ang2,R1,R2,Co1,Co2,Newform),
        (X=norm ->
        trace('\nUsing trigonometric addition\n %t = 0\n',[Newform],1);
        true),
        !.

/* Both terms have the same angle  */
trigsolve(_,[tf(Y,N,Ang,_,_),tf(Z,M,Ang,_,_)],C,two(norm),Newform) :-
        derive(C,Y,Z,M,N,Ang,Newform),
        trace('\n%t \n',[Newform],1),
        !.

/* Two terms have different functors and angles,but same coeff */
trigsolve(X,[tf(Y,M,Ang1,R1,Co1),tf(Z,N,Ang2,R2,Co2)],0,two(norm),Newform) :-
        (M=N;eval(-M,N)),
        ((Y=sin,Z=cos) -> M1=M,N1=N;
        Y=cos,Z=sin,N1=M,M1=N),
        convert_functor(X,M1,N1,Ang1,Ang2,R1,Co1,Newform),
        !.

/* AP case  */
trigsolve(_,List,0,ap(A,D),New1) :- length(List,N),
        odd(N),
        checkpairs(List,A,D,N,Term1+Term2),
        tidy(Term1+Term2=0,New),
        trace('\nAdding in pairs\n%t\n',[New],1),
        try_factorize(apcase,Term1,Term2,New1),
        !.

/* Mixed case */
trigsolve(X,List,0,mixed(Sin,[A]),Final) :-
        trigsolve(X,Sin,0,two(mixed),New1=0),
        inv_trigtype(A,Term),
        tidy(New1 +Term =0,New),
        trace('\nAdding sin terms \n%t \n',[New],1),
        try_factorize(addone,New1+Term,Final),
        !.

trigsolve(X,List,0,mixed([A],Cos),Final) :-
        trigsolve(X,Cos,0,two(mixed),New1=0),
        inv_trigtype(A,Term),
        tidy(New1 +Term =0,New),
        trace('\nAdding cosine terms \n%t \n',[New],1),
        try_factorize(addone,New1+Term,Final),
        !.

trigsolve(X,List,0,mixed(Sin,Cos),Final) :-
        trigsolve(X,Sin,0,two(mixed),New1=0),
        trigsolve(X,Cos,0,two(mixed),New2=0),
        tidy(New1 + New2 =0,New),
        trace('\nAdding sin terms and cos terms\n%t \n',[New],1),
        try_factorize(addboth,New1+New2,Final),
        !.

  % Do some factorization, to take the load off collection
  % This is hacky, should be done using tf/5 representation.

try_factorize(apcase,Term1,Term2,New) :-
        match(Term1,Fac*B),
        not atomic(Fac),
        match(Term2,Fac*C),
        tidy(Fac*(B+C)=0,New),
        trace('\n%t\n',[New],1),
        !.
try_factorize(apcase,Term1,Term2,New) :-

        match(Term1,Term2*B),
        not atomic(Term2),
        tidy(Term2*(Term1+1)=0,New),
        trace('\n%t\n',[New],1),
        !.

try_factorize(apcase,_,_) :- !,fail.

try_factorize(addone,A+B*G,F1=0) :-
        match(B*G,C*D),
        not atomic(C),
        match(A,C*E),
        tidy(C*(D+E),F1),
        trace('\n%t\n',[F1=0],1),
        !.

try_factorize(addone,A+B,F1=0) :-
        match(A,C*B),
        not atomic(B),
        tidy(B*(C+1),F1),
        trace('\n%t\n',[F1=0],1),
        !.

try_factorize(addboth,A+B*G,F1=0) :-
        match(B*G,C*D),
        not atomic(C),
        match(A,C*E),
        tidy(C*(D+E),F1),
        trace('\n%t\n',[F1=0],1),
        !.

try_factorize(_,Old,Old=0) :- !.

/* Sum of two sines case  */
add_angle(sin,N,N,A1,A2,R1,R2,Coeff1,Coeff2,New) :-
        eval(N*2,N1),
        sumdiff(Coeff1,Coeff2,A1,A2,R1,R2,Sum,Diff,Sum1,Diff1),
        correct_sin(Sum,Sum1,Newsum,Fac,R1,R2),
        correct_cos(Diff,Diff1,Newdiff,R1,R2),
        tidy(Fac*N1*sin(Newsum)*cos(Newdiff),New),
        !.

/* Difference of two sines cases */
add_angle(sin,N,M,A1,A2,R1,R2,Coeff1,Coeff2,New) :-
        eval(N>0),
        eval(N*2,N1),
        sumdiff(Coeff1,Coeff2,A1,A2,R1,R2,Sum,Diff,Sum1,Diff1),
        correct_sin(Diff,Diff1,Newdiff,Fac,R1,R2),
        correct_cos(Sum,Sum1,Newsum,R1,R2),
        tidy(Fac*N1*sin(Newdiff)*cos(Newsum),New),
        !.

add_angle(sin,N,M,A1,A2,R1,R2,Coeff1,Coeff2,New) :-
        eval(M>0),
        eval(M*2,N1),
        sumdiff(Coeff2,Coeff1,A2,A1,R2,R1,Sum,Diff,Sum1,Diff1),
        correct_sin(Diff,Diff1,Newdiff,Fac,R1,R2),
        correct_cos(Sum,Sum1,Newsum,R1,R2),
        tidy(Fac*N1*sin(Newdiff)*cos(Newsum),New),
        !.

/* Sum of two cosines */
add_angle(cos,M,M,A1,A2,R1,R2,Coeff1,Coeff2,New) :-
        eval(M*2,N1),
        sumdiff(Coeff1,Coeff2,A1,A2,R1,R2,Sum,Diff,Sum1,Diff1),
        correct_cos(Sum,Sum1,Newsum,R1,R2),
        correct_cos(Diff,Diff1,Newdiff,R1,R2),
        tidy(N1*cos(Newsum)*cos(Newdiff),New),
        !.

/* Difference of two cosines */
add_angle(cos,M,N,A1,A2,R1,R2,Coeff1,Coeff2,New) :-
        eval(M>0),
        eval(M*2,N1),
        sumdiff(Coeff2,Coeff1,A2,A1,R2,R1,Sum,Diff,Sum1,Diff1),
        correct_sin(Sum,Sum1,Newsum,Fac1,R1,R2),
        correct_sin(Diff,Diff1,Newdiff,Fac2,R1,R2),
        tidy(N1*Fac1*Fac2*sin(Newsum)*sin(Newdiff),New),
        !.

add_angle(cos,M,N,A1,A2,R1,R2,Coeff1,Coeff2,New) :-
        eval(N>0),
        eval(N*2,N1),
        sumdiff(Coeff1,Coeff2,A1,A2,R1,R2,Sum,Diff,Sum1,Diff1),
        correct_sin(Sum,Sum1,Newsum,Fac1,R1,R2),
        correct_sin(Diff,Diff1,Newdiff,Fac2,R1,R2),
        tidy(N1*Fac1*Fac2*sin(Newsum)*sin(Newdiff),New),
        !.



/* Find the half_sum and half_difference of two angles */

/* Angles are of the form A*R and B*R,A and B are numbers */
sumdiff(plus(A,0),plus(B,0),_,_,R,R,Sum,Diff,Sum1,Diff1) :- eval((A+B)/2,Sum1)
        eval((A-B)/2,Diff1),
        tidy(Sum1*R,Sum),
        tidy(Diff1*R,Diff),
        !.

/* General case */
sumdiff(_,_,A1,A2,_,_,Sum,Diff,Sum,Diff) :- tidy((A1+A2)/2,Sum1),
        tidy((A1-A2)/2,Diff1),
        poly_form(Sum1,Sum),
        poly_form(Diff1,Diff),
        !.

/* Equation is M*sin(Ang)+N*cos(Ang) =0,so tan(Ang)=-N/M  */
derive(0,sin,cos,M,N,Ang,tan(Ang) = K) :- eval((-N)/M,K),!.
derive(0,cos,sin,N,M,Ang,tan(Ang) = K) :- eval((-N)/M,K),!.

/* Equation is M*sin(Ang)+N*cos(Ang) = C,so (M^2+N^2)sin(Ang+Beta) = C,
where beta is arctan(N/M)  */
/*At present this is the best place for this rule as:
Should only be used as a collection rule when there are only 2 terms
If homogenization is used on the general case,the simplify routines
get overloaded. */

derive(C,sin,cos,M,N,Ang,New = C) :- eval((M^2+N^2)^(1/2),R),
        eval(arctan(N/M),Beta),
        tidy(R*sin(Ang+Beta),New),
        !.

derive(C,cos,sin,N,M,Ang,New = C) :- eval((M^2+N^2)^(1/2),R),
        eval(arctan(N/M),Beta),
        tidy(R*sin(Ang+Beta),New),
        !.

/* Convert cos(X) to sin(90-X)  */
convert_functor(X,M,M,Ang1,Ang2,R1,Co1,NewE) :-
        tidy((90-Ang2),Newang),
        tidy(M*(sin(Ang1) + sin(Newang))=0,New),
        trace('\nRewriting (cos(X) = sin(90-X)\n%t\n',[New],1),
        mod_angsize1(X,NR,Newang,NC),
        trigsolve1(X,[tf(sin,M,Ang1,R1,Co1),tf(sin,M,Newang,NR,NC)],Co1,NC,New
        !.

convert_functor(X,M,N,Ang1,Ang2,R1,Co1,NewE) :-
        eval(M>0),
        tidy((90-Ang2),Newang),
        tidy(M*(sin(Ang1) - sin(Newang))=0,New),
        trace('\nRewriting (cos(X) = sin(90-X)\n%t\n',[New],1),
        mod_angsize1(X,NR,Newang,NC),
        trigsolve1(X,[tf(sin,M,Ang1,R1,Co1),tf(sin,N,Newang,NR,NC)],Co1,NC,New
        !.

convert_functor(X,N,M,Ang1,Ang2,R1,Co1,NewE) :-
        eval(M>0),
        tidy((90-Ang2),Newang),
        tidy(M*(sin(Newang) - sin(Ang1))=0,New),
        trace('\nRewriting (cos(X) = sin(90-X)\n%t\n',[New],1),
        mod_angsize1(X,NR,Newang,NC),
        trigsolve1(X,[tf(sin,M,Newang,NR,NC),tf(sin,N,Ang1,R1,Co1)],Co1,NC,New
        !.

 % Check equation has not become trivial
trigsolve1(X,List,Coeff1,Coeff2,true)  :-
        tidy(Coeff1,NewC),
        tidy(Coeff2,NewC),
        !,
        trace('\nEquation collapses to 0 = 0\n',1).

trigsolve1(X,List,_,_,Ans) :- trigsolve(X,List,0,two(norm),Ans).

 % Find the coefficient and remainder of the angle
```

```
mod_angsize(X,U,T,Ans) :- arg(1,T,Z),mod_angsize1(X,U,Z,Ans),!.

mod_angsize1(X,U,Z,plus(N,B1)) :- match(Z,A+B),
        contains(X,A),
        freeof(X,B),
        mod_angsize1(X,U,A,plus(N,C)),
        tidy(B+C,B1),
        !.

mod_angsize1(X,U,Z,plus(N,0)) :- match(Z,N*U),
        number(N),
        contains(X,U),
        !.

mod_angsize1(X,Z,Z,plus(1,0)) :- not number(Z),contains(X,Z),!.

sincos(sin,tf(sin,_,_,_,_)) :- !.
sincos(cos,tf(cos,_,_,_,_)) :- !.

checksin_cos1([]) :- !.
checksin_cos1([sin(_)|T]) :- checksin_cos1(T),!.
checksin_cos1([cos(_)|T]) :- checksin_cos1(T),!.

/* AP case  */

apcheck(X,List,A,D) :- maplist(set_coeff,List,Newlist),
        apcheck1(Newlist,A,D),
        trace('\nAngles are in arithmetic progression \n',1),
        !.

set_coeff(tf(_,_,_,_,C),C) :- !.

apcheck1(L,plus(A,0),diff(D,0)) :- non_add(L,L1),
        sort(L1,[A|S]),
        apcheck2([A|S],D),
        !.

apcheck1(L,plus(X,A),diff(0,D)) :- additive_angles(L,L1,X),
        sort(L1,[A|S]),
        apcheck2([A|S],D),
        !.
apcheck2([],_) :- !.
apcheck2([N],_) :- !.
apcheck2([H,H1|T],D) :- eval(H1-H,D),apcheck2([H1|T],D),!.

non_add([],[]) :- !.
non_add([plus(X,0)|S],[X|T]) :- non_add(S,T),!.

additive_angles([],[],_) :- !.
additive_angles([plus(X,A)],[A],X) :- !.
additive_angles([plus(X,A),plus(X,B)|S],[A|T],X) :-
        additive_angles([plus(X,B)|S],T,X),
        !.

checkpairs(List,A,_,1,Term) :-
        member(tf(T,C,Z,Y,A),List),
        inv_trigtype(tf(T,C,Z,Y,A),Term),
        !.
checkpairs(List,plus(A,0),diff(D,0),N,New1+Tail) :-
        member(tf(T,C,Z,Y,plus(A,0)),List),
        eval(A+(N-1)*D,X),
        member(tf(T,C,Z1,Y,plus(X,0)),List),
        add_angle(T,C,C,Z,Z1,Y,Y,plus(A,0),plus(X,0),New1),
        eval(A+D,V),
        eval(N-2,N1),
        checkpairs(List,plus(V,0),diff(D,0),N1,Tail),
        !.
checkpairs(List,plus(A,A1),diff(0,D),N,New1+Tail) :-
        member(tf(T,C,Z,Y,plus(A,A1)),List),
        eval(A1+(N-1)*D,X),
        member(tf(T,C,Z1,Y1,plus(A,X)),List),
        add_angle(T,C,C,Z,Z1,Y,Y1,plus(A,A1),plus(A,X),New1),
        eval(A1+D,V),
        eval(N-2,N1),
        checkpairs(List,plus(A,V),diff(0,D),N1,Tail),
        !.

inv_trigtype(tf(sin,C,Z1,_,_),C*sin(Z1)) :- !.
inv_trigtype(tf(cos,C,Z1,_,_),C*cos(Z1)) :- !.


% Get signs right
correct_sin(Sum,Sum,Sum,Unk,Unk) :- freeof(Unk,Sum),!.
correct_sin(_,Sum,X,F,Unk,Unk) :- number(Sum),correct_sin1(Sum,F,X,Unk),!.
correct_sin(_,Sum,Sum,1,_,_) :- !.

correct_cos(Sum,Sum,Sum,Unk,Unk) :- freeof(Unk,Sum),!.
correct_cos(_,Sum,X,Unk,Unk) :- number(Sum),correct_cos1(Sum,X,Unk),!.
correct_cos(_,Sum,Sum,_,_) :- !.

correct_sin1(Sum,1,New,Unk) :- eval(Sum>0),tidy(Sum*Unk,New),!.
correct_sin1(Sum,(-1),New,Unk) :- tidy(-1*Sum*Unk,New),!.

correct_cos1(Sum,New,Unk) :- eval(Sum>0),tidy(Sum*Unk,New),!.
correct_cos1(Sum,New,Unk) :- tidy(-1*Sum*Unk,New),!.
```

```
/* NAS1. :

                                                  Bernard Silver
                                                  Updated: 24 February 82
Created: May 1981
*/

:- public
                findrhs/2,
                nas1/3.

% This method is similar to isolate,it it used to solve equations where
% all occurrences of the unknown are dominated by a function other than =,+,*
% eg sin(x^2+x+1)=(1/2).  Should also remove multiplicative constants.

% Top level
% If equation is of the right type find the position of the dominating
% function and prepare to isolate it
nas1(L=R,X,[1|Pos]) :- L=..[Func|Args],
        nas1ok(Func,Args,X,Pos),
        !.

nas1ok(+,_,_,_) :- !,fail.
nas1ok(*,[A,B],X,[1]) :- contains(X,A),freeof(X,B),!.
nas1ok(*,[B,A],X,[2]) :- contains(X,A),freeof(X,B),!.
nas1ok(*,_,_,_) :- !,fail.
nas1ok(log,[A,B],X,[1]) :- contains(X,A),freeof(X,B),!.
nas1ok(log,[B,A],X,[2]) :- contains(X,A),freeof(X,B),!.
nas1ok(log,_,_,_) :- !,fail.
nas1ok(^,[A,B],X,[1]) :- contains(X,A),freeof(X,B),!.
nas1ok(^,[B,A],X,[2]) :- contains(X,A),freeof(X,B),!.
nas1ok(^,_,_,_) :- !,fail.
nas1ok(_,_,_,[1]) :- !.

% Defensive checking,make sure that no unknowns occur on the right hand
% side of the isolated equation

findrhs(A#B,P) :- !,findrhs(A,C),findrhs(B,D),append(C,D,P).
findrhs(A=B,[B]) :- !.
```

```
/* HOMOG.TOP :

                                                    Bernard Silver
                                                    Updated: 2 September 82
*/



                                    % HOMOGENIZATION  ROUTINE
                                    % NOTE: Requires equation is in
                                    % weak normal form



% Solve case of Homogenization with messages

homog(Eqn,X,New,Term,V,Off) :-
        homog1(Eqn,X,New,Term,V,Off,Homeqn,solve),
        trace('\nRewriting equation in terms of %t\ngives %t\n',[Term,Homeqn],
        trace('\nSubstituting   %t for %t gives\n %t\n',[V,Term,New],1).

% Top Level of Homogenization proper

homog1(Eqn,Unk,Neweqn,Term,V,Offend,Homeqn,Flag) :-
        findtype(Type,Offend),
        trace('\nOffending set is %t\n',[Offend],2),
        anaz(Type,Eqn,Unk,Offend,Term,Flag),
        trace('\nReduced term is %t\n',[Term],2),
        perform_rewrites(Term,Offend,Homeqn,Unk,Type),
        change_the_variable(V,Unk,Homeqn,New).

 % Equation can have Homogenization applied to it
multiple_offenders_set(Eqn=Rhs,Off,X) :-
        parse(Eqn,Off,X),
        length(Off,N),
        !,
        N>1.


 % Rewrite the offenders set and obtain new Homogenized equation
perform_rewrites(Term,Offend,Homeqn,Unk,Type) :-
        rew(Term,Offend,Sub,Unk,Type),
        subs1(Eqn,Sub,Homeqn).

 % Now change the variable, reporting substitutions if neccesary

change_the_variable(V,Unk,Homeqn,New) :-
        report_subs(Unk,Sub),
        identifier(V),
        subst(Term=V,Homeqn,Neweq),
        tidy(Neweq,Neweqn).

% Find the offenders set (ie the terms which prevent the parsing
% of Eqn as a rational equation ) (Assumes Eqn has been tidied
% so no / or - occurs)

parse(Eqn,Set,Unk) :- dl_parse(Eqn,Set1-[],Unk),listtoset(Set1,Set).

dl_parse(A+B,L-L1,Unk) :- !,dl_parse(A,L-L2,Unk),dl_parse(B,L2-L1,Unk).
dl_parse(A*B,L-L1,Unk) :- !,dl_parse(A,L-L2,Unk),dl_parse(B,L2-L1,Unk).
dl_parse(Unk^N,[Unk^N|L]-L,Unk) :- number(N),!.
dl_parse(A^B,L,Unk) :- number(B), !,dl_parse(A,L,Unk).
dl_parse(Unk,[Unk|L]-L,Unk) :- !.
dl_parse(A,L-L,Unk) :- freeof(Unk,A),!.
dl_parse(A,[A|L]-L,Unk) :- !.

% Find the type of the offending set

findtype(trig,L) :- checklist(trigf,L),!.
findtype(log(_),L) :- checklist(logf,L),!.
findtype(genpol,L) :- maplist(genpoly,L,L1),rational_gcd_list(L1,N),!,N \= 1.
findtype(exp,L) :- checklist(expp,L),!.
findtype(hyper,L) :- checklist(hyperf,L),!. %Just hyperbolics
findtype(hyper_exp,L) :- checklist(hypexp,L),!. %Hyperbolics and exponentials
findtype(mixed,_) :- !.

 % Recognizers for each type

trigf(X) :- member(X,[sin(_),cos(_),tan(_),sec(_),cosec(_),cot(_)]),!.
logf(X) :- member(X,[log(_,_)]),!.
hyperf(X) :- member(X,[sinh(_),cosh(_),sech(_),tanh(_),coth(_),cosech(_)]),!.
expp(_^_) :- !.
expp1(e^_) :- !.
hypexp(X) :- (expp1(X);hyperf(X)),!.
genpoly(X,1) :- atom(X),!.
genpoly(X^N,N) :- atom(X),number(N),!.

%  Find which terms are hyperbolic and which are exponential in hyper_exp case
split_case(L,Exp,Hyp) :- split_case1(L,Exp,Hyp),non_trivial([Exp,Hyp]),!.

split_case1([],[],[]) :- !.
split_case1([H|T],[H|A],B) :- expp1(H),!,split_case1(T,A,B).
split_case1([H|T],A,[H|B]) :- hyperf(H),!,split_case1(T,A,B).

%  Check that both occur in this case

non_trivial([]) :- !.
non_trivial([[]|_]) :- !,fail.
non_trivial([_|T]) :- !,non_trivial(T).

% Try to choose reduced term . Arguments of anaz are
% Type of offenders set, Equation, the Unknown, the offenders set
% the reduced term, and a flag to show if the problem is a sim or solve one

 % Trig case, find the gcd of all angles that occur, then choose functor

anaz(trig,Eqn,Unk,Offend,Term,_) :-
        findangle(Unk,Offend,Angle),!,
        anaz1(Eqn,Angle,Offend,Term,Unk,_).

%  Exponential case where terms are of the form a^f(x) where a is the same
%  for all members of the offending set. We find the 'gcd' of the f(x)

anaz(exp,_,Unk,Offend,Base^Power,_) :-
        maplist(expcase1(Base,Rest,Unk),Offend,NewList),
        form(Rest,NewList,Power),!.

%  Other exponential case where terms are of the form a^(c*x+d).
```

```
anaz(exp,_,Unk,Offend,Base^Power,_) :-
        maplist(expcase2(Unk),Offend,NewList),
        coeff_exp(NewList,Base,Rest),
        form1(Unk,Rest,Power),
        !.

%  Normal log case dealing with terms like log(x,4) and log(2,x) in the
%  offenders set.

anaz(log(_),_,Unk,Offend,Term,_) :-
        maplist(laura(Arg2,Unk),Offend,NewList),
        onetest(NewList,Arg1),
        logocc(Arg1,Arg2,X,Offend).


%  Other log case where the logs are converted to base 10.

anaz(log(10),_,Unk,Offend,log(10,Term),_) :-
        checklist(laura1(Unk,Term),Offend),
        !.

 % The generalized polynomial case

anaz(genpol,_,Unk,Offend,U,_) :-
        maplist(genpolcase(Unk),Offend,List1),
        signed(List1,P),
        rational_gcd_list(List1,N),
        eval(P*N,N1),
        form4(Unk,N1,U),
        !.

 % Hyperbolics.  Find the gcd of all the 'angles' as in trig case

anaz(hyper,Eqn,Unk,Offend,Term,Flag) :-
        findangle(Unk,Offend,Angle),
        hyper_find(Eqn,Unk,Offend,Term,Angle,Flag),!.

 % Both exponentials and hyperbolics, find gcd of all angles and powers.

anaz(hyper_exp,_,Unk,Offend,e^Term,_) :-
        split_case(Offend,Exp,Hyper),
        maplist(angle_size(Unk,Rest),Hyper,Angle),
        maplist(expcase1(e,Rest,Unk),Exp,NewList),
        append(Angle,NewList,Newlist1),
        rational_gcd_list(Newlist1,Gcd),
        form1(Rest,Gcd,Term),
        !.

% Choose reduced_term  using simplicity metric

anaz(_,_,Unk,Offend,T,_) :-
        trace('Choosing reduced term via simplicity metric',2),
         reduced_term(Offend,Unk,T).
```

```
/* HOMOG.TRG :

                                                    Bernard Silver
                                                    Updated: 5 September 82
*/
:- public
                anaz1/6,
                angle_size/4,
                cc/2,
                cch/2,
                cosecfind/1,
                cosechp/3,
                cosecp/3,
                cosfind/1,
                coshp/3,
                cosp/3,
                cothp/3,
                cs/2,
                csh/2,
                expcc/4,
                expcs/4,
                expsc/4,
                expss/4,
                exptt/4,
                findangle/3,
                hyper_find/6,
                secfind/1,
                sechp/3,
                secp/3,
                sinfind/1,
                sinhp/3,
                st/2,
                sth/2,
                tanhp/3.

 %   Find gcd of angles in offending set
findangle(Unk,Offend,Angle) :-
        maplist(angle_size(Unk,Rest),Offend,List),
        form(Rest,List,Angle),
        !.

angle_size(Unk,Rest,Term,Coeff) :-
        arg(1,Term,Arg),
        angle_size1(Unk,Rest,Arg,Coeff),
        !.

angle_size1(Unk,Rest,Arg,Coeff) :-
        match(Arg,A+B),
        contains(Unk,A),
        freeof(Unk,B),
        angle_size1(Unk,Rest,A,Coeff),
        !.

angle_size1(Unk,Rest,Arg,Coeff) :-
        match(Arg,Coeff*Rest),
        number(Coeff),
        contains(Unk,Rest),
        !.

angle_size1(Unk,Rest,Other,1) :-
```

```
        not number(Other),
        contains(Unk,Other),
        match(Other,Rest),
        !.

 % Find the reduced term
 % First,see if offending set contains only cos & sin,or sec & tan,
 % or cot & cosec.If so eliminate (ie choose the other as reduced term)
 % the one that occurs to only even powers,if this happens
 % Flag indicates whether sim or solve is the top level

anaz1(Eqn,Ans,Offend,R,X,Flag) :-
        findtype_trig(Type,Offend),
        action(Type,R,Eqn,Ans,X,Flag),
        !.

 % Same case for hyperbolic functions

hyper_find(Eqn,Unk,Offend,Term,A,Flag) :-
        findtype_hyper(Type,Offend),
        action(Type,Term,Eqn,A,Unk,Flag),
        !.

hyper_find(Eqn,_,_,e^A,A,_) :- !. %If first clause fails use e^A as reduced te

 % See if equation needs tan(R) as a reduced term because equation contains
 % the correct functions.

anaz1(Eqn,Ans,Offend,tan(Ans),X,_) :- tantype(Offend,Ans),taneqn(Eqn,X,Ans),!.

 % Otherwise,choose as reduced term the term that occurs most often

anaz1(Eqn,Ans,Offend,R,_,_) :-
        find_common(Offend,Eqn,R1,Ans),
        !,
        makenice(R1,R).

 % If no term occurs more than once,choose according to an order of niceness

anaz1(_,Ans,Offend,R,_,_) :- anaz2(Ans,Offend,R),(R=tan(Ans) -> ! ;true).

 % If resulting equation can't be solved try tan(half_angle) method,when
 % this method is applicable

anaz1(_,Ans,Offend,tan(R),X,_) :-
        maplist(angle_size(X,Rest),Offend,L1),
        ((match(Ans,M*Rest),number(M));M=1),
        half_angle(M,L1,Ans,R,Rest),
        trace('\nTrying tan half-angle method\n',1),
        !.

% Check to see if tan(x/2) method might work

half_angle(M,List,Angle,Angle,_) :-
        eval(2*M,N),
        member(N,List),
        checklist(half_angle_check1(M),List),
        !.

half_angle(M,List,_,A1,Rest) :-
```

```
        checklist(half_angle_check2(M),List),
        form2(M,Rest,A1),
        !.

 % Check to see if a term occurs more than once in the equation

find_common(L1,Eqn,R,Ans) :-
        maplist(nocc(Eqn),L1,L2),
        great_el(L2,Ans),
        Ans>1,
        correspond(R,L1,L2,Ans),
        arg(1,R,x),
        !.

 % Check for sin_cos etc pairs

findtype_trig(sin_cos,Offend) :-
        memberchk(cos(X),Offend),
        memberchk(sin(X),Offend),
        checklist(cs(X),Offend),
        !.

findtype_trig(cosec_cot,Offend) :-
        memberchk(cosec(X),Offend),
        memberchk(cot(X),Offend),
        checklist(cc(X),Offend),
        !.

findtype_trig(sec_tan,Offend) :-
        memberchk(sec(X),Offend),
        memberchk(tan(X),Offend),
        checklist(st(X),Offend),
        !.

 % Hyperbolic cases

findtype_hyper(sinh_cosh,Offend) :-
        memberchk(cosh(X),Offend),
        memberchk(sinh(X),Offend),
        checklist(csh(X),Offend),
        !.

findtype_hyper(cosech_coth,Offend) :-
        memberchk(cosech(X),Offend),
        memberchk(coth(X),Offend),
        checklist(cch(X),Offend),
        !.

findtype_hyper(sech_tanh,Offend) :-
        memberchk(sech(X),Offend),
        memberchk(tanh(X),Offend),
        checklist(sth(X),Offend),
        !.

action(Type,R,Eqn,Ans,X,Flag) :-
        parse2(Eqn,X,Offend),
        action1(Type,R,Offend,Ans,Flag),
        !.

 % If one of pair occurs only to even powers eliminate it
```

```
action1(sin_cos,sin(A),Offend,A,_) :-
        maplist(cosp(A),Offend,L1),
        checklist(even,L1),
        !.

action1(sin_cos,cos(A),Offend,A,_) :-  !.

action1(sec_tan,tan(A),Offend,A,_) :-
        maplist(secp(A),Offend,L1),
        checklist(even,L1),
        !.

action1(sec_tan,sec(A),Offend,A,_) :- !.

action1(cosec_cot,cot(A),Offend,A,_) :-
        maplist(cosecp(A),Offend,L1),
        checklist(even,L1),
        !.

action1(cosec_cot,cosec(A),Offend,A,_) :- !.

 % Hyperbolic cases
action1(sinh_cosh,sinh(A),Offend,A,_) :-
        maplist(coshp(A),Offend,L1),
        checklist(even,L1),
        !.

action1(sinh_cosh,cosh(A),Offend,A,_) :-
        maplist(sinhp(A),Offend,L1),
        checklist(even,L1),
        !.

action1(sinh_cosh,sinh(A),_,A,sim) :- !.  %Only for sim case

action1(sech_tanh,tanh(A),Offend,A,_) :-
        maplist(sechp(A),Offend,L1),
        checklist(even,L1),
        !.

action1(sech_tanh,sech(A),Offend,A,_) :-
        maplist(tanhp(A),Offend,L1),
        checklist(even,L1),
        !.

action1(sech_tanh,tanh(A),_,A,sim) :- !.  %Only for sim case

action1(cosech_coth,coth(A),Offend,A,_) :-
        maplist(cosechp(A),Offend,L1),
        checklist(even,L1),
        !.

action1(cosech_coth,cosech(A),Offend,A,_) :-
        maplist(cothp(A),Offend,L1),
        checklist(even,L1),
        !.

action1(cosech_coth,coth(A),_,A,sim) :- !.  %Only for sim case

 % Check for tan case
tantype([],_) :- !.
tantype([H|T],X) :- tantype1(H,X),!,tantype(T,X).

tantype1(tan(_),_) :- !.
tantype1(cot(_),_) :- !.
tantype1(sec(X),Y) :- match(X,Y),!.
tantype1(cosec(X),Y) :- match(X,Y),!.

taneqn(Eqn,X,Ans) :- parse2(Eqn,X,Offend),check_tan(Offend,Ans),!.

check_tan([],_) :- !.
check_tan([H|T],Ans) :- check_tan1(H,Ans),!,check_tan(T,Ans).

check_tan1(tan(_),_) :- !.
check_tan1(cot(_),_) :- !.
check_tan1(sec(Ans)^N,Ans1) :- integer(N),even(N),match(Ans,Ans1),!.
check_tan1(cosec(Ans)^N,Ans1) :- integer(N),even(N),match(Ans,Ans1),!.

 % Choose reduced term in order of niceness

anaz2(Ans,Offend,sin(Ans)) :-
        member(sin(Ans),Offend),
        member(cosec(Ans),Offend),
        !.

anaz2(Ans,Offend,cos(Ans)) :-
        member(cos(Ans),Offend),
        member(sec(Ans),Offend),
        !.

anaz2(Ans,Offend,cos(Ans)) :-
        member(cos(Ans),Offend),
        member(cos(X),Offend),
        diff(X,Ans),
        !.

anaz2(Ans,Offend,sin(Ans)) :- member(sin(Ans),Offend),!.

anaz2(Ans,Offend,cos(Ans)) :- member(cos(Ans),Offend),!.

anaz2(Ans,Offend,cos(Ans)) :- member(sec(Ans),Offend),!.

anaz2(Ans,Offend,sin(Ans)) :- member(cosec(Ans),Offend),!.

anaz2(Ans,Offend,sin(Ans)) :- some(sinfind,Offend),!.

anaz2(Ans,Offend,cos(Ans)) :- some(cosfind,Offend),!.

anaz2(Ans,Offend,sin(Ans)) :- some(cosecfind,Offend),!.

anaz2(Ans,Offend,cos(Ans)) :- some(secfind,Offend),!.

anaz2(Ans,_,tan(Ans)) :- !.


cs(X,sin(X)) :- !.
cs(X,cos(X)) :- !.
cc(X,cot(X)) :- !.
cc(X,cosec(X)) :- !.
st(X,sec(X)) :- !.
st(X,tan(X)) :- !.
```

```
 % Hyperbolic cases
csh(X,sinh(X)) :- !.
csh(X,cosh(X)) :- !.
cch(X,coth(X)) :- !.
cch(X,cosech(X)) :- !.
sth(X,sech(X)) :- !.
sth(X,tanh(X)) :- !.

sinfind(sin(_)) :- !.
cosfind(cos(_)) :- !.
secfind(sec(_)) :- !.
cosecfind(cosec(_)) :- !.

 % Recognize powers of trig functions in the equation
cosp(Ang,cos(Ang)^N,N) :- integer(N),!.
cosp(Ang,cos(Ang),1) :- !.
cosp(_,_,0) :- !.
secp(Ang,sec(Ang)^N,N) :- integer(N),!.
secp(Ang,sec(Ang),1) :- !.
secp(_,_,0) :- !.
cosecp(Ang,cosec(Ang)^N,N) :- integer(N),!.
cosecp(Ang,cosec(Ang),1) :- !.
cosecp(_,_,0) :- !.

 % Recognize powers of hyperbolic functions in the equation
coshp(Ang,cosh(Ang)^N,N) :- integer(N),!.
coshp(Ang,cosh(Ang),1) :- !.
coshp(_,_,0) :- !.
sinhp(Ang,sinh(Ang)^N,N) :- integer(N),!.
sinhp(Ang,sinh(Ang),1) :- !.
sinhp(_,_,0) :- !.
sechp(Ang,sech(Ang)^N,N) :- integer(N),!.
sechp(Ang,sech(Ang),1) :- !.
sechp(_,_,0) :- !.
tanhp(Ang,tanh(Ang)^N,N) :- integer(N),!.
tanhp(Ang,tanh(Ang),1) :- !.
tanhp(_,_,0) :- !.
cosechp(Ang,cosech(Ang)^N,N) :- integer(N),!.
cosechp(Ang,cosech(Ang),1) :- !.
cosechp(_,_,0) :- !.
cothp(Ang,coth(Ang)^N,N) :- integer(N),!.
cothp(Ang,coth(Ang),1) :- !.
cothp(_,_,0) :- !.

makenice(cosec(X),sin(X)) :- !.
makenice(sec(X),cos(X)) :- !.
makenice(cot(X),tan(X)) :- !.
makenice(X,X) :- !.

 % expss(P,Q,X,T) expresses sin(Z) in terms of sin(X) where Z/X=Q/P
 % expcs expresses cos(Z) in terms of sin(X) etc.    The 4
 % functions are more or less mutually recursive, but expcc does
 % not depend on the others, though they call it

expss(P,P,X,sin(X)) :- !.

expss(P,Q,X,2*sin(X)*(1-sin(X)^2)^(1/2)) :- eval(Q/P=:=2),!.

expss(P,Q,X,(3*sin(X)-4*sin(X)^3)) :- eval(Q/P=:=3),!.
```

```
 % Where Q/P is odd a simple series expansion can be applied
expss(P,Q,X,A) :- eval(Q/P,N),eval(N mod 2,1),!,sinexp(sin(X),N,0,A).

 % sin(Y) = sin((Y-3*X) + 3*X) = sin(3*X)*cos(Y-3*X) + cos(3*X)*sin(Y-3*X)
 % We can now express each of these 4 terms in terms of sin(X) as
 % a recursive step. The 4 terms are A,B,C and D below.

expss(P,Q,X,(A*B+C*D)) :-
        eval(3*P,P1),
        eval(Q-3,Q1),
        expss(P,P1,X,A),
        expcs(P,Q1,X,B),
        expcs(P,P1,X,C),
        expss(P,Q1,X,D),
        !.

 % Similarly for sin in terms of cos
expsc(P,P,X,(1-cos(X)^2)^(1/2)) :- !.

expsc(P,Q,X,2*cos(X)*(1-cos(X)^2)^(1/2)) :-eval(Q/P=:=2),!.

expsc(P,Q,X,(4*cos(X)^2-1)*(1-cos(X)^2)^(1/2)) :- eval(Q/P=:=3),!.

expsc(P,Q,X,(A*B+C*D)) :-
        eval(3*P,P1),
        eval(Q-3,Q1),
        expsc(P,P1,X,A),
        expcc(P,Q1,X,B),
        expcc(P,P1,X,C),
        expsc(P,Q1,X,D),
        !.

 %  cos in terms of sin
expcs(P,P,X,(1-sin(X)^2)^(1/2)) :- !.

expcs(P,Q,X,(1-2*sin(X)^2)) :-eval(Q/P=:=2),!.

expcs(P,Q,X,(1-4*sin(X)^2)*(1-sin(X)^2)^(1/2)) :- eval(Q/P=:=3),!.

expcs(P,Q,X,(A*B-C*D)) :-
        eval(3*P,P1),
        eval(Q-3,Q1),
        expcs(P,P1,X,A),
        expcs(P,Q1,X,B),
        expss(P,P1,X,C),
        expss(P,Q1,X,D),
        !.

 % Series exists for cos in terms of cos
expcc(P,Q,X,Y) :- eval(Q/P,N),cosexp(cos(X),N,0,Y),!.

 % Base case, series complete
cosexp(A,N,R,X) :- eval(2*R,R1),eval(R1+1,R2),(N=R1;N=R2),coeff1(A,N,R,X),!.

 % Recurse
cosexp(X1,N,R,X-(Y)) :- coeff1(X1,N,R,X),eval(R+1,R1),!,cosexp(X1,N,R1,Y).

 % Produce the coefficients for the series, very ugly
```

```
coeff1(Fans,N,R,X*(ZZ)) :-
        fact(R,R1),
        eval(N-2*R-1,N1),
        eval(N-R-1,N2),
        eval(N1+1,N3),
        fact(N2,Z2),
        fact(N3,Z3),
        eval((2^N1*N*Z2)/(R1*Z3),X),
        form4(Fans,N3,ZZ),
        !.

 % The sin expansion for odd Q/P is very similar to cos cos series
sinexp(X,N,A,B*(Z)) :- eval((-1)^((N-1)/2),B),cosexp(X,N,A,Z),!.

 % Expand tan(n*x) in terms of tan(m*x)  (m < n)
 % Tan produces a numerator and denominator series.

exptt(I,J,X,(Z)/(Y)) :-
        eval(J/I,N),
        tanexp_num(tan(X),N,1,Z),
        tanexp_denom(tan(X),N,0,Y),
        !.

 % Obtain numerator
tanexp_num(A,N,R,X) :- eval(R+1,R1),(N=R1;N=R),coeff2(A,N,R,X),!.
tanexp_num(A,N,R,X-(Y)) :-
        coeff2(A,N,R,X),
        eval(R+2,R1),
        !,
        tanexp_num(A,N,R1,Y).


 % Obtain the denominator
tanexp_denom(A,N,R,X) :- eval(R+1,R1),(N=R1;N=R),coeff2(A,N,R,X),!.
tanexp_denom(A,N,R,X-(Y)) :-
        coeff2(A,N,R,X),
        eval(R+2,R1),
        !,
        tanexp_denom(A,N,R1,Y).

 % Different coefficients from the other series

coeff2(A,N,R,X*(ZZ)) :- calc_coeff(N,R,X),form4(A,R,ZZ),!.

calc_coeff(N,R,X) :-
        fact(R,Rfact),
        fact(N,Nfact),
        eval(N-R,P),
        fact(P,Pfact),
        eval(Nfact/(Pfact*Rfact),X),
        !.
```

```
/*      LOG
                                Written by Bernard Silver 19.8.81
                                        Updated: 23 March 82
*/
%declarations%

:- public
                losmethod/4.
%end%

/* The log method is called by solve2. It solves equations of the
form a^f(x)*b^g(x)*...*y^z(x) = a1^f1(x)*b1^g1(x)*...*p1^q1(x)
where a,b,c,a1,b1 do not contain the unknown,x.
        For example the AEB question:
        4^(2*x+1)*5^(x-2)=6^(1-x)
is solved by taking logs base 4 and solving the linear equation */


/*The equation is in weak normal form.The method can be used only if the
equation is of the forms:
1) A+B=0 where A and B do not have + as the dominant functor
or 2)A=B,B is free of the unknown and A is as above  */
logmethod(A+B+C=0,_,_,_) :- !,fail.
logmethod(A+B=0,X,New,Base) :- postidy(A+B=0,N),logmethod(N,X,New,Base),!.
logmethod(A+B=C,_,_,_) :- !,fail.
logmethod(A=B,X,New,Base) :- suitable(A=B,X,Base),takelog(Base,A=B,New),!.

/* Having satisfied the above conditions now check that the terms in A and B
are of the correct type */
suitable(A=B,X,Base) :- suit1(A,X,L2),
        suit1(B,X,L1),
        union(L1,L2,L),
        find_log_base(L,Base),
        !.

suit1(A,X,[]) :- freeof(X,A),!. %If the term is free of x then it is ok
/* A*B is suitable if both A and B are,ie each is of the form C^D
where C is free of x and D contains x */
suit1(A*B,X,L) :- suit1(A,X,L1),suit1(B,X,L2),union(L1,L2,L),!.
suit1(A^B,X,[A]) :- freeof(X,A),contains(X,B),!.

/* The base to take logs to is the smallest of the numbers a,b,c etc if these
are integers,if they are all integers or 1/integer use the smallest integer,
otherwise use base 10  */
find_log_base(L,Base) :- checklist(integer,L),least_el(L,Base),Base \==1,!.
find_log_base(L,Base) :- find_bases(L,Base),!.
find_log_base(_,10) :- !.

find_bases(L,Base) :- find_bases1(L,List),least_el(List,Base).

find_bases1([],[]) :- !.
find_bases1([H|T],[H|R]) :- integer(H),find_bases1(T,R),!.
find_bases1([H|T],[I|R]) :-
        number(H),
        eval(numer(H)=1),
        eval(denom(H),I),
        find_bases1(T,R),
        !.

/* Clauses to take the logs  */
```

```
takelos(Base,A=B,C=D) :- !,takelos(Base,A,C),takelos(Base,B,D).
takelos(Base,A*B,C+D) :- !,takelos(Base,A,C),takelos(Base,B,D).
takelos(Base,A^B,B*C) :- !,takelos(Base,A,C).
takelos(Base,A,los(Base,A)) :- !.

postidy(A+B=0,N) :-
         match(A,C*D),
         number(C),
         eval(C<0),
         eval(-C,C1),
         tidy(B=D*C1,N),
         !.
postidy(A+B=0,N) :- match(B,C*D),
         number(C),
         eval(C<0),
         eval(-C,C1),
         tidy(A=D*C1,N),!.
postidy(A+B=0,N) :- tidy(A= -1*(B),N),!.
```

```
/*                NASTY                       14.9.81
                                              Updated: 6 September 82
*/
%declarations%

:- public
                findbase/2,
                good_fun/1,
                invert_exp/2,
                nasty/2,
                nasty2/2,
                nasty_method/3,
                nice_at/1,
                pt/1,
                pta/1.
%end%


/*              CODE                                         */
/* Nasty in the context of the code and comments means a term u^x where
x is a rational non-integer and u is anything.Here offending term means
the same as it does in homogenization  */


/* Has equation been seen before */
nasty_method(Eqn,X,Ans) :-
        looping(Eqn,X),
        tidy(Eqn,Eqn1),
        try_nasty_method(Eqn1,X,Ans),
        !.

/* Try to deal with non-rational nasty functions  */
try_nasty_method(Eqn,X,Neweqn) :-
        parse4(Eqn,X,U,other),
        subnasty(X,U,V),
        find_symbols(Eqn,V,Symbols,Posns),
        nasty_act(Symbols,Posns,Eqn,X,Neweq),
        tidy(Neweq,Neweqn),
        !.

/* Clear rational functions */
try_nasty_method(Eqn,X,Neweqn) :-
        parse4(Eqn,X,U,neg),
        exp_nasty_list(X,U,V),
        remove_subsumed(V,Termlist),
        multiply_through(Eqn,Termlist,Neweqn,X),
        tidy(Neweqn,New),
        trace('\nClearing of rational functions\n\n%t\n',[New],1),
        !.


/* The isolate case   */

nasty_act(Symbols,[Posn|_],Eqn,X,New) :-
        nice(Symbols),
        append(Posn,[1],Posn1),
        position(Term,Eqn,Posn1),
        trace('\nTrying to isolate %t\n in %t\n',[Term,Eqn],1),
        try_isolate(Posn1,Eqn,New),
        !.
```

```
try_isolate(Posn,Eqn,New) :- isolate(Posn,Eqn,New),!.
try_isolate(_,_,_) :-   writef('\nFailed to isolate\n'),!,fail.

/* The cancelling pair case   */
/* Left to tidy at present  */
/* Eventually we will need rules to cancel sin(arcsin(x)) etc  */

/* Attraction case  */

nasty_act(Symbols,Posns,Eqn,X,New) :-
        find_attract_list(Symbols,N,L,Type),
        nmember(Posn,Posns,N),
        strip(Posn,L,Newp),
        position(Term,Eqn,Newp),
        nas_rule(Term,Nterm,Type),
        subst(Term=Nterm,Eqn,New1),
        tidy(New1,New),
        trace('\nAttracting nasty functions\n%t\n',[New],1),
        !.

parse4(A,Unk,Bag,Type) :- dl_parse4(A,Unk,Bag-[],Type).

dl_parse4(A,Unk,L-L,_) :- freeof(Unk,A),!.
dl_parse4(A=B,Unk,L-L1,T) :- !,
        dl_parse4(A,Unk,L-L2,T),
        dl_parse4(B,Unk,L2-L1,T).

dl_parse4(A*B,Unk,L-L1,T) :- !,
        dl_parse4(A,Unk,L-L2,T),
        dl_parse4(B,Unk,L2-L1,T).

dl_parse4(A+B,Unk,L-L1,T) :- !,
        dl_parse4(A,Unk,L-L2,T),
        dl_parse4(B,Unk,L2-L1,T).

dl_parse4(A^B,Unk,X,other) :- integer(B),B > 0,!,dl_parse4(A,Unk,X,other).
dl_parse4(A,_,[A|L]-L,_) :- !.

/* See if any of the terms found are nasty rather than offending  */

nasty(X,Y) :- root_nasty(X,Y),!.
nasty(X,Y) :- exp_nasty(X,Y),!.
nasty(X,Y) :- trig_nasty(X,Y),!.

/* Root type nasty */
root_nasty(X,U^N) :- contains(X,U),number(N),not integer(N),eval(N>0),!.

/* Negative exponent nasty */
exp_nasty(X,U^N) :- contains(X,U),number(N),eval(N<0),diff(X,U),!.

exp_nasty_list(_,[],[]) :- !.
exp_nasty_list(X,[H|Rest],[H|RestV]) :-
        exp_nasty(X,H),
        !,
        exp_nasty_list(X,Rest,RestV).
exp_nasty_list(X,[_|Rest],RestV) :-
        exp_nasty_list(X,Rest,RestV).
```

```
trig_nasty(X,Y) :- (arctrigf(Y);trigf(Y)),contains(X,Y),!.

/* Find the functions dominating,and the positions of,the nasty functions */
find_symbols(_,[],[],[]) :- !.

        find_symbols(E,[H|T],[H1|T1],[H2|T2]) :- find_symbols1(E,H,H1,H2),
        find_symbols(E,T,T1,T2),
        !.

find_symbols1(Eqn,X,Y,B) :-
        pos1(X,Eqn,A,B),
        expon(X,P),
        append(A,[P],Y),
        !.

pos1(X,X,[],[]) :- !.
pos1(X,E,[Op|L],[N|Pos]) :-
        E=..[Op1,Arg|Args],
        set_ops(Op,Op1,E),
        nmember(T,[Arg|Args],N),
        pos1(X,T,L,Pos),
        !.

set_ops(exp(Arg1),_,E) :- E=..[^,_,Arg1|_],!.
set_ops(Op1,Op1,_) :- !.

expon(U^N,exp(N)) :- number(N),!.
expon(X,X) :- arctrigf(X),!.
expon(X,X) :- trigf(X),!.

/* Remove terms form list if they are subsumed by others,ie
if U^-N and U^-M,M>N both occur keep only U^-M     */
remove_subsumed([],_) :- !, fail.
remove_subsumed(V,Termlist) :-
        listtoset(V,List),                  % Cheap test
        rem_sub(List,Termlist,[]).

rem_sub([],Termlist,Termlist) :- !.
rem_sub([H|Rest],Termlist,Acc) :-
        member_match(H,Acc,NewAcc),
        !,
        rem_sub(Rest,Termlist,NewAcc).
rem_sub([H|Rest],Termlist,Acc) :-
        match(H,U^N),
        number(N),
        rem_sub(Rest,Termlist,[U^N|Acc]).

member_match(H,[],_) :- !, fail.
member_match(H,[U^N|Rest],[U^K|Rest]) :-
        match(H,U^M),
        !,
        least(N,M,K).
member_match(H,[Term|Rest],[Term|NewRest]) :-
        member_match(H,Rest,NewRest).

least(N,M,N) :- eval(N=<M), !.
least(N,M,M).

/* Is the function dominating list nice,ie can isolation be used  */
nice([]) :- !.
nice([List|Rest]) :-
        nice_list(List),
        !,
        nice(Rest).

nice_list([]) :- !.
nice_list([Fun|Rest]) :-
        good_fun(Fun),
        !,
        nice_list(Rest).

/* Isolatable functions (need to add arctrig etc) */
good_fun(+) :- !.
good_fun(=) :- !.
good_fun(*) :- !.
good_fun(X) :- arctrigf(X),!.
good_fun(exp(N)) :- number(N),not integer(N),eval(numer(N)=1),!.

/* Is the function dominating list attractable */
find_attract_list([],_,_,_) :- !,fail.
find_attract_list([H|T],1,M,Type) :- attract_list(H,M,Type),!.
find_attract_list([_|T],N,M,Type) :-
        find_attract_list(T,N1,M,Type),
        N is N1+1,
        !.

attract_list([exp(N)|T],K,Type) :-
        integer(N),
        last(exp(M),T),
        set_nasty_type(M,N,Type),
        append(T1,[exp(M)],T),
        checkpt(T1),
        length(T,K),
        !.
attract_list([X|T],K,trig) :- trigf(X),checkpta(T),length(T,K),!.

attract_list([_|T],M,Type) :- attract_list(T,M,Type),!.


set_nasty_type(M,N,root(M)) :- eval(1/N,M),!.
set_nasty_type(M,N,negroot(M)) :- eval(1/N,-1*M),!.
set_nasty_type(M,N,neg(M)) :- eval(M<0),!.

pt(*) :- !.
pt(+) :- !.

pta(X) :- pt(X),!.
pta(X) :- arctrigf(X),!.

arctrigf(X) :- member(X,[arcsin(_),arccos(_),arctan(_)]),!.
/* Attraction Rules (many to be added)   */

nas_rule(A^2 ,Exp,root(N)) :- dist(A,A1),tidy(A1,A2),expon_exp(A2^2,N,Exp),!.
nas_rule(A^2,Exp,negroot(N)) :-
        dist(A,A1),
        tidy(A1,A2),
        expon_inv_exp(A2^2,N,Exp),
        !.
nas_rule(A^2,Exp,neg(N)) :- neg_exp(A^2,N,Exp),!.
nas_rule(sin(X),Exp,trig) :- sinatt(X,Exp),!.
nas_rule(cos(X),Exp,trig) :- cosatt(X,Exp),!.
nas_rule(tan(X),Exp,trig) :- tanatt(X,Exp),!.

expon_exp(Old,N,New) :- eval(N=(1/2)),expon_exp1(Old,N,New),!.

expon_inv_exp(Old,N,New) :- eval(N=(-1/2)),expon_inv_exp1(Old,N,New),!.

expon_exp1(A^2,N,C^2 + 2*C*D^N + D) :- match(A,D^N+C),!.
expon_exp1(A^2,N,C^2 + 2*C*E*D^N + D*E^2) :- match(A,C+E*D^N),!.
expon_exp1(A^2,N,C^2*D) :- match(A,C*D^N),!.

expon_inv_exp1(A^2,N,C^2 + 2*C*D^N + D^(-1)) :- match(A,D^N+C),!.
expon_inv_exp1(A^2,N,C^2 + 2*C*E*D^N + D^(-1)*E^2) :- match(A,C+E*D^N),!.
expon_inv_exp1(A^2,N,C^2*D^(-1)) :- match(A,C*D^N),!.

neg_exp(A^2,N,A^2) :- wordsin(A,L),L=[],!.
neg_exp(A^2,N,X*Y) :- match(A,B*C),!,neg_exp(B^2,N,X),neg_exp(C^2,N,Y).
neg_exp(A^2,N,B^E+2*C*B^N +C^2) :-
        match(A,B1+C),
        neg_exp_match(B1,F,B,N),
        eval(2*N,E),
        !.
neg_exp(A^2,_,A^2) :- !.


neg_exp_match(Exp,1,B,N) :- match(Exp,B^N),!.
neg_exp_match(Exp,F,B,N) :- match(Exp,F*B^N),!.

sinatt(X,Exp) :- match(X,(-1)*Y),sinatt(Y,E1),tidy((-1)*E1,Exp),!.
sinatt(A+B,Exp) :-
        trig_inv(sin(A),X,F1),
        trig_inv(cos(A),Y,F2),
        trig_inv(sin(B),Z,F3),
        trig_inv(cos(B),W,F4),
        member(cancel,[F1,F2,F3,F4]),
        merge(X*W,X1),
        merge(Y*Z,X2),
        tidy(X1 + X2,Exp),
        !.

cosatt(X,Exp) :- match(X,Y*(-1)),cosatt(Y,Exp),!.
cosatt(A+B,Exp) :-
        trig_inv(sin(A),X,F1),
        trig_inv(cos(A),Y,F2),
        trig_inv(sin(B),Z,F3),
        trig_inv(cos(B),W,F4),
        member(cancel,[F1,F2,F3,F4]),
        merge(W*Y,X1),
        merge(Z*X,X2),
        tidy(X1 - X2,Exp),
        !.

tanatt(X,Exp) :- match(X,Y*(-1)),cosatt(Y,Exp1),tidy((-1)*Exp1,Exp),!.
tanatt(A+B,Exp) :-
        trig_inv(tan(A),X,F1),
        trig_inv(tan(B),Y,F2),
        member(cancel,[F1,F2]),
        merge(X*Y,Z),
        tidy((X+Y)/(1-Z),Exp),
        !.
```

```
trig_inv(sin(X),Y,F) :- match(X,(-1)*Z),trig_inv(sin(Z),W,F),tidy((-1)*W,Y),!.
trig_inv(sin(arcsin(X)),X,cancel) :- !.
trig_inv(sin(arccos(X)),Y,cancel) :- tidy((1-X^2)^(1/2),Y),!.
trig_inv(sin(X),sin(X),no) :- !.

trig_inv(cos(X),Y,F) :- match(X,(-1)*Z),trig_inv(cos(Z),Y,F),!.
trig_inv(cos(arccos(X)),X,cancel) :- !.
trig_inv(cos(arcsin(X)),Y,cancel) :- tidy((1-X^2)^(1/2),Y),!.
trig_inv(cos(X),cos(X),no) :- !.

trig_inv(tan(X),Y,F) :- match(X,(-1)*Z),trig_inv(tan(Z),W,F),tidy((-1)*W,Y),!.
trig_inv(tan(arctan(X)),X,cancel) :- !.
trig_inv(tan(X),tan(X),no) :- !.

/* strip(L,M,L1) holds when removing the last M elements from list L
gives list L1 */
strip(L,N,L1) :- append(L1,List,L),length(List,N),!.

/* Do the multiplication to rationalize  */
multiply_through(Lhs=Rhs,List,New,X) :-
        dist(Lhs,Exp),
        decomp(Exp,[+|L]),
        mult(List,L,NewL),
        recomp(NewLhs,[+|NewL]),
        free_mult(List,Rhs,NewRhs),
        weaknf(NewLhs=NewRhs,X,Left=Right),
        tidy(Left=Right,New),
        !.

dist(Old,New) :- prepd(Old,New1),dist1(New1,New),!.

dist1(A+B,C+D) :- !,dist1(A,C),dist1(B,D),!.
dist1((A+B)*C,Y + Z) :- !,dist1(A*C,Y),dist1(B*C,Z).
dist1(C*(A+B),Y+Z) :- !,dist1(A*C,Y),dist1(B*C,Z).
dist1(C*(A+B)*D,Y+Z) :- !,dist1(C*D*A,Y),dist1(C*D*B,Z).
dist1(X,X) :- !.

prepd(X,Y) :- decomp(X,[*|L]), prepd1(L,Y),!.
prepd(X,X) :- !.

prepd1(L,Y) :- set_dist(L,Mult,[],Plus),re_dist(Mult,Plus,Y),!.

set_dist([],_,_,_) :- !,fail.
set_dist([A+B|T],Prod,Acc,A+B) :- !,append(T,Acc,Prod1),recomp(Prod,[*|Prod1])
set_dist([H|T],Ans,Acc,Plus) :- !,append([H],Acc,Newacc),
set_dist(T,Ans,Newacc,Plus).


re_dist(M1,P+Q,X+Y) :- prepd(M1*P,X),prepd(M1*Q,Y),!.


mult(Termlist,[],[]) :- !.
mult(Termlist,[H|Rest],[NewH|NewRest]) :-
        domult(Termlist,H,NewH),
        mult(Termlist,Rest,NewRest).

domult(Termlist,H,NewH) :-
        mulbag_to_list(H,Mullist),
        domult(Termlist,Mullist,NewH,1).
domult([],Args,Term*Acc,Acc) :-
        !,
        recomp(Term,[*|Args]).
domult([U^N|Rest],Args,Prod,Acc) :-
        exp_member(U,Args,NewArgs,K),
        eval(K-N,M),
        domult(Rest,NewArgs,Prod,U^M*Acc),
        !.

mulbag_to_list(H,Mullist) :- decomp(H,[*|Mullist]), !.
mulbag_to_list(H,[H]).

exp_member(U,[],[],0) :- !.
exp_member(U,[H|Rest],Rest,1) :- match(H,U), !.
exp_member(U,[H|Rest],Rest,K) :- match(H,U^K),eval(K<0), !. %fix???
exp_member(U,[H|Rest],[H|NewRest],K) :-
        exp_member(U,Rest,NewRest,K).

free_mult(List,0,0) :- !.
free_mult([],Term,Term) :- !.
free_mult([U^N|Rest],Term,NewTerm) :-
        eval(-N,M),
        free_mult(Rest,U^M*Term,NewTerm).

/* Looping Check */
looping(Eqn,X) :- normstore(Eqn,X,Eqn1),looping1(Eqn1),!.

looping1(Eqn1)  :- seen_eqn(Eqn1),
        !,
        trace('\n*****LOOPING*****\nI have seen equation before\n',1),
        trace('\nTracing\n',1),
        cond_trace,
        fail.

looping1(Eqn1) :- asserta(seen_eqn(Eqn1)),!.

normstore(Eqn,X,Eq) :- subst(X  = unk,Eqn,Eqn1),
        !,
        remove_arbs(Eqn1,Eqn2),
        tidy(Eqn2,Eq).


/* Remove arbitrary integers */

remove_arbs(Eqn1,Eqn2) :-
        wordsin(Eqn1,Words),
        subintegral(Words,Word),
        remove_arbs1(Eqn1,Word,Eqn2),
        !.

remove_arbs1(X,[],X) :- !.
remove_arbs1(X,H,Y) :- make_arblist(H,Z),make_subl(H,Z,Y1),subst(X,Y1,Y),!.

make_arblist(H,Z) :- make_arblist1(H,Z,1),!.

make_arblist1([],[],_) :- !.
make_arblist1([H|T],[arb(N)|T1],N) :- M is N+1,make_arblist1(T,T1,M),!.

cond_trace :- flag(tflag,N,N),N>0,trace,!.
cond_trace :- !.

seen_eqn(_) :- fail.

integral(_) :- fail.

/* Merge roots in products      */
merge(A,X) :- eval(1/2,N),match(A,B^N*C^N),tidy(B*C,Y),tidy(Y^N,X),!.
merge(A,X) :- eval(1/2,N),match(A,Z*B^N*C^N),tidy(B*C,Y),tidy(Z*Y^N,X),!.
merge(A,A) :- !.

 % Converted sublists etc

subnasty(_,[],[]) :- !.
subnasty(X,[H|T],[H|T1]) :- nasty(X,H),!,subnasty(X,T,T1).
subnasty(X,[_|T],T1) :- subnasty(X,T,T1),!.

subintegral([],[]) :- !.
subintegral([H|T],[H|T1]) :- integral(H),!,subintegral(T,T1).
subintegral([_|T],T1) :- subintegral(T,T1),!.

checkpt([]) :- !.
checkpt([H|T]) :- pt(H),checkpt(T),!.

checkpta([]) :- !.
checkpta([H|T]) :- pta(H),checkpta(T),!.
```

```
/* SIMP.AX : Simplification axioms for TIDY

                                                    Bernard Silver
                                                    Updated: 13 May 82
*/

% PUBLIC
:- public simplify_axiom/2.

% MODES

:- mode simplify_axiom(+,-).
% Logs
simplify_axiom(log(U,U^V),V).
simplify_axiom(log(A,1),0).
simplify_axiom(U^log(U,V),V).

% Normalize square roots
simplify_axiom(sqrt(U),U^number(+,[1],[2])).

% Trig cancelling pairs
simplify_axiom(cos(arccos(X)),X).
simplify_axiom(arccos(cos(X)),X).

simplify_axiom(arcsin(sin(X)),X).
simplify_axiom(sin(arcsin(X)),X).

simplify_axiom(tan(arctan(X)),X).
simplify_axiom(arctan(tan(X)),X).

simplify_axiom(sec(arcsec(X)),X).
simplify_axiom(arcsec(sec(X)),X).

simplify_axiom(cosec(arccosec(X)),X).
simplify_axiom(arccosec(cosec(X)),X).

simplify_axiom(cot(arccot(X)),X).
simplify_axiom(arccot(cot(X)),X).

% Hyperbolic cancelling pairs
simplify_axiom(sinh(arcsinh(X)),X).
simplify_axiom(arcsinh(sinh(X)),X).

simplify_axiom(cosh(arccosh(X)),X).
simplify_axiom(arccosh(cosh(X)),X).

simplify_axiom(tanh(arctanh(X)),X).
simplify_axiom(arctanh(tanh(X)),X).

simplify_axiom(sech(arcsech(X)),X).
simplify_axiom(arcsech(sech(X)),X).

simplify_axiom(cosech(arccosech(X)),X).
simplify_axiom(arccosech(cosech(X)),X).

simplify_axiom(coth(arccoth(X)),X).
simplify_axiom(arccoth(coth(X)),X).

% Common trig cases
simplify_axiom(sin(arccos(X)),(1-X^2)^(1/2)#(1-X^2)^(1/2)*(-1)).
simplify_axiom(cos(arcsin(X)),(1-X^2)^(1/2)#(1-X^2)^(1/2)*(-1)).

simplify_axiom(arcsin(cos(X)),90-X).

simplify_axiom(arccos(sin(X)),90-X).
```

```
/*               ISOLAT.AX
                                                   Updated: 10 August 82
*/

:- public
                isolax/4.

/* AXIOMS FOR ISOLATION*/
/* FIRST ARGUMENT IS THE VARIABLE ISOLATED*/

/* unary minus */
isolax( 1 , -U=V , U= -1*V , true ).

/* plus */
isolax( 1 , U+V=W , U=W+(-1)*V , true ).
isolax( 2 , V+U=W , U=W+(-1)*V , true ).

/* multiplication */
isolax( 1 , U*V=W , U=W*V1 , non_zero(V) ) :- tidy(1/V,V1).
isolax( 2 , V*U=W , U=W*V1 , non_zero(V) ) :- tidy(1/V, V1).

/* logarithms */
isolax( 1 , log(U,1)=0 , U=N , arbint(N) ).
isolax( 1 , log(U,V)=W , U=V^W1 , non_zero(W) ) :- tidy(1/W,W1).
isolax( 2 , log(U,V)=W , V=U^W , true ) .

/* exponentiation */

isolax( 1 , U^0 = K , U=N , arbint(N) ) :-  K=1,!.

isolax( 1 , U^0=N,false,true) :- eval(N\= -1),
         trace('\nThe equation %t^0 = %t has no real roots\n',[U,N],1),
         trace('\n%t^0  must equal 1.\n',[U],1),
         !.

isolax( 1 , U^N = 0 , false , true ) :- negative(N),
trace('\n%t^%t = 0 has no real roots, %t^%t can not be 0\n',[U,N,U,N],1),
        !.

isolax( 1 , U^N=V , U=V^N1 , odd(N) ) :- tidy(1/N, N1).

isolax( 1 , U^N=V , false , true ) :- negative(V),
        integer(N),
        even(N),
        tidy(1/N,N1),
        trace('\nThe equation %t^%t = %t has no real roots\n',[U,N,V],1),
        trace('\nas %t^%t is not real\n',[V,N1],1),
        !.


isolax( 1 , U^N=V , U=V^N1 , non_neg(U) & even(N) ) :- tidy(1/N, N1).

isolax( 1 , U^N=V , U=V^N1 # U=(-1)*(V^N1) , even(N) ) :- tidy(1/N, N1).

isolax( 1 , U^A=V, U=V^A1 , not number(A) ) :- tidy(1/A,A1).

isolax( 2 , U^V=W , false , true ) :- positive(U),
        eval(W=<0),
trace('\n%t^%t = %t has no real roots, %t^%t must be > 0\n',[U,V,W,U,V],1),
        !.
```

```
isolax( 2 , U^V=W , V=log(U,W) , true ) .

/* sine */
isolax( 1, sin(U)=V,false,true) :- (eval(V>1);eval(V< -1)),
trace('\n%t=%t has no real roots, sin must lie in [-1,1]\n',[sin(U),V],1),
        !.

isolax( 1 ,sin(U)=V,U=arcsin(V), acute(U)).

isolax( 1 ,sin(U)=V,U=arcsin(V)#U=180+((-1)*arcsin(V)), non_reflex(U)).

isolax( 1 , sin(U)=V , U=N*180+ (-1)^N*arcsin(V) , arbint(N) ) .

/* cosine */
isolax( 1, cos(U)=V,false,true) :- (eval(V>1);eval(V< -1)),
trace('\n%t=%t has no real roots, cos must lie in [-1,1]\n',[cos(U),V],1),
        !.

isolax( 1 ,cos(U)=1,U=360*N,arbint(N)).

isolax( 1 ,cos(U)=V,U=arccos(V), non_reflex(U)).

isolax( 1 , cos(U)=V , U=2*N*180+arccos(V) #
                           U=2*N*180+ ((-1)*arccos(V)) , arbint(N) ) .

/* tangent */
isolax( 1 , tan(U)=V , U=N*180+arctan(V) , arbint(N) ) .

/* cosecant */
isolax( 1 , cosec(U)=V , U=N*180+ (-1)^N*arccosec(V) ,
                             arbint(N) ) .

/* secant */
isolax( 1 , sec(U)=V , U=2*N*180+arcsec(V) #
                           U=2*N*180+ ((-1)*arcsec(V)) , arbint(N) ) .

/* cotangent */
isolax( 1 , cot(U)=V , U=N*180+arccot(V) , arbint(N) ) .

/* inverse sine */
isolax( 1, arcsin(U)=V,false,true) :- (eval(U>1);eval(U< -1)),
        tidy(V,V1),
trace('\n%t=%t has no real roots, sin must lie in [-1,1]\n',[arcsin(U),V1],1),
        !.

isolax( 1 , arcsin(U)=V , U=sin(V) , true ) .

/* inverse cosine */
isolax( 1, arccos(U)=V,false,true) :- (eval(U>1);eval(U< -1)),
        tidy(V,V1),
trace('\n%t=%t has no real roots, cos must lie in [-1,1]\n',[arccos(U),V1],1),
        !.

isolax( 1 , arccos(U)=V , U=cos(V) , true ) .

/* inverse tangent */
isolax( 1 , arctan(U)=V , U=tan(V) , true ) .

/*inverse cosecant */
```

```
isolax( 1 , arccosec(U)=V , U=cosec(V) , true ) .

/* inverse secant */
isolax( 1 , arcsec(U)=V , U=sec(V) , true ) .

/* inverse cotangent */
isolax( 1 , arccot(U)=V , U=cot(V) , true ) .

/* sinh  */
isolax( 1, sinh(U)=V,  U=log(e,X),true) :- tidy(V+(V^2+1)^(1/2),X),!.

/* cosh */
isolax( 1, cosh(U)=V,false,true) :- eval(V<1),
        tidy(V,V1),
trace('\n%t=%t has no real roots, cosh must be >= 1\n',[cosh(U),V1],1),
        !.


isolax( 1, cosh(U)=V,U=log(e,X) # U=log(e,Y),true) :-
        tidy(V+(V^2-1)^(1/2),X),
        tidy(V-(V^2-1)^(1/2),Y),
        !.


/* tanh */
isolax( 1, tanh(U)=V,false,true) :- (eval(V=< -1);eval(V>=1)),
        tidy(V,V1),
trace('\n%t=%t has no real roots, tanh must lie in (-1,1)\n',[tanh(U),V1],1),
        !.

isolax( 1, tanh(U)=V,U=log(e,X)*(1/2),true) :- tidy((1+V)*(1-V)^ -1,X),!.

/* cosech */
isolax( 1, cosech(U)=V,U=log(e,X),non_zero(V)) :-
        tidy(1/V,V1),
        tidy(V1+(V1^2+1)^(1/2),X),
        !.

/* sech */
isolax( 1, sech(U)=V,false,true) :- (eval(V=<0);eval(V>1)),
        tidy(V,V1),
trace('\n%t=%t has no real roots, sech must lie in (0,1]\n',[sech(U),V1],1),
        !.


isolax( 1, sech(U)=V1,U=log(e,X) # U=log(e,Y),non_zero(V1)) :-
        tidy(1/V1,V),
        tidy(V-(V^2-1)^(1/2),Y),
        tidy(V+(V^2-1)^(1/2),X),
        !.

/* coth */
isolax( 1, coth(U)=V, false, true) :-
        ((eval(V>0),eval(V<1));(eval(V<0),eval(V> -1))),
        tidy(V,V1),
trace('\n%t=%t has no real roots, coth can not lie in (-1,1)\n',[coth(U),V1],1),
        !.

isolax( 1,coth(U)=V1,U=log(e,X)*(1/2),non_zero(V1)) :-
        tidy(1/V,V1),
        tidy((1+V)*(1-V)^ -1,X),
        !.

/* inverse sinh */
isolax( 1,arcsinh(U)=V,U=sinh(V),true).

/* inverse cosh  */
isolax( 1, arccosh(U)=V,false,true) :- eval(U<1),
        tidy(U,U1),
trace('\n%t=%t has no real roots, cosh must be >= 1\n',[arccosh(U1),V],1),
        !.


isolax( 1,arccosh(U)=V,U=cosh(V),true).

/* inverse tanh */
isolax( 1,arctanh(U)=V,false,true) :- (eval(U>=1);eval(U=< -1)),
        tidy(U,U1),
trace('\n%t=%t has no real roots, tanh must lie in (-1,1)\n',[arctanh(U1),V],1),
        !.

 solax( 1,arctanh(U)=V,U=tanh(V),true).

/* inverse sech */
isolax( 1, arcsech(U)=V,false,true) :- eval(U>0),eval(U=<1),
        tidy(U,U1),
trace('\n%t=%t has no real roots, sech must lie in [0,1]\n',[arcsech(U1),V],1),
        !.

isolax( 1, arcsech(U)=V,U=sech(V), true).

/* inverse cosech */
isolax( 1, arccosech(U)=V,U=cosech(V), true).

/* inverse  coth */
isolax( 1, arccoth(U)=V, false, true) :-
        ((eval(U>0),eval(U<1));(eval(U<0),eval(U> -1))),
        tidy(U,U1),
trace('\n%t=%t has no real roots, coth can not lie in (-1,1)\n',[arccoth(U1),V],1)
        !.

isolax( 1, arccoth(U)=V,U=coth(V), true).
```

```
/*                              INEQIS.AX       19.2.81 */
% modified 2.3.81
/*ISOLATION AXIOMS FOR >= */
/* multiplication */
isolax( 1 ,U*V>=W, U>=W*V1,positive(V)) :- tidy(1/V,V1).

isolax( 1 ,U*V>=W, U =< W*V1, negative(V)) :- tidy(1/V,V1).

isolax( 2 , V*U>=W, U>=W*V1, positive(V)) :- tidy(1/V,V1).

isolax( 2 , V*U>=W, U =< W*V1, negative(V)) :- tidy(1/V,V).

/* addition */
isolax( 1 ,U+V>=W,U>=W+(-1)*V,true).
isolax( 2 , V+U>=W, U>=W+(-1)*V, true ) .


/* sine */
isolax( 1 ,sin(U)>=V,U>=arcsin(V),acute(U)).

/* tangent */
isolax( 1 , tan(U) >= V, U >= arctan(V), acute(U)).

/*ISOLATION AXIOMS FOR > */
/* multiplication */
isolax( 1 ,U*V>W, U>W*V1,positive(V)) :- tidy(1/V,V1).

isolax( 1 ,U*V>W, U < W*V1, negative(V)) :- tidy(1/V,V1).

isolax( 2 , V*U>W, U>W*V1, positive(V)) :- tidy(1/V,V1).

isolax( 2 , V*U>W, U < W*V1, negative(V)) :- tidy(1/V,V1).

/* addition */
isolax( 1 ,U+V>W,U>W+(-1)*V,true).
isolax( 2 , V+U>W, U>W+(-1)*V, true).

/* sine */
isolax( 1 ,sin(U)>V,U>arcsin(V),acute(U)).

/* tangent */
isolax( 1 , tan(U) > V, U > arctan(V), acute(U)).

/* square root */
isolax( 1 ,U^K>V,U>V^2,true) :- eval(K=:=1/2),!.


/* Isolation Axioms for < */

/* multiplication */

isolax( 1 ,U*V < W, U < W*V1, positive(V)) :- tidy(1/V,V1).

isolax( 1 ,U*V < W, U>W*V1,negative(V)) :- tidy(1/V,V1).

isolax( 2 , V*U < W, U < W*V1, positive(V)) :- tidy(1/V,V1).

isolax( 2 , V*U < W, U>W*V1, negative(V)) :- tidy(1/V,V1).
```

```
/*                  COLLEC.AX          19.2.81   */
/* AXIOMS FOR COLLECTION*/
/* FIRST ARGUMENT IS THE VARIABLES COLLECTED*/
/* ALL COLLECTION AXIOMS APPLY TO TERMS DOMINATED BY + OR */

:- public                   collax/3.

collax( W , U*W+V*W , (U+V)*W ) .

collax( W , W+V*W , (V+1)*W ) .

collax( W , W+W , 2*W ) .

collax( U&V , (U+V)*(U+ (-1*V)) , U^2+ -1*(V^2) ) .

collax( W , W^U*W^V , W^(U+V) ) .

collax( W , W*W^V , W^(V+1) ) .

collax( W , W*W , W^2 ) .

collax( U , sin(U)*cos(U) , sin(2*U)* (1/2) ) .

collax( U , cos(U)^2+ -1*(sin(U)^2) , cos(2*U) ) .

collax( U , sin(U)*cos(V)+cos(U)*sin(V) , sin(U+V) ) .

collax( U&V , sin(U)*cos(V)+ -1*(cos(U)*sin(V)) , sin(U+ (-1*V)) ) .

collax( U , cos(U)*cos(V)+ -1*(sin(U)*sin(V)) , cos(U+V) ) .

collax( U , cos(U)*cos(V)+sin(U)*sin(V) , cos(U+ (-1*V)) ) .

collax( U , cos(U)^2 + sin(U)^2 , 1 ).

collax( U , log(U,X) + log(U,Y) , log(U,X*Y) ) .

collax( U , A*log(U,X) + B*log(U,Y),log(U,X^A*Y^B) ).

collax( U, A*log(U,X) + log(U,Y),log(U,X^A*Y) ).
```

```
/*                  ATTRAC.AX          19.2.81    */
/* New axioms added 17.9.81 */
/* AXIOMS FOR ATTRACTION*/
/* FIRST ARGUMENT IS THE SET OF THE VARIABLES ATTRACTED*/

:- public                    attrax/3.

attrax( U & V , U*W+V*W , (U+V)*W ) .

attrax( U & V , W^U*W^V , W^(U+V) ) .

attrax( U & V , log(W,U)+log(W,V) , log(W,U*V) ) .

attrax( U & V , A*log(W,U)+B*log(W,V),log(W,U^A*V^B) ) .

attrax( U & V , A*log(W,U)+log(W,V),log(W,U^A*V) ) .

attrax( U & V , U*log(W,V) , log(W,V^U) ) .

attrax( U & V , log(W,V)*log(U,W) , log(U,V) ) .

attrax( U & V , U=V , U+(-1*V)=0 ) .

attrax( U & V , U>V , U+(-1*V)>0 ) .

attrax( U & V , U>=V , U+(-1*V)>=0 ) .

attrax( V & W , (U^V)^W , U^(V*W) ) .

attrax( U & V , U^(V*W) , (U^V)^W ) .
```

```
/*      HOMOG.REW       */
/* Written by Bernard Silver  Jan 1981  */
% Updated: 30 June 82

:- public
                rew/5,
                rew1/5.

/* Try to rewrite each of the terms in the offending set as a
    function of the reduced term */
rew(X,L,Subs,Unk,Type) :-  newtype(Type,New),
        maplist(rew1(New,X,Unk),L,L1),
        make_subl(L,L1,Subs),
        !.

                            %  Kludge for stopping recursive calls of rew-rule
                            %  in mixed case, and for getting the log case right
newtype(mixed,_) :- !.
newtype(log(X),log) :- X \== 10.
newtype(C,C) :- !.

rew1(_,X,_,X,X) :- !.
rew1(Type,A^B,Unk,Old,New) :- !,rew_rule(Type,A^B,Old,New,Unk).
rew1(Type,X,Unk,A^B,C^D) :- rew1(Type,X,Unk,A,C),rew1(Type,X,Unk,B,D),!.
rew1(Type,X,Unk,Old,New) :- rew_rule(Type,X,Old,New,Unk),!.

/* rew_rule(Type,Term1,Term2,Exp,Unk) gives Exp as a rewrite of Term2 in terms */
/* of Term1,where Unk is the  unknown, and the rule is for type Type */

/* Special cases   */
rew_rule(_,X,Y,X,_) :- match(X,Y),!.

rew_rule(_,_,Y,Y,Unk) :- freeof(Unk,Y),!.

/* Generalized Polynomial Rewrite rules  */
rew_rule(genpol,X^M,X,(X^M)^K,X) :- number(M),!,eval(1/M,K).

rew_rule(genpol,X^N,X^M,(X^N)^K,X) :- number(N),number(M),!,eval(M/N,K).

/* Hyperbolic Rewrite rules  */
rew_rule(T,e^X,sinh(Z),((e^X)^K-(e^X)^(-K))/2,_) :- (T = hyper;T = hyper_exp),
        break(X,Z,P,Q),
        eval(Q/P,K),
        !.

rew_rule(T,e^X,cosh(Z),((e^X)^K+(e^X)^(-K))/2,_) :- (T = hyper;T = hyper_exp),
        break(X,Z,P,Q),
        eval(Q/P,K),
        !.

rew_rule(T,e^X,tanh(Z),((e^X)^K-(e^X)^(-K))*((e^X)^K+(e^X)^(-K))^ -1,_) :-
        (T = hyper;T = hyper_exp),
        break(X,Z,P,Q),
        eval(Q/P,K),
        !.

rew_rule(T,e^X,sech(Z),((e^X)^K+(e^X)^(-K))^ -1*2,_) :-
        (T = hyper;T = hyper_exp),
        break(X,Z,P,Q),
        eval(Q/P,K),!.
```

```
rew_rule(T,e^X,cosech(Z),((e^X)^K-(e^X)^(-K))^ -1*2,_) :-
        (T = hyper;T = hyper_exp),
        break(X,Z,P,Q),
        eval(Q/P,K),!.

rew_rule(T,e^X,coth(Z),((e^X)^K+(e^X)^(-K))*((e^X)^K-(e^X)^(-K))^ -1,_) :-
        (T = hyper;T = hyper_exp),
        break(X,Z,P,Q),
        eval(Q/P,K),
        !.

rew_rule(T,sinh(X),cosh(X),(1 + sinh(X)^2)^(1/2),_) :-
        (T = hyper;T = hyper_exp),!.

rew_rule(T,cosh(X),sinh(X),(cosh(X)^2 - 1)^(1/2),_) :-
        (T = hyper;T = hyper_exp),!.

rew_rule(T,sech(X),tanh(X),(1 - sech(X)^2)^(1/2),_) :-
        (T = hyper;T = hyper_exp),!.

rew_rule(T,tanh(X),sech(X),(1 - tanh(X)^2)^(1/2),_) :-
        (T = hyper;T = hyper_exp),!.

rew_rule(T,coth(X),cosech(X),(coth(X)^2 - 1)^(1/2),_) :-
        (T = hyper;T = hyper_exp),!.

rew_rule(T,cosech(X),coth(X),(1 + cosech(X)^2)^(1/2),_) :-
        (T = hyper;T = hyper_exp),!.


/* Exponential Rewrite rules  */
rew_rule(T,A^B,A^C,A^B,_) :- (T = exp;T = hyper_exp),match(B,C),!.

rew_rule(T,A^B,V^Z,X*Y,Unk) :-  (T == exp;T == hyper_exp),
        match(Z,C*D+E),
        !,
        rew_rule(T,A^B,V^(C*D),X,Unk),
        rew_rule(T,A^B,V^E,Y,Unk).

rew_rule(T,A^B,A^Z,A^C*A^B,X) :- (T = exp;T = hyper_exp),
        match(Z,B+C),
        freeof(X,C),!.

rew_rule(T,A^B,A^Z,(A^B)^C,X) :- (T = exp;T = hyper_exp),
        match(Z,B*C),
        freeof(X,C),!.

rew_rule(Type,A^B,C^D,C^E*Z,X) :- Type == exp,
        number(A),
        number(C),
        match(D,B+E),
        freeof(X,E),
        rew_rule(exp,A^B,C^B,Z,X),
        !.

rew_rule(exp,A^B,C^B,(A^B)^N,_) :- powered(A,N,C),!.

rew_rule(T,A^B,A^C,(A^B)^D,_) :- (T = exp;T = hyper_exp),
        match(B,E*F),
        number(E),
        match(C,G*F),
        number(G),
        eval(G/E,D),
        !.

/* Trignometric Rewrite rules */
rew_rule(T,sin(X),sin(Z),V*cos(C) + V1*sin(C),U) :- T == trig,
        match(Z,B + C),
        contains(U,B),
        freeof(U,C),
        rew_rule(trig,sin(X),sin(B),V,U),
        rew_rule(trig,sin(X),cos(B),V1,U),
        !.

rew_rule(T,sin(X),cos(Z),V*cos(C) - V1*sin(C),U) :- T == trig,
        match(Z,B + C),
        contains(U,B),
        freeof(U,C),
        rew_rule(trig,sin(X),sin(B),V1,U),
        rew_rule(trig,sin(X),cos(B),V,U),
        !.

rew_rule(T,cos(X),sin(Z),V*cos(C) + V1*sin(C),U) :- T == trig,
        match(Z,B + C),
        contains(U,B),
        freeof(U,C),
        rew_rule(trig,cos(X),sin(B),V,U),
        rew_rule(trig,cos(X),cos(B),V1,U),
        !.

rew_rule(T,cos(X),cos(Z),V*cos(C) - V1*sin(C),U) :- T == trig,
        match(Z,B + C),
        contains(U,B),
        freeof(U,C),
        rew_rule(trig,cos(X),cos(B),V,U),
        rew_rule(trig,cos(X),sin(B),V1,U),
        !.


rew_rule(trig,sin(X),cos(Z),V,_) :- break(X,Z,P,Q),
        absol(Q,Q1),
        expcs(P,Q1,X,V),
        !.

rew_rule(trig,sin(X),sin(Z),I*(V),_) :- break(X,Z,P,Q),
        absol(Q,Q1),
        eval(sign(Q),I),
        expss(P,Q1,X,V),
        !.

rew_rule(trig,cos(X),sin(Z),I*(V),_) :- break(X,Z,P,Q),
        absol(Q,Q1),
        eval(sign(Q),I),
        expsc(P,Q1,X,V),
        !.

rew_rule(trig,cos(X),cos(Z),V,_) :- break(X,Z,P,Q),
        absol(Q,Q1),
        expcc(P,Q1,X,V),
        !.

rew_rule(trig,tan(X),sec(X),(1+tan(X)^2)^(1/2),_) :- !.

rew_rule(trig,sec(X),tan(X),(sec(X)^2-1)^(1/2),_) :- !.

rew_rule(trig,cot(X),cosec(X),(1+cot(X)^2)^(1/2),_) :- !.

rew_rule(trig,cosec(X),cot(X),(cosec(X)^2-1)^(1/2),_) :- !.

rew_rule(T,tan(X),tan(Z),(V + tan(C))/(1 - tan(C)*V),U) :- T == trig,
        match(Z,B + C),
        contains(U,B),
        freeof(U,C),
        rew_rule(trig,tan(X),tan(B),V,U),
        !.

rew_rule(trig,tan(X),tan(Z),I*(V),_) :- break(X,Z,P,Q),
        absol(Q,Q1),
        eval(sign(Q),I),
        exptt(P,Q1,X,V),
        !.

rew_rule(trig,tan(X),cosec(X),(1+tan(X)^2)^(1/2)/tan(X),_) :- !.

rew_rule(trig,tan(X),sin(X),tan(X)/(1+tan(X)^2)^(1/2),_) :- !.

rew_rule(trig,tan(X),cos(X),1/(1+tan(X)^2)^(1/2),_) :- !.

/* Tan half-angle Rewrite rules */
rew_rule(trig,tan(X),sin(Z),2*tan(X)*(1+tan(X)^2)^(-1),_) :-break(X,Z,P,Q),
        eval(Q/P=:=2),!.

rew_rule(trig,tan(X),cos(Z),(1-tan(X)^2)*(1+tan(X)^2)^(-1),_) :-break(X,Z,P,Q),
        eval(Q/P=:=2),!.

/* Reciprocal function Rewrite rules  */
rew_rule(T,X,tan(Z),A*B^ -1,Unk) :- T == trig,
        rew_rule(trig,X,sin(Z),A,Unk),
        rew_rule(trig,X,cos(Z),B,Unk),
        !.

rew_rule(T,A,sec(Z),(B)^ -1,Unk) :- T == trig,
        rew_rule(trig,A,cos(Z),B,Unk),
        !.

rew_rule(T,A,cosec(Z),(B)^ -1,Unk) :-  T == trig,
        rew_rule(trig,A,sin(Z),B,Unk),
        !.

rew_rule(T,A,cot(Z),(B)^ -1,Unk) :- T == trig,
        rew_rule(trig,A,tan(Z),B,Unk),
        !.


/* Logarithmic Rewrite rules  */
rew_rule(log,log(X,Y),log(Y,X),log(X,Y)^ -1,_) :- !.

rew_rule(log,log(X,Y),log(Z,Y),N*log(X,Y),_) :- powered(X,N,Z),!.
rew_rule(log,log(X,Y),log(Y,Z),N*log(X,Y)^ -1,_) :- powered(X,N,Z),!.

rew_rule(log,log(X,Y),log(X,Z),N*log(X,Y),_) :- powered(Y,N,Z),!.

rew_rule(log,log(X,Y),log(Z,X),N*log(X,Y)^ -1,_) :- powered(Y,N,Z),!.

                                % Reduced term is log base 10
rew_rule(log(10),log(10,X),log(X,10),log(10,X)^ -1,_) :- !.

rew_rule(log(10),log(10,X),log(A,X),Term,Unk) :-
        number(A),
        tidy(log(10,X)/log(10,A),Term),
        !.

rew_rule(log(10),log(10,X),log(X,A),Term,Unk) :-
        number(A),
        tidy(log(10,A)*(log(10,X)^ -1),Term),
        !.

/* Failure  */
rew_rule(_,X,Y,_,_) :- !,
        trace('\nFailed to find a rewrite for %t\n in terms of %t\n',[Y,X],2),
        fail.
```

```
/* FACTS. :

        Miscellaneous facts for PRESS

                                                Bernard Silver
                                                Updated: 30 May 82

*/

/* EXPORTS  */

:-      public special_atom/1,
        commutative/1,
        associative/1.

/* MODES   (Defined as used now, may need changing later) */

:-      mode special_atom(+),
        commutative(+),
        associative(+).

 % Special atoms are positive, and therefore non_neg and non_zero
 pecial_atom(e) :- !.

special_atom(pi) :- !.

 % Properties of functions
commutative(+) :- !.
commutative(*) :- !.

associative(+) :- !.
associative(*) :- !.
```

```
/*                  INIT.MEC                 1.4.81  */

/*   Various initialisations for MECHO.   */

  const(g).
  const(zero).



/*SEMANTIC INFORMATION*/

quantity(reactionq1).   measure(reactionq1,reaction1).
quantity(muq).          measure(muq,mu).
quantity(vcq).          measure(vcq,vc).
quantity(velq1).        measure(velq1,vel1).
quantity(tq0).          measure(tq0,t0).
quantity(tq1).          measure(tq1,t1).
quantity(tq2).          measure(tq2,t2).
quantity(tq3).          measure(tq3,t3).
quantity(dq0).          measure(dq0,d0).
quantity(dq1).          measure(dq1,d1).
quantity(dq2).          measure(dq2,d2).
quantity(dq3).          measure(dq3,d3).
quantity(aq0).          measure(aq0,a0).
quantity(aq1).          measure(aq1,a1).
quantity(aq3).          measure(aq3,a3).
quantity(vq).           measure(vq,v).
quantity(hq1).          measure(hq1,h1).
quantity(hq2).          measure(hq2,h2).
quantity(hq).           measure(hq,h).
quantity(rq).           measure(rq,r).
quantity(tq).           measure(tq,t).
quantity(aq).           measure(aq,a).
quantity(angq).         measure(angq,ang).
quantity(dq).           measure(dq,d).
quantity(mq).           measure(mq,m).
quantity(mq1).          measure(mq1,m1).
quantity(mq2).          measure(mq2,m2).
quantity(tsnq).         measure(tsnq,tsn).
quantity(l1q).          measure(l1q,l1).
quantity(yq).           measure(yq,y).

incline(s3,tq,cc).      slope(s3,right).                concavity(s3,stline).
incline(s4,yq,bot).     slope(s4,right).                concavity(s4,stline).
angle(tp1,angq,semi).   partition(semi,[s1,s2]).
                        slope(s1,left).                 concavity(s1,right).
                        slope(s2,right).                concavity(s2,right).
angle(tp2,dq,dome).     slope(dome,left).               concavity(dome,right).


/*unknowns*/
sought(h1).             sought(h2).             sought(t).
sought(v).              sought(h).              sought(x).
sought(a).              sought(d).              sought(l).
intermediate(y).        intermediate(ang).      intermediate(m).
given(r).               given(g).               given(m1).
given(m2).
```

```
/*                          POLPAK          */

/*      Polynomial arithmetic package
                Gathered together by Leon 23.2.81
                Extra methods added 3.4.81
                Guessing roots by remainder theorem by Bernard
                        Last Updated: 30 March 82
*/

%declarations%

:- public
                even_anti_symmetric/1,
                factor_out/3,
                guess/2,
                make_poly/3,
                odd_anti_symmetric/1,
                odd_symmetric/1,
                poly/4,
                poly_form_coeff/2,
                poly_norm/3,
                reify/3,
                sym_method/3,
                z_norm/2.

:- mode
        addnorm(+,+,?),
        add_poly(+,+,?),
        allowed_guess(+,?),
        anti_symmetric(+,+),
        binomial(+,+,?),
        build_red(+,+,+,?),
        denorm(+,?),
        denorm1(+,+,?),
        div_lin(+,+,+,+,?),
        even_anti_symmetric(+),
        factors_of(+,?,?),
        factor_out(+,+,?),
        get_coeff_factor(+,?),
        get_cs(+,?),
        guess(+,?),
        guess1(+,+,+),
        make_poly(+,+,?),
        map_reify(+,+,?),
        mappoly_form_coeff(+,?),
        mappoly_form1(+,?),
        odd_anti_symmetric(+),
        odd_symmetric(+),
        pbag_add(+,+,?,?,?,?,?),  % can probably better than this
        pbag_norm(+,?),
        poleval(+,+,?),
        poleval1(+,+,+,?),
        poly(+,+,?,?),
        poly_form_coeff(+,?),
        poly_norm(+,+,?),
        reify(+,+,?),
        sym_method(+,?,?),
        sym_reduce(+,?,?),
        symmetric(+,+),
        times(+,+,?),
          times_norm(+,+,?),
          timesingl(+,+,+,?),
          trans(+,?),
          z_norm(+,?).

/* Put polynomials in normal form (succeeds only for polynomials) */

poly_norm(X,Poly,Pbag2) :- !,
          poly(X,Poly,Pbag,poly),
          mappoly_form_coeff(Pbag,Pbag1),
          z_norm(Pbag1,Pbag2).

/* Tidy coefficients */

poly_form_coeff(polyand(N,E),polyand(N,E1)) :- poly_form(E,E1).



/* Forms bag of coefficients */

poly(X,X,[polyand(1,1)],Flag) :- !.

poly(X,X^N,[polyand(N,1)],poly) :-
          integer(N), !.

poly(X,X^N,[polyand(N,1)],simp) :-
          integer(N), !.

poly(X,(X^N)^(-1),[polyand(N1,1)],Flag) :-
          integer(N), !, eval(-N,N1).


poly(X,E,[polyand(0,E)],Flag) :-
          freeof(X,E), !.

poly(X,S+T,Ebag,Flag) :-
          !,
          poly(X,S,Sbag,Flag), poly(X,T,Tbag,Flag),
          add_poly(Sbag,Tbag,Ebag).

poly(X,S*T,Ebag,Flag) :- !,
          poly(X,S,Sbag,Flag), poly(X,T,Tbag,Flag),
          times(Sbag,Tbag,Ebag).

poly(X,S^N,Ebag,Flag) :-
          integer(N),
          eval(N > 0),
          !,
          poly(X,S,Sbag,Flag),
          binomial(Sbag,N,Ebag).

poly(X,E,[polyand(0,E1)],simp) :- !,
          E=..[Sym|Args],
          mappoly_form1(Args,Args1),
          E1=..[Sym|Args1].

/* Add two coefficients bags  */

add_poly([],Bag,Bag) :- !.
add_poly(Bas,[],Bas) :- !.

add_poly(S,T,Sum) :-
        pbas_norm(S,Snorm),
        pbas_norm(T,Tnorm),
        addnorm(Snorm,Tnorm,Sum),
        !.


addnorm([],T,T) :- !.

addnorm(S,[],S) :- !.

addnorm([polyand(N,E)|P],[polyand(M,F)|Q],[polyand(N,E)|Y]) :-
                eval(N > M),
                addnorm(P,[polyand(M,F)|Q],Y),
                !.

addnorm([polyand(N,E)|P],[polyand(M,F)|Q],[polyand(N,Y)|Z]) :-
                eval(N = M),
                addnorm(P,Q,Z),
                tidy(E+F,Y),
                !.

addnorm([polyand(N,E)|P],[polyand(M,F)|Q],[polyand(M,F)|Y]) :-    %N < M
                addnorm(Q,[polyand(N,E)|P],Y), !.



/* Multiply two coefficient bass  */

times([],Bas,[]) :- !.

times(Bas,[],[]) :- !.

times(S,T,Prod) :-
        pbas_norm(S,Snorm),
        timesnorm(Snorm,T,Prod),
        !.

timesnorm(S,[polyand(N,E)],X) :- timesingl(S,N,E,X), !.


timesnorm(S,[polyand(N,E)|R],X) :-
        timesingl(S,N,E,Y),
        timesnorm(S,R,Z),
        addnorm(Y,Z,X),
        !.

timesingl([],N,E,[]) :- !.

timesingl([polyand(M,F)],N,E,[polyand(X,Y)]) :-
                eval(M+N,X),
                tidy(F*E,Y),
                !.

timesingl([polyand(M,F)|R],N,E,[polyand(X,Y)|Z]) :-
                timesingl(R,N,E,Z),
                eval(M+N,X),
                tidy(F*E,Y).
```

```
/* Binomial expansion of coefficient bas */


binomial(Bas, 0, [polyand(0,1)]) :- !.

binomial(Bas, 1, Bas) :- !.

binomial(Sbas, N, Ebas) :-
                !,
                eval(N-1,N1),
                binomial(Sbas,N1,Ebas1),
                times(Sbas,Ebas1,Ebas).



/*      Put polynomial bass into a normal form   */

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

pbas_norm([polyand(X,Y)],[polyand(X,Y)]) :- !.

pbas_norm([polyand(N,E)|R],Pnorm) :-
                integer(N),
                pbas_norm(R,[polyand(M,F)|S]),
                integer(M),
                pbas_add(N,M,E,F,S,Pnorm),
                !.

pbas_add(N,M,E,F,S,[polyand(N,E),polyand(M,F)|S]) :- eval(N > M), !.

pbas_add(N,M,E,F,S,[polyand(N,Y)|S]) :- N = M, tidy(E+F,Y), !.

pbas_add(N,M,E,F,S,[polyand(M,F),polyand(N,E)|S]).


% Remove any  terms with zero coefficient

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

z_norm([polyand(N,0)|R],Pnorm) :- z_norm(R,Pnorm), !.

z_norm([polyand(N,A)|R],[polyand(N,A)|Pnorm]) :- z_norm(R,Pnorm).

/* Put in normal form,undo the effect of z_norm   */

denorm([polyand(0,A)],[polyand(0,A)]) :- !.
denorm([polyand(N,A)|R],[polyand(N,A)|R1]) :- denorm1(N,R,R1),!.

denorm1(0,_,[]) :- !.
denorm1(N,[polyand(L,B)|R],[polyand(L,B)|R1]) :- eval(N-1 =:= L),denorm1(L,R,R1),
denorm1(N,R,[polyand(M,0)|R1]) :- eval(N-1,M),denorm1(M,R,R1),!.

/* Code to factor out the linear factor x+B   */

factor_out([polyand(N,A)|Plist],B,Qlist) :-
                !,
                eval(N-1,M),
                    div_lin(Plist,M,A,B,Qlist).

div_lin([],-1,0,_,[]) :- !.

div_lin([],-1,_,_,_) :- !, trace('Division error \n',1).

div_lin([polyand(N,C)|Plist],M,A,B,[polyand(M,A)|Qlist]) :-
                    eval(N < M),                        % The sparse case
                    !,
                    eval(M-1,M1),
                    tidy(A*B* -1,A1),
                    div_lin([polyand(N,C)|Plist],M1,A1,B,Qlist).

div_lin([polyand(N,C)|Plist],M,A,B,[polyand(M,A)|Qlist]) :-
                    eval(N = M),              % N should never be greater than M
                    !,
                    eval(M-1,M1),
                    tidy(C - A*B,A1),
                    div_lin(Plist,M1,A1,B,Qlist).

/* Evaluate the polynomial represented by Plist,at Val to give Ans   */
poleval(Plist,Val,Ans) :- poleval1(Plist,Val,0,Ans),!.

poleval1([polyand(_,A)|R],V,Res,Ans) :- eval(Res*V+A,X),
          poleval1(R,V,X,Ans),
          !.
poleval1([],_,Ans,Ans) :- !.

/*Try to guess roots by applying remainder theorem  */

guess(Plist,Root) :- denorm(Plist,Plist1),
          get_coeff_factor(Plist1,M),
          last(polyand(0,K),Plist1),
          gcd2([M,K],K1),
          eval(K/K1,K2),
          allowed_guess(K2,List),
          guess1(List,Plist1,Root),!.

% Find the gcd of all the coefficients, check that its not 0
get_coeff_factor(List,Gcd) :- get_cs(List,Coefflist),
          gcd2(Coefflist,Gcd),
          eval(Gcd\= 0),
          !.

get_cs([],[]) :- !.
get_cs([polyand(_,L)|T],[L|T1]) :- number(L),get_cs(T,T1),!.

% If Plist has an integer root then root is a factor of constant term
% divided by the gcd of all the coefficients.

allowed_guess(K,[1,-1|T]) :- factors_of(K,T,2),!.

factors_of(K,[],10) :- !.
factors_of(K,[M,H|T],M) :- eval(K/M,N),
          integer(N),
          eval(-M,H),
          eval(M+1,M1),
          factors_of(K,T,M1),
          !.
factors_of(K,T,M) :- eval(M+1,M1),factors_of(K,T,M1),!.
```

```
/* Using list of possible roots see if any are roots */

guess1([],_,_) :- !,fail.
guess1([N|_],Plist,Root) :- poleval(Plist,N,Ans),
      eval(Ans = 0),
      !,
      Root = N.
guess1([_|T],Plist,Root) :- guess1(T,Plist,Root),!.


/* Reconstitute bag of coefficients into polynomial */

make_poly(X,Bag1,Poly) :- !,
             mapreify(X,Bag1,Bag2),
             recomp(Poly,[+|Bag2]).


/* reify coefficient and power into product */

reify(X,polyand(0,E),E) :- !.

reify(X,polyand(1,E),E*X) :- !.

reify(X,polyand(N,E),E*X^N) :- !.


/* Method for standard reciprocal equations  */

sym_method(X,[polyand(N,A)|Plist],Poly) :-
      symmetric(N,[polyand(N,A)|Plist]),
      !,
      sym_reduce(X,[polyand(N,A)|Plist],Poly).


/* Reduce symmetric polynomial to one with half the degree */

sym_reduce(X,[polyand(N,A)|Plist],NewPoly) :-
      eval(N/2,M),
      build_red(M,0,Plist,Qlist),
      trans([polyand(M,A)|Qlist],Rlist),
      make_poly(X+1/X,Rlist,NewPoly),
      !.

build_red(M,M,_,[]) :- !.

build_red(M,K,[polyand(N,A)|Plist],[polyand(M1,A)|Qlist]) :-
      eval(M-K-1,M1),
      eval((2*M-K)-1,N),
      !,
      eval(K+1,K1),
      build_red(M,K1,Plist,Qlist).

build_red(M,K,Plist,[polyand(M1,0)|Qlist]) :-
      eval(M-K-1,M1),
      eval(K+1,K1),
      build_red(M,K1,Plist,Qlist).

% Special code which holds for the quartic case
```

```
trans([polyand(2,A),polyand(1,B),polyand(0,C)],
       [polyand(2,A),polyand(1,B),polyand(0,D)]) :-  tidy(C-2*A,D),!.

trans(Plist,Plist) :- writef('Relevant reduction code not written'),fail.


/* Test if polynomial is symmetric or anti-symmetric */

odd_symmetric([polyand(N,A)|Plist]) :-
              odd(N),symmetric(N,[polyand(N,A)|Plist]).

odd_anti_symmetric([polyand(N,A)|Plist]) :-
              odd(N),anti_symmetric(N,[polyand(N,A)|Plist]).

even_anti_symmetric([polyand(N,A)|Plist]) :-
              even(N),anti_symmetric(N,[polyand(N,A)|Plist]).



symmetric(_,[]) :- !.

symmetric(N,[polyand(M,_)]) :-  eval(N/2,M), !.

symmetric(N,[polyand(L,A)|Plist]) :-
             append(Qlist,[polyand(M,A)],Plist),
             eval(M+L,N),
             !,
             symmetric(N,Qlist).

anti_symmetric(N,[]) :- !.
anti_symmetric(N,[polyand(L,A)|Plist]) :-
             append(Qlist,[polyand(M,B)],Plist),
             eval(M+L,N),
             eval(-A,B),
             !,
             anti_symmetric(N,Qlist).

 % Converted maplists etc
mappoly_form1([],[]) :- !.
mappoly_form1([H|T],[H1|T1]) :- poly_form1(H,H1),mappoly_form1(T,T1),!.


mapreify(_,[],[]) :- !.
mapreify(X,[H|T],[H1|T1]) :- reify(X,H,H1),mapreify(X,T,T1),!.

mappoly_form_coeff([],[]) :- !.
mappoly_form_coeff([H|T],[H1|T1]) :- poly_form_coeff(H,H1),
       mappoly_form_coeff(T,T1),
       !.
```

```
/*      POLYIS           31.3.81  */

%declarations%

:- public
                half_poly/3,
                poly_form/2,
                poly_form1/2.

/*****************************
        POLYNOMIAL NORMAL FORM
*****************************/


/* Use polynomial form for simplification (always succeeds) */

poly_form(true,true).
poly_form(false,false).

poly_form(Exp,Poly) :- !,
        poly_form1(Exp,New),
        tidy(New,Poly).

/* Look for terms to simplify */

poly_form1(Exp,Poly) :-
        Exp=..[Sym|Args], ispred(Sym), !,
        maplist(poly_form1,Args,PArgs),
        Poly=..[Sym|PArgs].

/* Apply to term */

poly_form1(Exp,Poly) :- !,
        wordsin(Exp,Vars),
        sublist(mult_occ(Exp),Vars,Vars1),
        poly_form(Vars1,Exp,Poly).



/* Test for predicate or logical connective */

ispred(&).       ispred(#).       ispred(=).
ispred(>).       ispred(>=).      ispred(<).       ispred(=<).



/* Put term in polynomial normal form with respect to list of variables*/

poly_form([],Exp,Exp) :- !.

poly_form([Var|Vars],Exp,Poly) :- !,
        poly(Var,Exp,Ebag1,simp),
        maplist(half_poly(Vars),Ebag1,Ebag2),
        make_poly(Var,Ebag2,Poly).


/* Apply poly_form to coeffs */

half_poly(Vars,polyand(N,E1), polyand(N,E2)) :- !,
        poly_form(Vars,E1,E2).
```

```
%   Press:Weaknf.                       Updated: 13 Sept 81
%                                       Author:  Bernard Silver 28.4.81
%   Put expression into weak normal form for collection, attraction, &c.

:- public weaknf/3,
        zero_rhs/2,
        filter/4.
:- mode
    weaknf(+, +, -),
        zero_rhs(+, -),
        filter(+, +, -, -).

weaknf(Eqn, Var, New) :-
        zero_rhs(Eqn, Mid),
        decomp(Mid, [+|Bag]),
        filter(Bag, Var, Lhs, Rhs),
        tidy(Lhs=Rhs, New), !.

weaknf(Eqn, Var, New=0)  :- zero_rhs(Eqn,New),!.
%   put an equation Lhs=Rhs into the form New=0.

        zero_rhs(Lhs=0, Lhs) :- !.
        zero_rhs(Lhs=Rhs, New) :- tidy(Lhs-Rhs, New).

%   split a sum bag into Lhs, holding all elements containing Var,
%   and Rhs, holding all the elements not containing Var.  We are
%   free to use '-' in Rhs, as it will be tidied before use.

        filter([Head|Tail], Var, Head+More, Rest) :-
                contains(Var, Head), !,
                filter(Tail, Var, More, Rest).
        filter([Head|Tail], Var, More, Rest-Head) :- !,
                filter(Tail, Var, More, Rest).
        filter([],        Var, 0,    0).
```

```
%   Press:Match.                                  Updated: 14 March 82

%=========================================================================%
%                        Pattern Matcher                   24.2.81        %
%=========================================================================%

%   Exports...

:- public
    corresponding_arguments/4,  %   (replaces any1)
    decomp/2,
    match/2,
    recomp/2,
*   ac-op/5.

:- mode
    corresponding_arguments(+,-,-,-),
    decomp(+,?),
        ac_decomp(+,+,?,?),
        ac_op(+,?,?,?,-),
    recomp(?,+),
        ac_recomp(+,+,?),
    match(+,?),
        match_arguments(+,+,+),
        split_two_ways(+,?,?).

%   replace OldA by NewA in one elment of Old, giving New.

corresponding_arguments([OldA|Tail], OldA, [NewA|Tail], NewA).
corresponding_arguments([Head|Tail], OldA, [Head|Rest], NewA) :-
        corresponding_arguments(Tail, OldA, Rest, NewA).


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

%   decomp(Term, List) and recomp(Term, List) are generalisations of univ,
%   i.e. Term =.. List, treating the four known associative commutative
%   operators as function symbols having any number of arguments.

%   They are called in the patterns
%       decomp(Old, [Op|Olds]),         %   var(Op)
%       any1(<foo>, Olds, News),
%       recomp(New, [Op|News]),
%   in collect and attract, and elsewhere in the form
%       decomp(Old, [+|_])      trig_fac,multiply_through,weaknf
%       recomp(New, [+|_])      make_poly.

%   ac_op(Op, X, Y, X Op Y, Idn) means that Op is known to be a commutative
%   associative operator, that X Op Y =.. [Op,X,Y], and that Idn Op X = X
%   i.e. Idn is the identity of Op.  All four operators have an identity.
%   The fifth clause is a hack for 1/(X*Y), but is still true.

ac_op(+, X, Y, X+Y, 0)      :- !.
ac_op(*, X, Y, Y*X, 1)      :- !.       %   note reversal!
ac_op(&, X, Y, X&Y, true)   :- !.       %   conjunction
ac_op(#, X, Y, X#Y, false) :- !.        %   disjunction
%%ac_op(*, X^N, Y^N, (Y*X)^N, 1) :- !.


decomp(Term, [Op|Args]) :-
        functor(Term, Op, 2),
        ac_op(Op, _, _, _, _), !,
        ac_decomp(Term, Op, Args, []).
%%decomp((X*Y)^(-1), [*|Args]) :-                  %   special hack
%%      ac_decomp((X*Y)^(-1), *, Args, []).
decomp(Term, List) :-
        Term =.. List.


        ac_decomp(Term, Op, [Term|R], R) :-
                var(Term), !.
        ac_decomp(Term, Op, L, R) :-
                ac_op(Op, X, Y, Term, _), !,
                ac_decomp(X, Op, L, M), !,
                ac_decomp(Y, Op, M, R).
        ac_decomp(Term, Op, [Term|R], R).



recomp(Term, [Op|Args]) :-
        ac_op(Op, _, _, _, _), !,
        ac_recomp(Args, Op, Term).
recomp(Term, List) :-
        Term =.. List.

        ac_recomp([[]|Args], Op, Term) :- !,
                ac_recomp(Args, Op, Term).
        ac_recomp([Exp], Op, Term) :- !,
                Term = Exp.
        ac_recomp([Exp|Args], Op, Term) :- !,
                ac_op(Op, Exp, Mid, Term, _), !,
                ac_recomp(Args, Op, Mid).
        ac_recomp([], Op, Term) :-
                ac_op(Op, _, _, _, Term).


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

%   match two terms, using the associativity and commutativity of + and *.

match(Lhs, Rhs) :-
        functor(Lhs, Op, 2),
        ac_op(Op, Arg1, Arg2, Rhs, _), !,
        decomp(Lhs, [Op|Olds]), !, c1         c2
        split_two_ways(Olds, [C1|Cs1], [C2|Cs2]),
  *     recomp(D1, [Op|C1]Cs1]),
        recomp(D2, [Op|C2]Cs2]),
        match(D1, Arg1),
        match(D2, Arg2).

match(Lhs, Lhs) :-                  %   atoms match themselves
        atomic(Lhs), !.

match(Neg, -1*Pos) :-               %   hack round the representation of
        number(Neg),                %   negative numbers
        eval(Neg < 0),              %   rationals are around now!
        eval(-Neg, Pos), !.
match(-1*Pos, Neg) :-               %  can't happen if Lhs is tidied first
        number(Neg),
        eval(Neg < 0),
        eval(-Neg, Pos), !.
```

```
match(Lhs, Rhs) :-
        functor(Lhs, Functor, Arity),
        functor(Rhs, Functor, Arity), !,
        match_arguments(Arity, Lhs, Rhs).


        match_arguments(0, Lhs, Rhs) :- !.
        match_arguments(N, Lhs, Rhs) :-
                arg(N, Lhs, LhsNth),
                arg(N, Rhs, RhsNth),
                match(LhsNth, RhsNth),
                M is N-1,
                match_arguments(M, Lhs, Rhs).


        split_two_ways([Head|Tail], A, B) :-
                split_two_ways(Tail, A1, B1),
                (    A = [Head|A1], B = B1
                |    B = [Head|B1], A = A1
                ).
        split_two_ways([], [], []).

/* Obsolete Code
%-------------------------------------------------------------------------%

%   apply Proc to some member of Old to get New.
%   This belongs is some utility file, not here.

any1(Proc, [Old|Olds], [New|Olds]) :-
        apply(Proc, [Old,New]).                         %   <- apply
any1(Proc, [Old|Olds], [Old|News]) :-
        any1(Proc, Olds, News).


%   rewrite Old into New using Rule.

rewrite(Rule, Old, New) :-                                      %   +, *, &, #
        functor(Old, Op, 2),
        ac_op(Op, Arg1, Arg2, Lhs, _), !,
        decomp(Old, [Op|Olds]), !,
        apply(Rule, [Lhs,Rhs]),                                 %   <- apply
        split_two_ways(Olds, [C1|Cs1], [B|Bs1]),
        split_two_ways([B|Bs1], [C2|Cs2], Rest),
        recomp(D1, [Op,C1|Cs1]),
        recomp(D2, [Op,C2|Cs2]),
        match(D1, Arg1),
        match(D2, Arg2),
        recomp(New, [Op,Rhs|Rest]).

rewrite(Rule, Old, New) :-                                      %   other operators
        functor(Old, Functor, Arity),
        functor(Lhs, Functor, Arity),
        apply(Rule, [Lhs,New]),                                 %   <- apply
        match(Old, Lhs).

%   apply Proc recursively to Old to get New.

recurse(Proc, Old, New) :-
        Old =.. [Functor|OldArgs],
        New =.. [Functor|NewArgs],
        maplist(Proc, OldArgs, NewArgs), !.

%   apply Proc to Old directly to get New.

try_rewrite(Proc, Old, New) :-
        apply(Proc, [Old,New]), !.
try_rewrite(Proc, Old, Old).

%   apply Rule to Old (as often as possible) to get New.

try_rewrite2(Rule, Old, New) :-
        rewrite(Rule, Old, Exp), !,
        try_rewrite2(Rule, Exp, New).
try_rewrite2(Rule, Old, Old).

%   select a pair of numbers X and Y from list L, residue R.

pairint(L, X,Y, R) :-
        select(X, L, S), integer(X), %   number(X) ??
        select(Y, S, R), integer(Y). %   number(Y) ??


END of obsolete code.  */


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

%   Given a Term, discover all the constants, atoms, and functors occuring
%   in it.  The Term is known to be ground.

functors_in(Term, List) :-
        functors_in(Term, L, []),
        sort(L, List).

        functors_in(Term, [Term|R], R) :-
                atom(Term), !.
        functors_in(Term, [Abso|R], R) :-
                number(Term), !,
                eval(abs(Term), Abso).
        functors_in(Term, [Head|L], R) :-
                functor(Term, Functor, Arity),
                functor(Head, Functor, Arity), !,
                functors_in(Arity, Term, L, R).

                functors_in(0, Term, R, R) :- !.
                functors_in(N, Term, L, R) :-
                        arg(N, Term, Argument),
                        functors_in(Argument, L, M),
                        K is N-1, !,
                        functors_in(K, Term, M, R).
```

```
/* INT : Finds intervals of terms in PRESS

                                                          Alan Bundy
                                                          Updated: 30 May 82
        Alan Bundy 19.12.79
        Revised version 14.3.80
        Further revised 26.3.81
        Cosmetics by Lawrence 18 June 81
        (Couple of small fixes since then)
        Added clauses for positive etc for
        special atoms like pi and e
 */

/* EXPORT */

  :- public     vet/2,
                positive/1,
                negative/1,
                non_neg/1,
                non_pos/1,
                non_zero/1,
                acute/1,
                obtuse/1,
                non_reflex/1,

                less_than/2,            % Used in a \+

                find_int/2,             % Exported for convenience
                int_apply/3.


/* IMPORT */
/*
                error/3                 from  UTIL:TRACE

                memberchk/2             from  UTIL:SETROU

                number/1                from  LONG
                eval/1
                eval/2

                measure/2               from  notional Mecho database
                quantity/1
                angle/3
                incline/3
                concavity/2
                slope/2
                partition/2

                special_atom/1          from  ARITH:FACTS
*/


/* MODES */

  :- mode       vet(+,?),
                positive(+),
                negative(+),
                non_neg(+),
                non_pos(+),
                non_zero(+),
                acute(+),
                obtuse(+),
                non_reflex(+),

                    gen_combine(+,?),
                    combine(+,+,?),
                    in(+,+),
                    sub_int(+,+),
                    below(+,+),
                    disjoint(+,+),
                    overlap(+,+),
                    marker_flip(?,?),

                        default_interval(?),
                    find_int(+,?),
                        find_int2(+,-),
                        find_int_args(+,-,-),
                        find_simple_int(+,-),
                            make_assumption_positive(+),

                    int_apply(+,+,-),
                        int_apply_all(+,+,-),
                        all_are_contained(+,+),
                        make_regions(+,+,-),
                            split(+,+,+,-),
                                split1(+,+,-),
                            cartesian_product(+,+,-,?),
                                cart_prod(+,+,+,-,?),
                        find_limits(+,+,+,-),
                            clean_up(+,-),
                            limits(+,+,+,+,?),
                            set_bnds(+,+,+,-),
                                updown_flip(+,+,-),
                                set_bnd(+,+,-),

                    order(+,+,?,?),
                    less_than(+,+),
                    calc(+,+,?),
                        breakup_bnds(+,-,-),
                    comb(+,?),

                    mono(+,?,?),

                    classify(+,-),
                        interval(+,-,-),
                        collect_intervals(+,+,-),
                        quad(+,+,+,?).


/*
    Data structures

        <interval>      has form        i(LMarker,Bottom,Top,RMarker)
        <boundary>      has form        b(N,Marker)

                where:
                        Bottom, Top, N             are  <numbers>
                        LMarker, RMarker, Marker   are  one of {open,closed}
        An interval ranges between Bottom and Top and is open or closed at
        the ends depending on LMarker (for Bottom) and RMarker (for Top).

        A boundary is an end of an interval. There are operations defined
        over these boundaries which are then used to help define the
        operations over intervals. Note that the notion of a boundary does
        NOT involve any specific end of an interval (ie Top/Bottom). They
        are a generalisation over all such ends.

*/

[% @@@ - marker (top of code)


/****************************************/
/* Use interval information - top level */
/****************************************/

                        % Check that solution is admissible

vet(true,true).

vet(false,false).

vet(A&B,A1&B1) :- vet(A,A1), vet(B,B1).

vet(A#B,A1#B1) :- vet(A,A1), vet(B,B1).

vet(A=B,A=B) :-
        find_int(A,IntA), find_int(B,IntB),
        overlap(IntA,IntB),
        !.

vet(A=B,false).


                        % X is positive, negative, acute, etc.

positive(X) :- atom(X),special_atom(X),!.  %Hack for e and pi

positive(X) :- find_int(X,i(L,B,T,R)), less_than(b(0,closed),b(B,L)).

negative(X) :- find_int(X,i(L,B,T,R)), less_than(b(T,R),b(0,closed)).

non_neg(X) :- atom(X),special_atom(X),!.  %Hack for e and pi

non_neg(X) :- find_int(X,i(L,B,T,R)), less_than(b(0,open),b(B,L)).

non_pos(X) :- find_int(X,i(L,B,T,R)), less_than(b(T,R),b(0,open)).

non_zero(X^N) :- !, non_zero(X).         %ad hoc patch (replaces negative(N)

non_zero(X) :- atom(X),special_atom(X),!. % Hack for e and pi

non_zero(X) :-
        find_int(X,i(L,B,T,R)),
        ( less_than(b(0,closed),b(B,L)) ; less_than(b(T,R),b(0,closed)) ),
        !.

acute(X) :-
```

```
        find_int(X,i(L,B,T,R)),
        less_than(b(0,open),b(B,L)),
        less_than(b(T,R),b(90,open)).

obtuse(X) :-
        find_int(X,i(L,B,T,R)),
        less_than(b(90,open),b(B,L)),
        less_than(b(T,R),b(180,open)).

non_reflex(X) :-
        find_int(X,i(L,B,T,R)),
        less_than(b(0,open),b(B,L)),
        less_than(b(T,R),b(180,open)).



/****************************************/
/*      Manipulating Intervals          */
/****************************************/


                          % Combine a list of intervals by sweeping list and
                          %  accumulating the combined intervals.

gen_combine([FirstInt|RestInts],Result)
     :- gen_combine(RestInts,FirstInt,Result).


gen_combine([],Result,Result).

gen_combine([Int|RestInts],Acc,Result)
     :- combine(Int,Acc,NewAcc),
        gen_combine(RestInts,NewAcc,Result).


                          % Combine x and y intervals

combine(i(Lx,Bx,Tx,Rx), i(Ly,By,Ty,Ry), i(L,B,T,R)) :-
        order(b(Tx,Rx),b(Ty,Ry),_,b(T,R)),
        order(b(Bx,Lx),b(By,Ly),b(B,L),_).


                          % Number N is contained in interval

in(N,i(L,B,T,R)) :- !,
        sub_int(i(closed,N,N,closed),i(L,B,T,R)).


                          % x interval is contained in second interval

sub_int(i(Lx,Bx,Tx,Rx),i(L,B,T,R)) :-
        marker_flip(L,L1), marker_flip(R,R1),
        less_than(b(B,L1),b(Bx,Lx)), less_than(b(Tx,Rx),b(T,R1)).

                          % x interval is wholly below y interval

below(i(Lx,Bx,Tx,Rx),i(Ly,By,Ty,Ry)) :-
        less_than(b(Tx,Rx),b(By,Ly)), !.

                          % x and y intervals are disjoint
```

```
disjoint(IntX,IntY) :- below(IntX,IntY), !.
disjoint(IntX,IntY) :- below(IntY,IntX), !.

                        % x and y intervals overlap

%% overlap(IntX,IntY) :- \+ disjoint(IntX,IntY).

overlap(IntX,IntY) :- disjoint(IntX,IntY), !, fail.
overlap(_,_).


                        % open and closed are opposites
                        %  (this is how to flip them)

marker_flip(open,closed) :- !.
marker_flip(closed,open).



/****************************************/
/* X lies in closed or open interval    */
/****************************************/


                % Worst case default for intervals

default_interval(i(open,neginfinity,infinity,open)).


                % Let's try to do better.

find_int(X,Interval)
     :- find_int2(X,Result),            % guarantee mode (+,-)
        Interval = Result.


                        % Catch variables (shouldn't be there!)

find_int2(V,_)
     :- var(V),
        !,
        error('Interval package given variable: %w',[V],fail).

                        % Base cases
                        %  Numbers have point intervals
                        %  Symbols (atoms) have various special cases

find_int2(X,i(closed,X,X,closed)) :- number(X), !.

find_int2(X,Interval) :- atom(X), !, find_simple_int(X,Interval).


                        % Special case normalisation

                        % Convert ^(-1) to 1/

find_int2(X^(-1), Int) :- !,
        find_int2(1/X, Int).
```

```
                        % Deal with exponentials to even power

find_int2(X^N, i(L,B,T,R)) :-
        even(N), !,
        find_int(abs(X), i(Lx,Bx,Tx,Rx)),
        calc(^,[b(Bx,Lx),b(N,closed)],b(B,L)),
        calc(^,[b(Tx,Rx),b(N,closed)],b(T,R)).

                        % Convert cosecant to sine

find_int2(csc(X), Int) :- !, find_int2(1/sin(X), Int).

                        % Convert secant to cosine

find_int2(sec(X), Int) :- !, find_int2(1/cos(X), Int).

                        % Convert cotangent to tangent

find_int2(cot(X), Int) :- !, find_int2(1/tan(X), Int).


                        % General case
                        %  Recursively find intervals for arguments and
                        %  then int_apply to sort this out. This will use
                        %  monotonicity of F to calculate interval of Term
                        %  from arguments.

find_int2(Term,Int) :-
        find_int_args(Term,F,IntList),
        int_apply(F,IntList,Int),
        !.


                        % If the general case fails

find_int2(sin(X), i(closed,(-1),1,closed)) :- !.
find_int2(cos(X), i(closed,(-1),1,closed)) :- !.

find_int2(X,Default) :- default_interval(Default).


                        % Find a list of intervals corresponding to the
                        %  arguments of Term. Also return the functor.

find_int_args(Term,Fn,IntList)
     :- functor(Term,Fn,Arity),
        find_int_args(1,Arity,Term,IntList).


find_int_args(N,Max,_,[]) :- N > Max, !.

find_int_args(N,Max,Term,[Int|IntRest])
     :- arg(N,Term,Arg),
        find_int2(Arg,Int),
        N1 is N+1,
        find_int_args(N1,Max,Term,IntRest).
```

```
                         % Find the interval for a simple symbol
                         %  This involves looking to see if we know
                         %  anything special about the symbol which will
                         %  help us.
                         % Ad hoc patch for gravity - proper solution means
                         %  allowing equations between quantities and defining
                         %  g as measure(g,32,ft/sec^2).
                         % Otherwise try to classify symbol (if it is an angle)
                         % Otherwise assume all quantities are positive
                         %       (possibly extreme?)
                         % If there is no useful info we must use the default.

find_simple_int(g,i(open,1,infinity,open)) :- !.

find_simple_int(X,Int) :- classify(X,Int), !.

find_simple_int(M,i(open,0,infinity,open)) :-
        measure(Q,M), quantity(Q),
        !,
        make_assumption_positive(M).

find_simple_int(X,Default) :- default_interval(Default).



                         % Make and remember assumption

make_assumption_positive(X) :- assumed_positive(X), !.

make_assumption_positive(X)
     :- assert( assumed_positive(X) ),
        trace('I assume %t positive.\n',[X],1).



/****************************************************************/
/* Find interval of function from intervals of its arguments */
/****************************************************************/


                                 % Simple case

int_apply(F,Region,Int) :-
        mono(F,Is,Mono),
        all_are_contained(Region,Is),
        !,
        find_limits(F,Region,Mono,Int).

                                 % Complex Case

int_apply(F,Region,Int) :-
        mono(F,MRegion,Mono),
        make_regions(Region,MRegion,NewRegions),
        int_apply_all(NewRegions,F,IntervalSet),
        !,
        gen_combine(IntervalSet,Int).



                         % int_apply all intervals in a set (list)
```

```
int_apply_all([],_,[]).

int_apply_all([Region1|Rest],F,[Int1|IRest])
     :- int_apply(F,Region1,Int1),
        int_apply_all(Rest,F,IRest).



                        % All the argument intervals are sub intervals of
                        %  the corresponding monotonic intervals for the
                        %  function (from mono). (ie maplist sub_int down
                        %  the two "argument" lists).

all_are_contained([],[]).

all_are_contained([ArgInt|ArgRest],[FInt|FRest])
     :- sub_int(ArgInt,FInt),
        all_are_contained(ArgRest,FRest).



                        % Given the list of actual intervals and the list
                        %  of monotonic intervals for the function build
                        %  a set of similar interval lists, derived from the
                        %  actual interval list, but such that each element
                        %  of each list in the set is wholly inside or outside
                        %  its corresponding monotonic function interval.
                        % This amounts to case splitting the actual interval
                        %  list into a set of intervals for more tractable
                        %  (sub) regions in the nD space.
                        % Implemented by splitting lists to form a list of
                        %  sets and taking the nD cartesian product. Note
                        %  that both split/4 and cartesian_product/4 perform
                        %  order reversals - which cancel each other out.

make_regions(Region,MRegion,NewRegions)
     :- split(Region,MRegion,[],ListOfSets),
        cartesian_product(ListOfSets,[],NewRegions,[]).



                        % Given the list of actual intervals and the list of
                        %  monotonic intervals for the function, we build
                        %  a list of n sets, where n is the arity of the
                        %  function (ie the length of the lists) and where
                        %  each set contains intervals which are wholly inside
                        %  or outside the corresponding monotonic function
                        %  intervals, such that the intervals in each set
                        %  would combine to form the corresponding actual
                        %  interval.
                        % The combining property follows from the way we split
                        %  up the actual intervals.
                        % The sets produced at the moment will only ever have
                        %  number of members m such that: 1 =< m =< 3.
                        %  The following special representations are used for
                        %  these cases:
                        %               singleton(A)
                        %               pair(A,B)
                        %               triple(A,B,C)
                        %  In fact the code will currently never produce sets
                        %  of 3 elements (triples), but I (Lawrence) think
                        %  this is probably a bug so have left the option, and
                        %  this comment, around til we see.
                        % Note that the list of sets built will be in reverse
                        %  order compared with the "argument" lists. This is
                        %  is implemented by an extra accumulator argument
                        %  (should be [] to start with) onto which each Set
                        %  is pushed.

split([],[],Result,Result).

split([ArgInt|ArgRest],[FInt|FRest],Sofar,Result)
     :- split1(ArgInt,FInt,Set),
        split(ArgRest,FRest,[Set|Sofar],Result).


                                % Intx wholly within Int

split1(Intx,Int,singleton(Intx)) :-
        sub_int(Intx,Int),
        !.

                                % Intx and Int overlap with Intx leftmost

split1(i(Lx,Bx,Tx,Rx), i(L,B,T,R), pair(i(L,B,Tx,Rx),i(Lx,Bx,B1,L1)) ) :-
        marker_flip(R,R1), marker_flip(L,L1),
        marker_flip(Lx,Lx1),
        correct(B,B1),
        less_than(b(Tx,Rx),b(T,R1)),
        \+ less_than(b(Tx,Rx),b(B,L)),
        less_than(b(Bx,Lx1),b(B,L)), !.



                        % Given a list of n sets produce the a set of the
                        %  elements from the nD cartesian product of the sets.
                        %  The incoming sets are represented with special
                        %  functors as there are only a few special cases (see
                        %  split). The resulting product set is represented as
                        %  a list. Each element will itself be a list (of n
                        %  intervals) where the order of this element list will
                        %  be the reverse of the order in which the items
                        %  were found in the original list of sets.
                        % The implementation involves an accumulator for the
                        %  (partial) element being built and uses the
                        %  difference list technique to build the final set
                        %  of elements (repn as a list).

cartesian_product([],Element,[Element|Z],Z).

cartesian_product([First|Rest],PartialElement,ProductSet,Z)
     :- cart_prod(First,Rest,PartialElement,ProductSet,Z).



cart_prod(singleton(A),Rest,PartialElement,PSet,Z)
     :- cartesian_product(Rest,[A|PartialElement],PSet,Z).

cart_prod(pair(A,B),Rest,PartialElement,PSet0,Z)
```

```
      :- cartesian_product(Rest,[A|PartialElement],PSet0,PSet1),
         cartesian_product(Rest,[B|PartialElement],PSet1,Z).

cart_prod(triple(A,B,C),Rest,PartialElement,PSet0,Z)
      :- cartesian_product(Rest,[A|PartialElement],PSet0,PSet1),
         cartesian_product(Rest,[B|PartialElement],PSet1,PSet2),
         cartesian_product(Rest,[C|PartialElement],PSet2,Z).



                             % Calculate Bottom and Top of Interval

find_limits(F,Region,Mono,Int) :-
         limits(bottom,F,Region,Mono,b(B,L)),
         limits(top,F,Region,Mono,b(T,R)),
         clean_up(i(L,B,T,R), Int).


                             % Hack to clear up various funnies

clean_up(i(_,undefined,_,_), Int) :- !, default_interval(Int).
clean_up(i(_,_,undefined,_), Int) :- !, default_interval(Int).
clean_up(i(L,B,0,R), i(L,B,-(0),R)) :- !.
clean_up(Int, Int).

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


                             % Calculate limit for a particular boundary

limits(TopBot,F,Region,Mono,Boundary)
      :- set_bnds(Mono,TopBot,Region,BoundaryList),
         calc(F,BoundaryList,Boundary).



                             % Form a boundary list from an interval list
                             %  given various details - up+down x top+bottom.

set_bnds([],_,[],[]).

set_bnds([Mono|MRest],TopBot,[Int|IRest],[Bnd|BRest])
      :- updown_flip(TopBot,Mono,NewMono),
         set_bnd(NewMono,Int,Bnd),
         set_bnds(MRest,TopBot,IRest,BRest).


         updown_flip(top,UD,UD).
         updown_flip(bottom,up,down) :- !.
         updown_flip(bottom,down,up).


         set_bnd(up,  i(L,B,T,R), b(T,R)).
         set_bnd(down,i(L,B,T,R), b(B,L)).



/******************************************/
```

```
/*      Manipulating Boundaries          */
/*****************************************/


                        % Put boundaries in order

                                        % Boundaries are identical
order(Bnd,Bnd,Bnd,Bnd) :- !.
                                        % One of Mis is closed
order(b(N,M1),b(N,M2),b(N,closed),b(N,closed)) :- !.
                                        % Numbers are different, N1 smallest
order(b(N1,M1),b(N2,M2),b(N1,M1),b(N2,M2)) :-
        eval(N1 < N2), !.
                                        % N2 is smallest
order(b(N1,M1),b(N2,M2),b(N2,M2),b(N1,M1)).



                        % Ordering of boundaries
                        %  (assumes intervals are consecutive)

less_than(b(X,Mx),b(Y,My)) :-
        comb([Mx,My],M),
        less_than_eval(M,X,Y).


less_than_eval(open,X,Y) :- eval( X =< Y ).

less_than_eval(closed,X,Y) :- eval( X < Y ).



                        % Apply Function F to a boundary list
                        %  Do this by combining the boundary markers and
                        %  applying F to the numbers.

calc(F,BoundaryList,b(X,M)) :-
        breakup_bnds(BoundaryList,Markers,Numbers),
        comb(Markers,M),
        Term =.. [F|Numbers],
        eval(Term,X),
        !.


breakup_bnds([],[],[]).

breakup_bnds([b(N,M)|Rest],[M|MRest],[N|NRest])
     :- breakup_bnds(Rest,MRest,NRest).



                        % Combine boundary markers
                        %  Result = open if any of the inputs is open

comb(MarkerList,Result) :- memberchk(open,MarkerList), !, Result = open.

comb(_,closed).
```

```
/*********************************************/
/* Monotonicity of Functions in each Interval */
/*********************************************/

/* unary minus */
mono(-, [i(closed,neginfinity,infinity,closed)], [down]).

/* addition */
mono(+,[i(closed,neginfinity,infinity,closed),
        i(closed,neginfinity,infinity,closed)], [up,up]).

/* binary minus */
mono(-,[i(closed,neginfinity,infinity,closed),
        i(closed,neginfinity,infinity,closed)], [up,down]).

/* absolute value */
mono(abs,[i(closed,neginfinity,-(0),closed)], [down]).
mono(abs,[i(closed,0,infinity,closed)], [up]).

/* multiplication */
mono(*,[i(closed,0,infinity,closed), i(closed,0,infinity,closed)],
        [up,up]).
mono(*,[i(closed,0,infinity,closed), i(closed,neginfinity,-(0),closed)],
        [down,up]).
mono(*,[i(closed,neginfinity,-(0),closed), i(closed,0,infinity,closed)],
        [up,down]).
mono(*,[i(closed,neginfinity,-(0),closed), i(closed,neginfinity,-(0),closed)],
        [down,down]).



/* division */
mono(/,[i(closed,0,infinity,closed), i(closed,0,infinity,closed)],
        [up,down]).
mono(/,[i(closed,0,infinity,closed), i(closed,neginfinity,-(0),closed)],
        [down,down]).
mono(/,[i(closed,neginfinity,-(0),closed), i(closed,0,infinity,closed)],
        [up,up]).
mono(/,[i(closed,neginfinity,-(0),closed), i(closed,neginfinity,-(0),closed)],
        [down,up]).


/* exponentiation */
mono(^,[i(open,0,infinity,closed),i(closed,0,infinity,closed)],
        [up,up]).
mono(^,[i(open,0,infinity,closed),i(closed,neginfinity,-(0),closed)],
        [down,up]).


/* logarithm */
mono(log,[i(closed,0,infinity,closed),i(closed,0,infinity,closed)],
        [down,up]).

/* sine */
mono(sin,[i(closed,(-90),90,closed)],[up]).
mono(sin,[i(closed,90,270,closed)],[down]).
mono(sin,[i(closed,270,450,closed)],[up]).

/* cosine */
mono(cos,[i(closed,0,180,closed)],[down]).
mono(cos,[i(closed,180,360,closed)],[up]).

/* tangent */
mono(tan,[i(open,(-90),90,open)],[up]).
mono(tan,[i(open,90,270,open)],[up]).
mono(tan,[i(open,270,450,open)],[up]).

/* inverse sine */
mono(arcsin,[i(closed,(-1),1,closed)],[up]).

/* inverse cosine */
mono(arccos,[i(closed,(-1),1,closed)],[down]).

/* inverse tangent */
mono(arctan,[i(open,neginfinity,infinity,open)],[up]).

/* inverse cosecant */
mono(arccsc,[i(closed,neginfinity,(-1),closed)],[down]).
mono(arccsc,[i(closed,1,infinity,closed)],[down]).

/* inverse secant */
mono(arcsec,[i(closed,neginfinity,(-1),closed)],[up]).
mono(arcsec,[i(closed,1,infinity,closed)],[up]).

/* inverse cotangent */
mono(arccot,[i(closed,neginfinity,-(0),open)],[down]).
mono(arccot,[i(open,0,infinity,closed)],[down]).




/***************************************************/
/*  Calculate Interval of Angle from Curve Type  */
/***************************************************/


                        % We classify a symbol using semantic information
                        %  from the (Mecho) database. Calls which are to
                        %  this database (notionally, Press does not really
                        %  share the same object-level database) are marked
                        %  as such.
                        % This method is only appropriate if the symbol is an
                        %  <angle>, and tries to find the interval of the
                        %  angle using general principles about curve types.

classify(Angle, Int ) :-
        measure(Q, Angle ),                      % database
        angle(Point, Q, Curve ), !,              % database
        interval(angle, Curve, Int ).

classify(Angle, Int ) :-
        measure(Q, Angle ),                      % database
        incline(Curve, Q, Point ), !,            % database
        interval(incline, Curve, Int ).


                         % Find interval from curve shape

                                 % For simple curves
```

```
interval(AI, Curve, Int ) :-
        concavity(Curve, Conv ),                   % database
        slope(Curve, Slope ), !,                   % database
        quad(AI, Slope, Conv, Int ).

                                  % For complex curves
interval(AI, Curve, Int ) :-
        partition(Curve, Clist ), !,               % database
        collect_intervals(Clist, AI, Rlist),
        gen_combine(Rlist, Int ).



                          % Collect up a list of intervals for all the parts
                          %  of a partitioned curve.

collect_intervals([],_,[]).

collect_intervals([First|Rest],AI,[FirstInt|RestInt])
     :- interval(AI,First,FirstInt),
        collect_intervals(Rest,AI,RestInt).



                          % Information about properties of simple curves
                          %  The interval depends on both the slope and the
                          %  concavity.

quad(angle,left,right,i(closed,0,90,closed)) :- !.
quad(incline,left,right,i(closed,90,180,closed)) :- !.

quad(angle,right,right,i(closed,90,180,closed)) :- !.
quad(incline,right,right,i(closed,180,270,closed)) :- !.

quad(angle,left,left,i(closed,180,270,closed)) :- !.
quad(incline,left,left,i(closed,270,360,closed)) :- !.

quad(angle,right,left,i(closed,270,360,closed)) :- !.
quad(incline,right,left,i(closed,0,90,closed)) :- !.

quad(angle,left,stline,i(open,180,270,open)) :- !.
quad(incline,left,stline,i(open,270,360,open)) :- !.

quad(angle,right,stline,i(open,270,360,open)) :- !.
quad(incline,right,stline,i(open,0,90,open)) :- !.

quad(angle,hor,stline,i(closed,270,270,closed)) :- !.
quad(incline,hor,stline,i(closed,0,0,closed)) :- !.

quad(angle,vert,stline,i(closed,180,180,closed)) :- !.
quad(incline,vert,stline,i(closed,270,270,closed)) :- !.



/* JOBS TO DO

        write symbolic version for finding max/mins

        use monotonicity in > >= etc Isolation rules
*/
```

```
%   Press:Diff.                                 Updated: 12 Sept 81

%==========================================================================%
%                          Differential Calculus           19.2.81         %
%==========================================================================%

:- public diffwrt/3.
:- mode
     diffwrt(+, -, +),
          dx(+, -, +),
               exactly_one_arg(+, +, -),
                    exactly_one_arg(+, +, +, ?).


diffwrt(Exp, Ans, Var) :-
          trace('Differentiating %c with respect to %t\n', [Exp, Var], 1),
          dx(Exp, Der, Var),
          tidy(Der, Ans),
          trace('   gives : %c\n', [Ans], 1), !.


dx(Exp, 0, X) :-
          freeof(X, Exp), !.

dx(X, 1, X) :- !.

dx(X^N, N*X^M, X) :-
          freeof(X, N),
          tidy(N-1, M), !.

dx(Exp^X, Exp^X*log(e,Exp)^(-1), X) :-
          freeof(X, Exp), !.

dx(log(e,X), X^(-1), X) :- !.

dx(tan(X), sec(X)^2, X) :- !.

dx(cot(X), -1*cosec(X)^2, X) :- !.

dx(sec(X), sec(X)*tan(X), X) :- !.        %  is this a good way to say it?

dx(arcsin(X), (1 + -1*X^2)^(-2 ^ -1), X) :- !.

dx(cosec(X), -1*cos(X)*cosec(X)^2, X) :- !.

dx(arcsin(X), (1 + -1*X^2)^(-2 ^ -1), X):- !.

dx(cosec(X), -1*cos(X)*cosec(X)^2, X):- !.

dx(arctan(X), (1+X^2)^(-1), X):- !.

dx(sin(X), cos(X), X) :- !.

dx(cos(X), -1*sin(X), X) :- !.

dx(A+B, DA+DB, X) :- !,
          dx(A, DA, X), !,
          dx(B, DB, X).

dx(C*A, C*DA, X) :-
```

```
        freeof(X, C),  !,  dx(A, DA, X).

dx(A*C, DA*C, X) :-
        freeof(X, C),  !,  dx(A, DA, X).

dx(A/C, DA/C, X) :-
        freeof(X, C),  !,  dx(A, DA, X).

dx(C/A, -1*C*DA/A^2, X) :-
        freeof(X, C), !, dx(A, DA, X).

dx(A*B, A*DB + B*DA, X) :- !,
        dx(A, DA, X), !, dx(B, DB, X).

dx(A/B, (B*DA + -1*A*DB)/B^2, X) :- !,
        dx(A, DA, X), !, dx(B, DB, X).

dx(Exp, Exp1*Arg1, X) :-
        exactly_one_arg(X, Exp, Arg),
        Arg \== X, !,
        gensym(var, T),
        subst(Arg=T, Exp, Mid),         dx(Mid, Mid1, T),
        subst(T=Arg, Mid1, Exp1), !,    dx(Arg, Arg1, X).

%   check that there is exactly one argument of Exp containing Term,
%   and return that argument as Arg.

exactly_one_arg(Term, Exp, Arg) :-
        functor(Exp, _, N),
        exactly_one_arg(N, Term, Exp, Arg).

        exactly_one_arg(0, Term, Exp, Ans) :- !, nonvar(Ans).
        exactly_one_arg(N, Term, Exp, Ans) :-
                arg(N, Exp, Arg),
                contains(Term, Arg), !,
                M is N-1, Arg = Ans, !,
                exactly_one_arg(M, Term, Exp, Ans).
        exactly_one_arg(N, Term, Exp, Ans) :-
                M is N-1,
                exactly_one_arg(M, Term, Exp, Ans).
```

```
/*                PROVER     19.2.81    */
/********************************
 THEOREM PROVERS
********************************/

/*FIND MAXIMUM OF SET*/

maximum(IneqC,AnsC) :-
        andtodot(IneqC,IneqL),
        maximum1(IneqL,AnsL),
        dottoand(AnsL,AnsC),
        !.

maximum1([],[]) :- !.

maximum1([Ineq],[Ineq]) :- !.

maximum1([Ineq|Rest],Ans) :-
        some(smaller(Ineq),Rest), !,
        maximum1(Rest,Ans).

maximum1([Ineq|Rest],[Ineq]) :-
        checklist(bigger(Ineq),Rest), !.

maximum1([Ineq|Rest],[Ineq|Ans]) :-
        maximum1(Rest,Ans), !.


/*INEQ1 DOMINATES INEQ2*/
smaller(Ineq2,Ineq1) :- bigger(Ineq1,Ineq2), !.

bigger(X>=Y,X>=Z) :- prove(Y>=Z), !.
bigger(X>Y,X>Z) :- prove(Y>=Z), !.
bigger(X>Y,X>=Z) :- prove(Y>=Z), !.
bigger(X>=Y,X>Z) :- prove(Y>Z), !.

/* Prove simple inequalities etc*/

prove(X>=Y) :- poly_form(X+(-1*Y) , E), non_neg(E), !.

prove(X>Y) :- poly_form(X+(-1*Y), E), positive(E), !.

prove(X=\=Y) :- poly_form(X+(-1*Y), E), non_zero(E), !.

prove(X=Y) :- poly_form(X+(-1*Y), 0), !.

/* Simplify formulae into true or false if possible*/

simplify(F,true) :- prove(F), !.

simplify(F,false) :- negation(F,NF), prove(NF), !.

simplify(F,F) :- !.

/* Negation of formula */
negation(F,NF) :- negation1(F,NF), !.
negation(F,NF) :- negation1(NF,F), !.

negation1(A=B,A=\=B).
negation1(A>=B,B>A).
```

```
%   Press:Misc.                             Updated: 30 June 82
%   Basic utilities for Press.  Written by Alan Bundy 31.8.80.
%   additional routines by Leon Sterling, Richard O'Keefe, and Bernard Silver

%   flag(tflag,_,1)  has been moved to  Press:Filin.

%   convert lists to conjunctions and vice versa.

:- public dottoand/2,                       andtodot/2.
:-  mode  dottoand(+, -),        andtodot(+, -).

dottoand([], true) :- !.
dottoand([Head|Tail], Head & Rest) :-
        dottoand(Tail, Rest).

andtodot(true, []) :- !.
andtodot(Head & Rest, [Head|Tail]) :- !,
        andtodot(Rest, Tail).
andtodot(Exp, [Exp]).

% Same for disjunctions
:- public ortodot/2,  dottoor/2.

:- mode ortodot(+,-),dottoor(+,-).

ortodot(false,[]) :- !.
ortodot(A#B,[A|T]) :- ortodot(B,T),!.
ortodot(A,[A]) :- !.

dottoor([],false) :- !.
dottoor([A],A) :- !.
dottoor([A|B],A#T) :- dottoor(B,T),!.

%   Occurrence clauses.

:- public freeof/2, singleocc/2, contains/2, mult_occ/2, mult_Occ/2.

freeof(Term, Exp)        :- occ(Term, Exp, 0), !.
singleocc(Term, Exp)     :- occ(Term, Exp, 1), !.
contains(Term, Exp)      :- occ(Term, Exp, N), N > 0, !.
mult_Occ(Term,Exp)       :- occ(Term, Exp, N), N > 1, !.
mult_occ(Exp,Term)       :- mult_Occ(Term,Exp).
                                % Above step to rationalise
                                % argument order in mult_occ calls.
%   test whether Exp is a least dominating expression of Term, i.e.
%   whether Exp contains at least two occurrences of Term directly.

:- public least_dom/2.
:-  mode  least_dom(+, +), least_dom(+, +, +, +).

least_dom(Term,Exp) :-
        functor(Exp,Op,_),
        commutative(Op),
        associative(Op),
        !,
        decomp(Exp,[Op|ArgList]),
        bag_mult_occ(Term,ArgList).

        bag_mult_occ(Term,[Arg|ArgList]) :-
                contains(Term,Arg),
    *           contains(Term,ArgList),
                !.
        bag_mult_occ(Term,[_|ArgList]) :- bag_mult_occ(Term,ArgList).

least_dom(Term, Exp) :-
        functor(Exp, _, N),
        least_dom(N, 0, Term, Exp).

        least_dom(N, 2, Term, Exp) :- !.
        least_dom(0, K, Term, Exp) :- !, fail.
        least_dom(N, K, Term, Exp) :-
                arg(N, Exp, Arg),
                contains(Term, Arg),
                M is N-1, L is K+1, !,
                least_dom(M, L, Term, Exp).
        least_dom(N, K, Term, Exp) :-
                M is N-1, !,
                least_dom(M, K, Term, Exp).


%   position(Term, Exp, Path) is true when Term occurs in Exp at the
%   position defined by Path.  It may be at other places too.

:- public position/3.
:-  mode  position(?, +, ?), position(+, ?, +, ?).

position(Term, Term, []).
position(Term, Exp, Path) :-
        (   var(Exp) ; atomic(Exp) ; number(Exp)    ), !, fail.
position(Term, Exp, Path) :-
        functor(Exp, _, N),
        position(N, Term, Exp, Path).

        position(0, Term, Exp, Path) :- !, fail.
        position(N, Term, Exp, [N|Path]) :-
                arg(N, Exp, Arg),
                position(Term, Arg, Path).
        position(N, Term, Exp, Path) :-
                M is N-1, !,
                position(M, Term, Exp, Path).


%   generate intermediate variables, or arbitray integer tokens.

:- public arbint/1,      identifier/1.
:-  mode  arbint(-),     identifier(-).

arbint(Var) :-
        gensym(n, Var),
        assert(integral(Var)),
        trace('\n\tLetting %t denote an arbitrary integer', [Var], 1), !.

identifier(Var) :-
        gensym(x, Var),
        assert(intermediate(Var)), !.

%   fix the variable to be isolated if it has not already been fixed.

:- public fixvar/2,      ok/1.
:-  mode  fixvar(+, ?), ok(+).
```

```
fixvar(Exp, Var) :-
        var(Var),                % why not drop this test?
        wordsin(Exp,Words),
        member(Var, Words),
        ok(Var),
        checkand(contains(Var), Exp), !.
fixvar(Exp, Var) :-
        nonvar(Var).

ok(Var) :-
        \+ call(const(Var)),
        (  call(sought(Var))
        ;  call(given(Var))
        ), !.


%   correspond(X, Xlist, Ylist, Y) is true when the position of X and Xlist
%   and the position of Y in Ylist (which is as long as Xlist) are the same.

:- public correspond/4.
:-  mode  correspond(?, +, +, ?).       %  the lists must be given

correspond(X, [X|_], [Y|_], Y) :- !.
correspond(X, [_|T], [_|U], Y) :-
        correspond(X, T, U, Y).


%   cond_print(Old,New) prints New unless it matches Old.

:- public cond_print/2.
:-  mode  cond_print(+, +).

cond_print(Old, New) :-
        call(match(Old, New)), !.
cond_print(Old, New) :-
        trace('\nTidying to %t\n', [New], 1).

%   apply a substitution, tidy the result, and print a message.

:- public subst_mess/3.
:-  mode  subst_mess(+, +, -).

subst_mess(Substitution, Old, New) :-
        subst(Substitution, Old, Mid),
        tidy(Mid, New),
        trace('Applying substitution %c\n   to    : %c\n   gives : %c\n',
                [Substitution, Old, New], 1), !.


%   Find the smallest (if C = <) or greatest (if C = >) term in a list of
%   terms, where comparison is by the size of a term.

:- public extreme_term/3.
:-  mode  extreme_term(+, +, -), extreme_term(+, +, +, +, -).
:-  mode  term_size(+, -), term_size(+, +, +, -).

extreme_term([Head|Tail], C, Term) :-
        term_size(Head, Size),
        extreme_term(Tail, Head, Size, C, Term).
        extreme_term([Head|Tail], Hold, Sold, C, Term) :-
                term_size(Head, Size),
                compare(C, Size, Sold), !,
                extreme_term(Tail, Head, Size, C, Term).
        extreme_term([Head|Tail], Hold, Sold, C, Term) :-
                extreme_term(Tail, Hold, Sold, C, Term).
        extreme_term([],          Term, _,    _, Term).

        term_size(Term, 1) :-
                (   var(Term) ; atomic(Term) ; number(Term)    ), !.
        term_size(Term, Size) :-
                functor(Term, _, N),
                term_size(N, Term, 1, Size).

                term_size(0, Exp, Ans, Ans) :- !.
                term_size(N, Exp, Acc, Ans) :-
                        arg(N, Exp, Arg),
                        term_size(Arg, Size),
                        Nxt is Acc+Size+1, M is N-1, !,
                        term_size(M, Exp, Nxt, Ans).


 % Flatten list
:- public flatten/2.

:- mode flatten(+,-),flatten(+,?,+).

flatten(X,Y) :- flatten(X,Y,[]),!.

flatten([],X,X).
flatten([H|T],L1,L3) :- flatten(H,L1,L2),
        flatten(T,L2,L3).
flatten(X,[X|Z],Z).

 % Delete all occurrences of X from list Y to get list Z
:- public delete/3.

:- mode delete(+,+,-).

delete(_,[],[]) :- !.
delete(H,[H|T],T1) :- delete(H,T,T1),!.
delete(H,[X|T],[X|T1]) :- delete(H,T,T1),!.

 % Remove false from a set of disjunctions, hack to replace bug in Tidy

:- public remove_false/2,  remove_dis_dups/2.

:- mode         remove_false(+,-), remove_dis_dups(+,-).

remove_false(Term,Ans) :- decomp(Term,[#|List]),
        delete(false,List,New),
        recomp(Ans,[#|New]),
        !.

remove_false(X,X) :- !.

 % Remove duplications in a disjunction

remove_dis_dups(A#B,X) :- !,
        ortodot(A#B,List),
        listtoset(List,List1),
        dottoor(List1,X).

remove_dis_dups(X,Y) :- tidy(X,Y).  % For cases that fall through
```

```
% WORDS                                        Updated: 21-Apr-81.

:- public wordsin/2, frequent_words/2.

:- mode
    wordsin(+, -),
    frequent_words(+, -),
        scan_term(+, ?, -),
            insert_word(?, +, -),
            scan_list(+, ?, -),
        tree_list(?, +, +, -),
        strip_num(+, -).

%   wordsin(Term, List)
% finds all the words (atom) which occur at least once in Term, and returns
% them in List.  Furthermore, the words are in descending order of frequency.
% E.g. wordsin(x*x+x*y+y^2+z^7, [x,y,z]).
% The order is supposed to be heuristic.

wordsin(Term, List) :-
        scan_term(Term, Some, Tree),
        tree_list(Tree, 1, [], Pairs),
        keysort(Pairs, Inorder),
        strip_num(Inorder, List).

%   frequent_words(Term, List)
% finds all the words (atoms) which occur more than once in Term, and returns
% them in List.  Furthermore, the words are in descending order of frequency.
% E.g. frequent_words(x*x+x*y+y^2+z^7, [x,y]).

frequent_words(Term, List) :-
        scan_term(Term, Some, Tree),
        tree_list(Tree, 2, [], Pairs),
        keysort(Pairs, Inorder),
        strip_num(Inorder, List).

        scan_term(Simp, Old_Tree, Old_Tree) :-
                var(Simp), !.
        scan_term(Simp, Old_Tree, Old_Tree) :-
                number(Simp), !.        %  was integer(Simp)
        scan_term(Atom, Old_Tree, New_Tree) :-
                atom(Atom), !,
                insert_word(Old_Tree, Atom, New_Tree).
        scan_term(List, Old_Tree, New_Tree) :-
                List = [_|_], !,
                scan_list(List, Old_Tree, New_Tree).
        scan_term(Term, Old_Tree, New_Tree) :-
                Term =.. [Functor|Args], !,
                scan_list(Args, Old_Tree, New_Tree).

                insert_word(t(C, W, L, R), W, t(D, W, L, R)) :- !,
                        (   var(C), D = 1
                        ;   integer(C), D is C+1
                        ),  !.
                insert_word(t(C, X, L, R), W, t(C, X, M, R)) :-
                        W @< X, !,
                        insert_word(L, W, M).
                insert_word(t(C, X, L, R), W, t(C, X, L, S)) :-
                        W @> X, !,
                        insert_word(R, W, S).
        scan_list([Head|Tail], Old_Tree, New_Tree) :-
                scan_term(Head, Old_Tree, Mid_Tree), !,
                scan_list(Tail, Mid_Tree, New_Tree).
        scan_list([],          Old_Tree, Old_Tree).

tree_list(Tree,          Thresh, Accum, Accum) :-
        var(Tree), !.
tree_list(t(N, X, L, R), Thresh, Accum, Answer) :-
        N < Thresh,
        tree_list(L, Thresh, Accum, Sofar), !,
        tree_list(R, Thresh, Sofar, Answer).
tree_list(t(C, W, L, R), Thresh, Accum, Answer) :-
        tree_list(L, Thresh, Accum, Sofar),
        Key is -C, !,
        tree_list(R, Thresh, [Key-W|Sofar], Answer).

strip_num([Key-Word|Rest], [Word|More]) :- !,
        strip_num(Rest, More).
strip_num([],              []).
```

```
/* GPORTR : First stab at a general all level portray handler.

                                                   Richard+Lawrence
                                                   Updated: 26 July 82

        This was Richard's code for his rational stuff.
        Eventually I must fix these problems by having the 'print'
        routine in the interpreter actually descend level by level
        taking operators into account and calling portray at each
        level to see whether the users wants to handle it.
        NB: this has now been done.  Why is gportr still around?

        The following magic numbers appear in put(N) calls:
        32 = space, 40 = "(", 41 = ")", 44 = ",", 91 = "[", 93 = "]".
        The magic number 1000 also appears; this is the priority of ','.

*/


/* EXPORT */

:- public
    portray/1.


/* MODES */

:- mode
    portray(?),
        prin(+, +),
            prin(+, +, +),
            prnf(+, +, +),
            prna(+, +, +),
            prnp(+, +, +, +),
            printail(+),
            oper(+, ?, ?),
                oper(+, +, ?, ?).



                           % Top level

portray(Term) :-
        prin(1000, Term).



                           % Print a term taking account of surrounding
                           %  operator priorities.

        prin(Prio, Term) :-
                (   var(Term)           %  _N style of variables
                |   atom(Term)          %  ordinary atoms
                |   Term = '$VAR'(N)    %  A1 style of variables from numbervars
                ),  !,
                writeq(Term).           %  quotes around e.g. 'foo baz'
        prin(Prio, Term) :- /*Q'*/
                portray_number(Term),   %  if a number
                !.
        /*  Other user-provided portrayal methods should be called here  */
```

```
prin(Prio, [Head|Tail]) :- !,   %  list
        put(91),                %  "["
        prin(1000, Head),
        printail(Tail).
prin(Prio, Term) :-             %  postfix operator
        functor(Term, Functor, 1),
        oper(Functor, Lp, 0), !,
        prnp(Prio, Lp, 0, 40),
        prna(Lp, Term, 1),
        prnf(Functor, 0, 1),
        prnp(Prio, Lp, 0, 41).
prin(Prio, Term) :-             %  prefix operator
        functor(Term, Functor, 1),
        oper(Functor, 0, Rp), !,
        prnp(Prio, 0, Rp, 40),
        prnf(Functor, 1, 0),
        prna(Rp, Term, 1),
        prnp(Prio, 0, Rp, 41).
prin(Prio, Term) :-             %  infix operator
        functor(Term, Functor, 2),
        oper(Functor, Lp, Rp),
        Lp > 0, Rp > 0, !,
        prnp(Prio, Lp, Rp, 40),
        prna(Lp, Term, 1),
        prnf(Functor, 0, 0),
        prna(Rp, Term, 2),
        prnp(Prio, Lp, Rp, 41).
prin(Prio, Term) :-
        functor(Term, Functor, N),
        writeq(Functor),
        prin(0, N, Term).


                                % print one argument of a term

        prna(Prio, Term, ArgNo) :-
                arg(ArgNo, Term, Arg),
                prin(Prio, Arg).

                                % print a functor with spaces

        prnf(',', _, _) :- !,
                write(', ').
        prnf(';', _, _) :- !,
                write('; ').
        prnf(Functor, L, R) :-
                prnp(L, 1, 1, 32),
                write(Functor),
                prnp(R, 1, 1, 32).

                                % print the arguments of a term

        prin(0, N, Term) :-
                put(40),                %  "("
                prna(1000, Term, 1),
                prin(1, N, Term).
        prin(N, N, Term) :- !,
                put(41).                %  ")"
        prin(L, N, Term) :-
                M is L+1,
                        write(', '),
                        prna(1000, Term, M), !,
                        prin(M, N, Term).


                                        % Print a parenthesis if the priorities
                                        %  around the operator require it.

                prnp(Prio, Lp, Rp, Char) :-
                        Prio >= Lp, Prio >= Rp, !.
                prnp(Prio, Lp, Rp, Char) :-
                        put(Char).


                                        % Print the tail of a list, being
                                        %  careful about partial instantiation
                                        %  at the end.

                printail(List) :-
                        nonvar(List), List = [Head|Tail], !,
                        write(', '),
                        prin(1000, Head), !,
                        printail(Tail).
                printail(Tail) :-
                        Tail \== [],
                        put(124),                %  "|"
                        prin(1000, Tail), !,
                        printail([]).
                printail([]) :-
                        put(93).                 %  "]"



                        % Check for operators.  Return left and right
                        %  precedences. These are Richard's conventions.
                        %  Note that prefix/postfix ops have 0 for their
                        %  other precedence.

oper(Op, Left,Right) :-
        current_op(Prec, Type, Op),
        oper(Type, Prec, Left, Right).


        oper( fx, Prec, 0, Prec).
        oper( fy, Prec, 0, Prec).
        oper(xf , Prec, Prec, 0).
        oper(yf , Prec, Prec, 0).
        oper(xfx, Prec, Prec, Prec).
        oper(xfy, Prec, Prec, More) :- More is Prec+1.
        oper(yfx, Prec, More, Prec) :- More is Prec+1.
```

```
/* HOMOG.MSC :

                                                Bernard Silver
                                                Updated: 8 August 82

*/

:- public
                absol/2,
                break/4,
                expcase1/5,
                expcase2/4,
                fact/2,
                form/3,
                form1/3,
                form2/3,
                form4/3,
                gcd1/2,
                gcd2/2,
                genpolcase/3,
                great_el/2,
                half_angle_check1/2,
                half_angle_check2/2,
                laura/4,
                laura1/3,
                least_el/2,
                lessone/1,
                logocc/4,
                make_subl/3,
                moreone/1,
                neg22/1,
                nocc/3,
                onetest/2,
                parse2/3,
                powered/3,
                reduced_term/3,
                report_subs/2,
                signed/2,
                subs1/3.

 % Various functions for recognizing certain forms

                        %  The exponential case with all offending terms
                        %  of the form a^f(x), a the same in all terms.
expcase1(A,B,X,A^Z,C) :- atom_num(A),match(Z,C*B+D),number(C),freeof(X,D),!.
expcase1(A,B,X,A^Z,1) :- atom_num(A),match(Z,B+C),freeof(X,C),!.
expcase1(A,B,X,A^Z,C) :- atom_num(A),match(Z,C*B),number(C),!.
expcase1(A,B,X,A^B,1).


                        %  The other exponential case
expcase2(B,A^Y,set(A,Z)) :- number(A),match(Y,Z*B+C),number(Z),freeof(B,C),!.
expcase2(B,A^Y,set(A,1)) :- number(A),match(Y,B+C),freeof(B,C),!.
expcase2(B,A^Y,set(A,Z)) :- number(A),match(Y,Z*B),number(Z),!.
expcase2(B,A^B,set(A,1)).

                        %  Check is the tan(half-angle) method can be used
half_angle_check1(M,M) :- !.
half_angle_check1(M,N) :- eval(2*M,N),!.
```

```
half_angle_check2(M,M) :- !.

genpolcase(X,X,1) :- !.
genpolcase(X,X^N,N) :- !.

                         %  Standard log case
laura(B,X,log(A,B),A) :- freeof(X,A),!.
laura(A,X,log(A,B),B) :- freeof(X,B),!.

                         %  Convert to log base 10 case
laura1(Unk,Term,log(A,Term)) :-
        number(A),
        contains(Unk,Term),
        !.

laura1(Unk,Term,log(Term,A)) :-
        number(A),
        contains(Unk,Term),
        !.

coeff_exp(L,M,N) :- set_members(L,L1,L2),
        gcd2(L1,M),
        gcd2(L2,N),
        !.

set_members([],[],[]) :- !.
set_members([set(A,B)|T],[A|X],[B|Y]) :- set_members(T,X,Y),!.

onetest(K,A) :- checklist(moreone,K),least_el(K,A),!.
onetest(K,A) :- checklist(lessone,K),great_el(K,A),!.
onetest(K,A) :- listtoset(K,[A]),!.

logocc(A,B,log(A,B),L) :- member(log(A,B),L),!.
logocc(A,B,log(B,A),L) :- member(log(B,A),L),!.

 % These form functions put terms together prettily,so 1*A is A for example

form(Unk,K,Z) :-gcd2(K,Gcd),absol(Gcd,Gcd1),!,form1(Unk,Gcd1,Z).

form1(Unk,A,Res) :- tidy(A*Unk,Res),!.

form2(M,Rest,Res) :- eval(M/2,N),tidy(Rest*N,Res),!.

form4(_,0,1) :- !.
form4(A,1,A) :- !.
form4(A,N,A^N) :- !.

 % This recognizes numeric expressions eg 3^(1/2),on which number fails
numeric(X) :- wordsin(X,L),L=[],!.

atom_num(X) :- atomic(X),!.
atom_num(X) :- numeric(X),!.

 % Parser for trig method
parse2(A&B,X,L) :- parse2(A,X,L1),parse2(B,X,L2),union(L1,L2,L),!.
parse2(A=B,X,L) :- parse2(A,X,L1),parse2(B,X,L2),union(L1,L2,L),!.
parse2(A*B,X,L) :- parse2(A,X,L1),parse2(B,X,L2),union(L1,L2,L),!.
parse2(A+B,X,L) :- parse2(A,X,L1),parse2(B,X,L2),union(L1,L2,L),!.
parse2(A^N,_,[A^N]) :- integer(N),(trigf(A);hyperf(A)),!.
```

```
parse2(A^N,X,L) :- number(N),parse2(A,X,L),!.
parse2(A,X,[]) :- freeof(X,A),!.
parse2(A,X,[A]) :- !.

 % Find the "smallest" term in the  offenders set

reduced_term([Unk],Unk,_) :- !,fail.            %Unk can't be the reduced term
reduced_term([A],Unk,A) :- !.
reduced_term(L,Unk,A) :- extreme_term(L, <, A),   %  return the smallest
        !,
        A \= Unk.

 % Make a list of the rewrites found,and substitute them into
 % the expression
subs1(Exp,[],Exp) :- !.
subs1(Exp,[H|T],E1) :- subst(H,Exp,E2),subs1(E2,T,E1),!.

make_subl([],[],[]) :- !.
make_subl([X|R],[X|R1],R2) :- !,make_subl(R,R1,R2).
make_subl([Hd|R],[H1|R1],[Hd=H1|R2]) :- make_subl(R,R1,R2),!.

 % List the rewrites used, if desired

report_subs(X,List) :- report,
        !,
        sublist(contains(X),List,New),
        trace('\nRewrites used are:\n',1),
        report_subs1(New).

report_subs(_,_) :- !.

report_subs1([]) :- !.
report_subs1([L=R|T]) :- trace('\n %t -> %t\n',[L,R],1),report_subs1(T),!.

 % Turn on the reporting
report_on :- report,trace('\nReporting is already on! Nothing done\n',1),!.
report_on :- asserta((report :- !)),trace('\nReporting turned on\n',1),!.

 % Turn off reporting
report_off :- report,retract((report :- !)),trace('\nReporting turned off\n',1),!
report_off :- trace('\nReporting is not on! Nothing done\n',1),!.

report :- fail.

 % Find the smallest and largest elements of a list of numbers
least_el([Hd],Hd) :- !.
least_el([Hd|Tl],Ans) :- least_el(Tl,Lwr),(eval(Hd < Lwr) -> Hd=Ans;Lwr=Ans),!.

great_el([Hd],Hd) :- !.
great_el([Hd|Tl],Ans) :- great_el(Tl,Hgr),(eval(Hd>Hgr) -> Hd=Ans;Hgr=Ans),!.

 % powered(A,B,C) if A^B=C,A not equal 1
powered(1,_,_) :- !,fail.
powered(A,1,A) :- !.
powered(A,N,A^N) :- number(N),!.
powered(A,B,C) :- number(A),number(C),eval(log(A,C),X),!,number(X),B=X.

nocc(Eqn,A,N) :- occ(A,Eqn,N),!.

lessone(A) :- number(A),eval(A < 1),!.
```

```
moreone(A) :- number(A),eval(A > 1),!.

 % Absolute value
absol(X,X1) :- eval(sign(X)*X,X1),!.

 % Given terms A and B break(A,B,I,J) finds I and J
 % so that A=I*Y,and B=J*Y,if this is possible
break(A,B,1,1) :- match(A,B),!.
break(A,B,1,C) :- number(A),number(B),eval(B/A,C),!.
break(A,B,1,J1) :- match(A,I*Y),number(I),match(B,Y*J),number(J),eval(J/I,J1),!.
break(A,B,1,J) :- match(B,J*A),number(J),!.
break(A,B,J,1) :- match(A,J*B),number(J),!.

 % Factorial function
fact(0,1) :- !.
fact(N,M) :- eval(N>0),eval(N-1,N1),fact(N1,M1),eval(M1*N,M),!.

 % Find the least common multiple of a set of integers
lcm([A],A) :- !.
lcm([A,B|T],X) :- gcd(A,B,Z),eval((A*B)/Z,Y),lcm([Y|T],X),!.


 % Find the greatest common divisor of a list of integers
gcd1([A],A) :- !.
gcd1([H|T],X) :- gcd1(T,Y),gcd(H,Y,X),!.

 % Find the greatest common divisor of a list of rationals
gcd2(L,X) :- listtoset(L,L1),gcd3(L1,X),!.

gcd3([A],A) :- !.
gcd3([H|T],Y) :- gcd3(T,X),
          eval(numer(H),H2),
          eval(denom(H),H1),
          gcd_calc(H2,H1,X,Y),
          !.

gcd_calc(A,B,C,C) :- eval(A/B,C),!.
gcd_calc(A,B,X2,X3) :-eval(numer(X2),C),
          eval(denom(X2),D),
           lcm([B,D],Z),
          eval((Z/B)*A,Z1),
          eval((Z/D)*C,Z2),
          gcd(Z1,Z2,Y),
          eval(Y/Z,X3),
          !.
 % If all numbers on a list are negative then signed(list,-1),
 % else signed(list,1)
signed(L,-1) :- checklist(neg22,L),!.
signed(_,1) :- !.

neg22(Num) :- eval(sign(Num)=:= (-1)),!.
```

```
%   Arith:Odds.                                 Updated: 12 Sept 81
%   odd and even natural numbers, and           Author:  Alan Bundy
%   gcd calculations.  Now just an interface to Longs.

:- public natnum/1.     :- mode natnum(+).
:- public odd/1.        :- mode odd(+).
:- public even/1.       :- mode even(+).
:- public gcd/3.        :- mode gcd(+, +, -).
:- public oddnum/1.     :- mode oddnum(+).

natnum(X) :-
        integer(X), X > 0.

odd(X) :-
        eval(odd(X)).

even(X) :-
        eval(even(X)).

gcd(X, Y, Z) :-
        eval(gcd(X,Y), Z).

oddnum(X) :-
        1 is X mod 2.
```

```
/*      RUNEX
        Commands to run test examples.         Updated: 20 April 82
        The examples are found in the
        files testex.prb, mecho.prb, lewis.prb          Leon
        and exam in the area extras.                    */


run :- (present(testex) ; ['extras:testex.prb']), !,
                checklist(stats, [logeqn(A1), expeqn(A2), trigeqn(A3),
                        negpolyeqn(B1),homogeqn(B2),chunkeqn(B3),
                        invlogeqn(C1),nastyeqn(C2),cosapeqn(C3),acbseqn(C4),
                        taklogeqn(C5),
                        coseqn(D1),sqrteqn(D2),pow2eqn(D3),quarteqn(D4)]).

smallrun :- (present(testex) ; ['extras:testex.prb']), !,
                checklist(stats,[logeqn(A1), expeqn(A2), trigeqn(A3)]).

mechorun :- (present(mecho), present(init) ;
                ['extras:init.mec','extras:mecho.prb']), !,
                checklist(stats, [simppull(A1), n14(A2), car(A3),
                        pulltab(A4), tower1(A5), stvineq(A6), conjineq(A7),
                        dome(A8), bloc(A9), train(A10), loop(A11)]).

lewisrun :- (present(lewis) ; ['extras:lewis.prb']), !,
                checklist(stats, [a1(X1), b1(X2), a2(X3), b2(X4), a3(X5),
                        b3(X6), a4(X7), b4(X8), a5(X9), b5(X10), a6(X11),
                        b6(X12), a7(X13), b7(X14), a1hard(X15), a2hard(X16),
                        b1hard(X17), b2hard(X18), c1hard(X19), c2hard(X20),
                        d1hard(X21), d2hard(X22)]).

aebrun :- examcheck,aebrunsol,!.

lonrun :- examcheck,lonrunsol,!.

dlonrun :- examcheck,dlonrunsol,!.

oxfrun :- examcheck,oxfrunsol,!.

highrun :- examcheck,highrunsol,!.

eurocamrun :- examcheck,eurorun,!.

exam :- present(exam),!,
        writef('\nextras:exam is already loaded, nothing done\n').
exam  :- writef('\n[Consulting extras:exam]\n'),consult('extras:exam'),!.

examcheck :- present(exam),!.
examcheck :- writef('\n[Consulting extras:exam]\n'),consult('extras:exam'),!.

/*Run problem with statistics*/

stats(Problem) :- Problem=..[Name,Arg], statistics(runtime,_),
                call(Problem), !, statistics(runtime,[ _, Time]),
                trace('\n%t took %t milliseconds and produced answer %e\n\n',
                        [Name,Time,Arg], 0).

stats(Problem) :- statistics(runtime,[ _, Time]),
  trace('\nSorry I could not prove %t and I spent %t not doing it \n\n',
        [Problem, Time], 0).
```

```
/* TEST. :

                                                    Bernard Silver
                                                    Updated: 30 June 82
*/
        %  Problems for demonstration of Press

text(1) :- writef('\nThis problem comes from the London 1978 A level exam.\n
We are asked to find the value(s) of for which
        log(2,x) + 4.log(x,2) = 5.\n\n').

text(2) :-
        writef('\nThis problem is from the A.E.B. A level exam of 1971.\n
We are required to find the value(s) of x such that
        cos(x) + 2.cos(2.x) + cos(3.x) = 0.\n\n').

text(3) :-
        writef('\nThis problem is from the A.E.B. 1971 A level paper.\n
The question asks for the value(s) of x which satisfy
        4^x - 2^(x+1) - 3 = 0.\n\n').

text(4) :-
        writef('\nThis question demonstrates the basic methods of PRESS.\n
The problem is to find the value(s) of x that satisfy
                log(e,x+1) + log(e,x-1) = 3.\n\n').

basic :- example4.

example1 :- text(1),demo1,ttynl.

example2 :- text(2),demo2,ttynl.

example3 :- text(3),demo3,ttynl.

example4 :- text(4),demo4,ttynl.

        % The questions

demo1 :- solve(log(2,x) + 4*log(x,2) = 5).             %lon(15)

demo2 :- solve(cos(x) + 2*cos(2*x) + cos(3*x) = 0).    %aeb(7)

demo3:- solve(4^x - 2^(x+1) - 3 = 0).                  %aeb(6)

demo4 :- solve(log(e,x+1) + log(e,x-1) = 3). %logeqn
```

```
/* SCORE. :

                                                  Bernard Silver
                                                  Updated: 3 July 82

*/

PRESS solves the following proportion of problems:

                          A LEVEL
                          =======
            Single equations            Sim. eqn.                Total
A.E.B.      23 out of 28                 6 out of 8              29 out of 36

London      32 out of 35                 2 out of 3              34 out of 38


London D.    9 out of 10                   N/A                    9 out of 10

Oxford       2 out of 2                  0 out of 1               2 out of 3
------------------------------------------------------------------------------
TOTAL       66 out of 75                8 out of 12             74 out of 87


                          O LEVEL
                          -------
            Single equations            Sim. eqn.                Total
Oxford       7 out of 8                  2 out of 2               9 out of 10


                          SCOTTISH HIGHER
                          ---------------
            Single equations            Sim. eqn.                Total
             5 out of 5                  3 out of 3               8 out of 8

                          ALL PROBLEMS
                          ------------
            Single equations            Sim. eqn.                Total

            78 out of 88                13 out of 17             91 out of 105

              88.5%                     76.5%                    86.67%
```

```
/*GOALS
A Selection of Algebra Problems
Alan Bundy   10.5.79
Updated and modified by Leon Sterling 24.2.81
Changed 14.4.81      */

/*TOP LEVEL RUN*/

smallrun :- checklist(stats,[logeqn(A1), expeqn(A2), trigeqn(A3)]).

run :- checklist(stats, [logeqn(A1), expeqn(A2), trigeqn(A3),
                         negpolyeqn(B1), aeb4(B2), homogeqn(B3),
                         chunkeqn(A4), %  lon10(B5),
                         simppull(A5), nl4(A6), car(A7), simpeqns(A8),
                         pulltab(A9), tower1(A10),
                         stvineq(A11), conjineq(A12),
                         dome(A13), bloc(A14), train(A15), nastyeqn(A16),
                         loop(A17)]).

tmprun :- checklist(stats, [logeqn(A1), expeqn(A2), trigeqn(A3),
                         negpolyeqn(B1),homogeqn(B2),chunkeqn(B3),
                         simpeqns(C1),simppull(C2),nl4(C3),pulltab(C4),
                         stvineq(D1),train(D2),
                         pow2eqn(E1),quarteqn(E2)]).



/*Run problem with statistics*/

stats(Problem) :- Problem=..[Name,Args], statistics(runtime,_),
                  call(Problem), !, statistics(runtime,[ _, Time]),
                  trace('\n%t took %t milliseconds and produced answer %e\n\n',
                         [Name,Time,Args], 0).

stats(Problem) :- statistics(runtime,[ _, Time]),
  trace('\nSorry I could not prove %t and I spent %t not doing it \n\n',
        [Problem, Time], 0).


/*SINGLE EQUATIONS*/

logeqn(Ans) :- solve(log(e,x+1) + log(e,x-1) = 3, x, Ans).

expeqn(Ans) :- solve((2^(x^2))^(x^3) = 2, x, Ans).

trigeqn(Ans) :-
  solve(((2^(cos(x)^2)*2^(sin(x)^2))^sin(x))^cos(x) = 2^(1/4), x, Ans).

negpolyeqn(Ans) :- solve(1/x^2 = 1/x, x, Ans).

homogeqn(Ans) :- solve(a^(x+1) + a^(2*x) = c, x, Ans).

chunkeqn(Ans) :- solve(cos(x)^2 + b*cos(x) = c, x, Ans).

nastyeqn(Ans) :- solve(y=((1+x^2)^(2^(-1))) / x, x, Ans).


/*single equation goals*/

coseqn(Ans) :- solve(cos(x-45) = sin(2*x) , x , Ans).
```

```
sqrteqn(Ans) :- solve(sqrt(5*x - 25) - sqrt(x-1) = 2 , x , Ans).

pow2eqn(Ans) :- solve(2^(2*x+8) - 32*2^x + 1 = 0, x , Ans).

quarteqn(Ans) :- solve(12*x^4 - 56*x^3 + 89*x^2 - 56*x + 12 = 0, x ,Ans).


/*SIMULTANEOUS EQUATIONS*/

/*trivial test equations*/
simpeqns(Ans) :- simsolve(
    a=b & b=c & c=1 & true , [a,c,b] , Ans).

/*simple pulley*/
simppull(Ans) :- simsolve(
    m1*g*cos(180) + (1*tsn + 0) = m1*(a1*1)  &
    m2*g*1 + (cos(180)*tsn + 0) + 0 = m2*(a1*1)  &
    true , [tsn,a1] , Ans).

/*pulley and table with friction*/
pulltab(Ans) :- simsolve(
    m1*g*cos(270) + (1*tsn + (cos(-270)*reaction1 + 1*mu*reaction1 + 0))
       + 0 = m1*(a1*1)  &
    m2*g*1 + (cos(180)*tsn + 0 ) + 0=m2*(a1*1)  &
    m1*g*1 + (cos(270)*tsn + (reaction1 + cos(270)*mu*reaction1 + 0)) + 0
       = m1*(a1*cos(270))  &
    true,
  [reaction1, tsn, a1]  , Ans).

/*natural language problem four*/
nl4(Ans) :- simsolve(
    v^2=0^2 + 5*(60*60)^2 / (1760*3)*2000/1760  &
    true ,  [v] , Ans).

/*simple car problem*/
car(Ans) :- simsolve(
    1760*3*d0=0*60*t + 1/2*a*60*t^2  &
    v = 0 + a*60*t  &
    true ,  [t, v]  , Ans).

/*tower p21 no13 Palmer & Snell*/
tower1(Ans) :- simsolve(
    v = vel1 + 32*t2  &
    d2 = vel1*t2 + 1/2*32*t2^2  &
    true ,  [vel1, v]  , Ans).

/*train problem p18 Palmer & Snell*/
train(Ans) :- simsolve(
    t0 = t1+(t2+(t3+0))  &
    45/60 = 0 + 2^(-1)/60^2*t1  &
    45/60*t2 = d2  &
    0 = 45/60 + (-2)/60^2*t3  &
    7 = d1+(d2+(d3+0))  &
    d1 = 0*t1 + 1/2*2^(-1)/60^2*t1^2  &
    d3 = 45/60*t3 + 1/2*(-2)/60^2*t3^2  &
    true  ,  [t0, t1, t2, t3, d2, d1, d3]  , Ans).

/*tower to determine value of g*/
tower2(Ans) :- simsolve(
    v = 0 + a0*t0  &
    vc = 0 + a1*t1 &
    v = vc +a1*t2  &
    t0 = t1 + (t2 + 0)  &
    d2 = vc*t2 + 1/2*a1*t2^2  &
    d1 = 0*t1 + 1/2*a1*t1^2  &
    true  ,  [v, vc, a1, t0, t1, a0]  , Ans).

/*INEQUALITIES*/

stvineq(Ans) :- solveineq(x > 1/(1+sin(y)^2), x, Ans).

conjineq(Ans) :- solveineq(2*g*h1>0 & 2*g*(h1-h2)>=0 &
                       2*g*(h1-h2)>0 & 2*g*(h1-h2)>=0 & true, X, Ans).


%   Press cannot solve problems involving real(E^K) as it once used to,
%   to here are temporary formulations which avoid that pattern.
%   bloc fails because fixvar can't find a suitable variable, while
%   loop gets almost to the end and can't find the maximum.

bloc(Val) :- (X=h1;X=h2), solveineq(
        sqrt(2*g*h1) > 0 &
        2*g*(h1-h2) >= 0 &
        sqrt(2*g*(h1-h2)) > 0 &
        2*g*(h1-h2-h3*tan(t)) >= 0 & true, X, Val).
%bloc(Val) :- solveineq(sqrt(2*g*h1)>0 & real(sqrt(2*g*(h1-h2))) &
%  sqrt(2*g*(h1-h2))>0 & real(sqrt(2*g*(h1-h2-l1*tan(t)))) & true,
%  X,Val).

loop(Val) :- min(
        2*g*h-2*g*r >= 0 &
        sqrt(2*g*h-2*g*r) > 0 &
        2*g*h-4*g*r >= 0 &
        2*g*h - 2*g*r*(1+sin(ang)) >= r*g*sin(ang) &
        sqrt(2*g*h - 4*g*r) > 0 &
        2*g*h - 2*g*r*(1+sin(ang)) >= r*g*sin(ang) & true, h, Val).

%loop(Minval) :- min(real(sqrt(2*g*h-2*g*r)) & sqrt(2*g*h-2*g*r)>0 &
%  real(sqrt(2*g*h-4*g*r)) & (2*g*h-2*g*r*(1+sin(ang))) >= r*g*sin(ang) &
%  sqrt(2*g*h-4*g*r)>0 &
%  (2*g*h-2*g*r*(1+sin(ang))) >= r*g*sin(ang) & true,  h,Minval).

dome(Minval) :- min(m*g*(3*sin(d)-2)>=0 & true,d,Minval).


/*CURRENT PROBLEMS*/

pb1(A) :- eval(arcsin(2^(-1)), A).

pb2 :- non_zero(0*2^(-1)).

pb4(Ans) :- solve( vc= (-(-t2)^(-1))*t1*(a0*(t1+t2)+(-vc))*t2^2,vc,Ans).

pb3( (-t2)^(-1)*(-(1+t1*(-(-t2)^(-1)))^(-1))*t1*a0*(t1+t2)).

pb5((r*g*sin(ang)+(-1)*((-1)*(g*2*r*(1+sin(ang)))))*(2^(-1)*g^(-1)) ).
```

```
/* some O level problems */

sim1 :- simsolve( y-2*x=0 & 3*x^2+x*y+y^2=144 , [x,y] , Ans).

sim2 :- simsolve( x+y=101 & x-y=1 , [x,y] , Ans).
```

```
/* EXAM. :

                                                  Bernard Silver
                                                  Updated: 3 July 82
*/
/*  A-Level questions gathered together by Bernard 21.4.81 */
:- assert((present(exam))).

                    /* AEB exam questions   */

/*  June 1971  Paper2*/                           %solved
aeb(1) :- solve(sec(2*x) + tan(2*x) = 3).
/* Show first that sec(2x)+tan(2x)=(1+tan(x))/(1-tan(x)) */

aeb(2) :- solve(3*sech(x)^2 + 4*tanh(x) + 1 = 0).        %solved

/*  Nov 1971 Paper1 */

aeb(3) :- simsolve(3*x^2 + 15*x*y - 56*y^2 + 56 = 0 &
         2*x^2 + 9*x*y - 33*y^2 + 28 = 0,[x,y],X).
/* Told to solve by eliminating the constant terms  */

aeb(4) :- solve(1 - 3*cos(x)^2 = 5*sin(x)).              %solved

aeb(5) :- solve(4^(2*x+1) * 5^(x-2) = 6^(1-x)).          %solved

aeb(6) :- solve(4^x - 2^(x+1) - 3 = 0).                  %solved

/* Paper2 */

aeb(7) :- solve(cos(x) + 2*cos(2*x) + cos(3*x) = 0).     %solved

/* Nov 1972 Paper1 */

aeb(8) :- simsolve(x^3 = 9*y & 4^(2*x) = 3^(x+y),[x,y],X).       %solved
/* Find the non-zero values of x & y */

aeb(9) :- solve(2*sin(x) + cos(x) = 1).                  %solved

aeb(10) :- solve(2*sin(x) + cos(2*x) = 1).               %solved

/* Paper2 */

aeb(11) :- solve(25*cos(x)^2 - 4*sin(x)^2 - 20*cos(x) - 8*sin(x) = 0).
/* First show that left-hand side can be expressed as the difference of
two squares  */

/* June 1973 Paper1 */

aeb(12) :- solve(9^(3*x^2) = 27^(15-x)).                 %solved

aeb(13) :- solve(log(e,2*x-5) + log(e,x-3) = 2*log(e,2*x-1) - log(e,2)). %solved

aeb(14) :- solve(cos(6*x) + sin(6*x) + cos(4*x) + sin(4*x) = 0). %solved

aeb(15) :- solve(cos(2*x) + 3*sin(x) + 1 = 0).           %solved

/* Paper2 */

aeb(16) :- solve((cot(2*x) + cosec(2*x))^2 = sec(2*x)).
```

```
/* Show first that (cot(x) + cosec(x))^2 = (1 + cos(x))/(1 - cos(x)) */

aeb(17)  :- solve(sin(2*x) + sin(3*x) + sin(5*x) = 0).

aeb(18) :- sim(cosh(x) - 3*sinh(y) = 0 &
        2*sinh(x) + 6*cosh(y) = 5,[x,y],X). %solved with sim

/* June 1974 Paper1 */

aeb(19) :- solve(3*sin(x) + 4*cos(x) = 1).              %solved

/* Nov 1974 Paper1 */

aeb(20) :- solve(sin(150-x) = 2*sin(x-30)).             %solved


aeb(21) :- solve(5*cos(2*x) - 2*sin(2*x) =  2).         %solved

/* 20 & 21  make one complete question  */

/*      June 1975 Paper1 */

aeb(22)  :- simsolve(log(16,x*y) = 7/2 &
        log(4,x)*log(4,y) = -8,[x,y],X).
/* Show first that log(16,x*y) = 1/2*log(4,x) + 1/2*log(4,y) */

aeb(23) :- solve(3*cos(x)^2 + 5*sin(x) - 1 = 0).                %solved

/*      Paper2  */

aeb(24) :- solve((1-tan(x))*(1+sin(2*x)) = 1 + tan(x)).         %solved

aeb(25) :- sim(2*cosh(y) - 7*sinh(x) = 3 &
        cosh(y) - 3*sinh(x)^2 = 2,[x,y],X). %solved with sim

/*      Nov 1975 Paper1 */

aeb(26) :- solve(log(x,8) + log(8,x) = 13/6).           %solved

/* Paper2 */

aeb(27) :- solve(sin(x) - sin(4*x) + sin(7*x) = 0). %solved

aeb(28) :-  solve(sin(3*x) = 2*cos(2*x)).
/* Verify that x=30 is a solution.Find general solution */

/* June 1976 Paper1 */

aeb(29) :-  simsolve(log(y,x) = 2 &
         log(2,x) + log(2,y) = 3,[x,y],X).  %solved

aeb(30) :-  solve(7*sin(x) - 24*cos(x) = 15).           %solved

aeb(31) :-  sim(cos(x) + cos(y) = 1 &
        sec(x) + sec(y) = 4,[x,y],X). %solved with sim

/* Paper2 */

aeb(32) :-  solve(cos(x) + cos(3*x) + cos(5*x) = 0).            %solved
```

```
/* Nov 1976 Paper1  */

aeb(33) :-  simsolve(a*log(4,128) - b*log(8,2) = 6     &
         log(2,a) + (1/3)*log(2,b^3) = 2*log(4,6),[a,b],X). %solved

aeb(34) :-  solve(2*sec(x) + 3*sin(x) = 4*cos(x)).       %solved

/* Paper2 */

aeb(35) :-  solve(x^3 - 9*x + 4 = 0).
/* aeb(35) gives substitution x = 2*3^(1/2)*cos(y)  */

/* June 1978 S paper */

aeb(36) :-  solve(cos(5*x) = cos(2*x)). %solved

                /* London questions   */

/* Jan 1976 Paper1  */

lon(1) :-  solve(10^(x - 3) = 2^(10 + x)).               %solved

lon(2) :-  solve(cot(2*x) = 2 + cot(x)).                 %solved

lon(3) :-  solve(cos(3*x) - 3*cos(x) = cos(2*x) + 1).            %solved

/* Paper3 */

lon(4) :-  solve(9*cosh(x) - 6*sinh(x) = 7).             %solved

/* June 1976 Paper1 */

lon(5) :-  solve(sin(x) + sin(2*x) = sin(3*x)).          %solved

lon(6) :-  solve(2*tan(x) + sec(2*x) = 2*tan(2*x)).
/* Whole question */


lon(7) :-  solve(log(x,45) + 4*log(x,2) - (1/2)*log(x,81) - log(x,10) = 3/2).
                %solved
/* Jan 1977 Paper1  */

lon(8) :-  solve(2^(2*x) - 5*2^(x + 1) + 16 = 0).                %solved

lon(9) :-  solve(8*cos(x) - 15*sin(x) = 3).              %solved

/* June 1977 Paper1  */

lon(10) :-  solve(e^(3*x) - 2*e^x - 3*e^( - x) = 0).             %solved

lon(11) :-  solve(2*sin(x) + cos(x) + 2 = 0).            %solved
/* lon(11) asks for tan(x/2) substitution   */

/* Paper2 */

lon(12)  :-  solve(7*sin(x)^2 - 5*sin(x) + cos(x)^2 = 0).                %solved

lon(13) :-  solve(8*sin(x) + 15*cos(x) = 17/2).          %solved

/* Special Paper  */
```

```
lon(14) :-  solve(x^4 - 7*x^3 + 14*x^2 - 7*x + 1 = 0).          %solved

/* Jan 1978 Paper1  */

lon(15) :-  solve(log(2,x) + 4*log(x,2) = 5).          %solved
/* Whole question */

/* Paper2 */

lon(16) :-  simsolve(2*x + 6*y + z = 0 &
                 ( - 1)*x + 2*y - z = 10 &
                 4*x + 3*y + z = 1,[x,y,z],X).          %solved

/* June 1978 Paper2 */

lon(17) :-  solve(5*cosh(x) - 3*sinh(x) = 5).          %solved

/* Jan 1979 Paper1 */

lon(18) :-  solve(3*cot(2*x) + 7*tan(x) = 5*cosec(2*x)).

/* Paper2 */

lon(19) :-  solve(sin(2*x) = sin(x)).          %solved
/* Whole question */

/* June 1979 Paper1 */

lon(20) :-  solve(sin(x) - 7*cos(x) + 5 = 0).          %solved
/* lon(20) suggests tan(x/2) method */

/* Paper2 */

lon(21) :-  solve(cos(3*x) + sin(3*x) = 1).          %solved

/* June 1980 Paper1 */

lon(22) :- solve(sin(3*x) = sin(x)^2). %solved
/* First expand sin(3*x) in the normal way  */

lon(23) :- solve(4*cos(x) + sin(x) = 1).  %solved
/* Must use  tan(x/2) method  */

/* Paper2 */

lon(24) :- solve(4^(3+x)/8^(10*x) = 2^(10 - 2*x)/64^(3*x)). %solved

lon(25) :- solve(log(2,(x+4)) = 2 - log(2,x)). %solved

lon(26) :- solve(6^(1/2)*cos(x) - 2^(1/2)*sin(x) = 2). %solved
/* given that a.cos(x) - b.sin(x) = 2.cos(x+pi/6)  */

/* Special Paper */
lon(27) :- solve(2*cosh(2*x) + sinh(x) = 2). %solved

lon(28) :- sim(sinh(x)*cosh(y) = 3 & cosh(x)*sinh(y) = -1, [x,y],X).
        %solved with sim

/* Jan 1981 Paper 2 */
```

```
lon(29) :- solve(sin(x) = cos(a)).                %solved
/* Solve for x, x and a are both in degrees */

/* June 1981  Paper1 */

lon(30) :- solve(e^(log(e,x)) + log(e,e^x) = 8). %solved

/* Paper2 */

lon(31) :- solve(sin(2*x) + sin(x) = 0). %solved

lon(32) :- solve(2*cosh(x) - 2*sinh(x) = 3). %solved

lon(33) :- solve(2^(2/x) = 32). %solved

lon(34) :- solve(log(x,2)*log(x,3) = 5). %solved

lon(35) :- solve(x^3 - 2*x^2 - 4*x + 8 = 0). %solved
/* Theory of equations type question.
        First we must find a relation between b,c and d which
holds when the roots of x^3 + b.x^2 + c.x + d = 0 are in G.P.
        Then we solve the equation above and verify that the roots are in
        G.P.  Note we do not prove that this relation holds implies
roots in G.P. */

lon(36) :- solve(4*x^3 - 24*x^2 + 23*x + 18 = 0). %solved
/* Given that roots are in A.P.  */
/* Special Paper */

lon(37) :- solve(sin(8*x)^2 - sin(7*x)^2 = sin(x)^2).

lon(38) :- simsolve(tan(y) + 2*sec(y) = 2*x & x*cot(y) - 2*cosec(y) = 3,
                [x,y],X).


/*              Oxford Board        */

/* Additional Maths */

/* 1976 Paper2  */

oxf(1) :-  simsolve(3*x + y = 5 &
                x^2 + 2*y^2 - 3*x + 2*y + 2 = 0,[x,y],X).                    %solved

/* Summer 1977 Paper1 */

oxf(2) :-  solve(8*cos(x) - sin(x) = 4).                     %solved

/* Paper2 */

oxf(3) :-  simsolve(2*x + 3*y = 5 &
                x^2 + y^2 - 6*x + 4*y = 0,[x,y],X).                  %solved

/* Autumn 1977 Paper2 */

oxf(4) :-  solve(2*sin(x)^2 - 1 = (1 + cos(x))^2).                   %solved

oxf(5) :-  solve(sec(x) - 1/sec(x) = sin(x)).                %solved
```

```
/* Summer 1978 Paper1 */

oxf(6) :-  solve(3*sin(x)^2  - cos(x) - 1 = 0).         %solved

/* Paper2 */

oxf(7) :-  solve(4*cot(2*x)*cosec(2*x) + sec(x)^2*cosec(x)^2 = 8/3).
/* For oxf(7) the solver was asked to show first:
a) cosec(x)^2 + sec(x)^2 = sec(x)^2*cosec(x)^2,
b) cosec(x)^2 - sec(x)^2 = 4*cot(2*x)*cosec(2*x)     */

/* Autumn 1978 Paper1 */

oxf(8) :-  solve(3*tan(3*x) - tan(x) + 2 = 0).          %solved
/* Show first that tan(3*x) =  (3*tan(x) - tan(x)^3)/(1 - 3*tan(x)^2)  */

/* Summer 1979 Paper2 */

oxf(9) :-  solve(sin(3*x) = 2*sin(x)).          %solved

oxf(10) :-  solve(sin(3*x) = 4*sin(x)).                 %solved

/* A level */

/* Summer 1977 Paper1 */

oxf(11) :-  simsolve(2*x^2 - 3*x*y + 2*y^2 = 8 &
                4*x^2 - 5*x*y + 2*y^2 = 4,[x,y],X).

/* Summer 1979 Paper1 */

oxf(12) :-  solve(4*cos(x) + 3*sin(x) = 2).             %solved

/* Paper2 */

oxf(13) :-  solve(x^4 - 6*x^3 - 7*x^2 + 36*x + 36 = 0).         %solved

        /* London Syllabus D A level */

/* Jan 1978 Paper2 */

dlon(1) :- solve(150*cos(x) + 80*sin(x) = 51).          %solved

/* June 1978 Paper2 */
dlon(2) :- solve(2*e^x - 2*e^(- x) = 3).                %solved

dlon(3) :- solve(3*cos(x) + 2*sec(x) + 5 = 0).          %solved
/* Question asks for values of cos(x) and tan(x)^2,rather than x */

dlon(4) :- solve(sin(x) + 7*cos(x) = 5).                %solved
/* Questions 3 and 4 make one complete question */

/* Special Paper */

dlon(5) :- solve(4*tan(2*x) + 3*cot(x)*sec(x)^2 = 0).   %solved

/* Jan 1979 Paper2 */

dlon(6) :- solve(sin(2*x) = cos(x)).           %solved
```

```
/* June 1979 Paper2 */

dlon(7) :- solve(sin(5*x) + sin(3*x) = 0).                   %solved

dlon(8) :- solve(cos(x) + cos(x + a) + cos(x + 2*a) = 1 + 2*cos(a)).
/* Find the smallest positive x,given that 0 < a < 90 */

dlon(9) :- solve(sin(30 + x) = cos(45 + x)). %solved

/* Special Paper */

dlon(10) :- solve(x^3 - 3*x^2 - 3*x + 1 = 0).               %solved
/* First show tan(3x) = (3tan(x) - tan(x)^3)/(1 - 3tan(x)^2) and then deduce
that roots of above equation are tan(15),tan(75) and tan(135) */

        /* Scottish Higher Mathematics */

/* 1977 Paper2 */

high(1) :- solve(10*cos(x)^2 + sin(x) - 7 = 0).  %solved

high(2) :- simsolve(x + 3*y = 4 & x^2 + 3*x*y + 5*y^2 - 6*x = 0,[x,y],X).
        %solved

/* 1978 Paper2 */

high(3) :- simsolve(2*x - 3*y + 1 = 0 & 2*x^2 + 3*y^2 + 3*x + y = 4,[x,y],X).
        %solved

high(4) :- solve(sin(5*x) + sin(x) = 3*cos(2*x)).       %solved

/* 1979 Paper2 */

high(5) :- simsolve(4*x + y - z = 12 & 3*x - y + 3*z = 0
                & 5*x - 3*y + 2*z = -1,[x,y,z],X).      %solved

high(6) :- solve(2*cos(2*x) + cos(x) - 1 = 0).  %solved

/* 1981 Paper2 */

high(7) :- solve(sin(5*x/2) - sin(3*x/2) = 0).  %solved

high(8) :- solve(9*6^(2*x) - 10*6^x + 1 = 0).   %solved
/* To be solved by factorizing 9*a^(2*x) - 10*a^x + 1 and setting a to 6 */

        /* Timing clauses */

timeprob(Prob) :- statistics(runtime,_),
        call(Prob),!,statistics(runtime,[_,Time]),
        trace('\n%t took %t milliseconds\n',[Prob,Time],0).

timeprob(Prob) :- statistics(runtime,[_,Time]),
        trace('\nCould not solve problem %t,the attempt  took %t milliseconds
        \n',[Prob,Time],0).

        /*  Runs */

runaeball :- aebrecurse(1),!.

runlonall :- lonrecurse(1),!.
```

```
runoxfall :- oxfrecurse(1),!.

rundlonall :- dlonrecurse(1),!.

runhighall  :- highrecurse(1),!.

aebrecurse(37) :-  trace('\nAEB run complete\n',0),!.

aebrecurse(N) :- timeprob(aeb(N)),eval(N+1,M),aebrecurse(M),!.

lonrecurse(39) :-  trace('\nLondon run complete\n',0),!.

lonrecurse(N) :- timeprob(lon(N)),eval(N+1,M),lonrecurse(M),!.

oxfrecurse(14) :-  trace('\nOxford run complete\n',0),!.

oxfrecurse(N) :- timeprob(oxf(N)),eval(N+1,M),oxfrecurse(M),!.

dlonrecurse(11) :-  trace('\nLondon D run complete\n',0),!.

dlonrecurse(N) :- timeprob(dlon(N)),eval(N+1,M),dlonrecurse(M),!.

highrecurse(9) :-  trace('\nScottish Higher run complete\n',0),!.

highrecurse(N) :- timeprob(dlon(N)),eval(N+1,M),highrecurse(M),!.

aebrunsol :-  checklist(timeprob,[aeb(1),aeb(2),aeb(4),aeb(5),aeb(6),aeb(7),
        aeb(8),aeb(9),aeb(10),aeb(12),aeb(13),aeb(14),aeb(15),aeb(18),
        aeb(19),aeb(20),aeb(21),aeb(23),aeb(24),aeb(25),aeb(26),aeb(27),aeb(30),
        aeb(31),aeb(32),aeb(33),aeb(34),aeb(36)]).

lonrunsol :- checklist(timeprob,[lon(1),lon(2),lon(3),lon(4),
        lon(5),lon(7),lon(8),lon(9),lon(10),lon(11),lon(12),
        lon(13),lon(14),lon(15),lon(16),lon(17),lon(19),lon(20),
        lon(21),lon(22),lon(23),lon(24),lon(25),lon(26),lon(27),lon(28),lon(29),
        lon(30),lon(31),lon(32),lon(33),lon(34),lon(35),lon(36)]).

oxfrunsol :- checklist(timeprob,[oxf(1),oxf(2),oxf(3),oxf(4),
        oxf(5),oxf(6),oxf(8),oxf(9),oxf(10),oxf(12),oxf(13)]).

dlonrunsol :- checklist(timeprob,[dlon(1),dlon(2),dlon(3),dlon(4),dlon(5),
        dlon(6),dlon(7),dlon(9),dlon(10)]).

highrunsol :- checklist(timeprob,[high(1),high(2),high(3),high(4),high(5),
        high(6),high(7),high(8)]).

eurorun :- checklist(timeprob,[aeb(5),aeb(32),oxf(8),lon(15),aeb(2),
        solve(log(e,x+1) + log(e,x-1) =3),lon(10)]).
```

```
/*              FAILED

                                                 Bernard Silver
                                                 Updated: 23 March 82


*/
 % This is the complete set of questions that PRESS fails on, for Lawrence
 % Grouped into types.

 % First type, hints and lemmas.
aeb(3) :- simsolve(3*x^2 + 15*x*y - 56*y^2 + 56 = 0 &
        2*x^2 + 9*x*y - 33*y^2 + 28 = 0,[x,y],X).
 % Told to solve by eliminating the constant terms.  This gives x = 2.y or
 % x = -5.y and these values are substituted in.


aeb(28) :-  solve(sin(3*x) = 2*cos(2*x)).
 % Hint: 'Verify that x=30 is a solution.Find general solution '
 % The fact that 30 is a solution enables us to factorize cubic equation
 % that appears after homogenization.  We can't factorize this otherwise.

 aeb(35) :-  solve(x^3 - 9*x + 4 = 0).
 % The hint suggests the substitution x = 2*3^(1/2)*cos(y).
 % The student can then rediscover the cubic solution method.


aeb(11) :- solve(25*cos(x)^2 - 4*sin(x)^2 - 20*cos(x) - 8*sin(x) = 0).
 % The hint tell us to show that the left-hand side can be expressed
 % as the difference of two squares, the solution is then easy.

aeb(16) :- solve((cot(2*x) + cosec(2*x))^2 = sec(2*x)).
 % The hint is to show first that
 % (cot(x) + cosec(x))^2 = (1 + cos(x))/(1 - cos(x)).
 % Still lots of work to do after, i.e. change unknown, clear rationals
 % then solve the quadratic.


oxf(7) :-  solve(4*cot(2*x)*cosec(2*x) + sec(x)^2*cosec(x)^2 = 8/3).
  % For oxf(7) the solver was asked to show first:
 % a) cosec(x)^2 + sec(x)^2 = sec(x)^2*cosec(x)^2,
 % b) cosec(x)^2 - sec(x)^2 = 4*cot(2*x)*cosec(2*x)
 % The question then simplifies to 2*cosec(x)^2 = 8/3


aeb(22)  :- simsolve(log(16,x*y) = 7/2 &
        log(4,x)*log(4,y) = -8,[x,y],X).
 % The hint is to  show first that log(16,x*y) = 1/2*log(4,x) + 1/2*log(4,y).
 % It is then fairly easy, after change of unknown.


 % Modified A.P.
dlon(8) :- solve(cos(x) + cos(x + a) + cos(x + 2*a) = 1 + 2*cos(a)).
 % We are asked to find the smallest positive x,given that 0 < a < 90.
 % The left hand side is equal to cos(x+a)*(1 + 2*cos(a)) by the A.P. trick
 % Thus factorising gives (1 + 2*cos(a))*(cos(x+a) - 1) = 0.
 % As a lies between 0 and 90 the first factor cannot be 0, so cos(x+a) must = 1.
 % This means x+a = 0, or 360 or 720 etc.  We want the first positive value of
 % x, so the first choice is ruled out, so the answer is x = 360 - a, which is
 % positive, by the bounds on a.
```

```
aeb(17)  :- solve(sin(2*x) + sin(3*x) + sin(5*x) = 0).
 % This is a variant of the A.P. trig case, but is much harder.
 % (This is the one that I wrongly claimed to know how to solve easily)
 % Still add the first and last terms as in trigmethod, but now
 % we obtain 2*sin(7/2*x)*cos(3/2*x) + sin(3*x) = 0.
 % (In the A.P. case one of the terms in the product is the same as the
 % remaining term.)  Now note that a factor of 2*cos(3/2*x) can he removed.
 % This is non-trivial.   Any other combination of addition works as well,
 % clearly the fact that the RHS of the equation is 0 is crucial.

 %Super Homogenization
 % This next section suggest a new method.  Instead of using
 % homogenization to rewrite all terms as fuctions of one term, we need
 % an intermediate stage.  We rewrite all trig terms as functions of
 % cos and sin, and then "see what we can do".


lon(6) :-  solve(2*tan(x) + sec(2*x) = 2*tan(2*x)).
 % Rewrite everything in terms of sin(x) and cos(x), clear rationals
 % and most terms cancel.

lon(18) :-  solve(3*cot(2*x) + 7*tan(x) = 5*cosec(2*x)).
 % Similar to lon(6) but more work needed at the end.
 % See note on dlon(5)


lon(38) :- simsolve(tan(y) + 2*sec(y) = 2*x & x*cot(y) - 2*cosec(y) = 3,
                [x,y],X).

 % Other types

oxf(11) :-  simsolve(2*x^2 - 3*x*y + 2*y^2 = 8 &
                4*x^2 - 5*x*y + 2*y^2 = 4,[x,y],X).
  % Solved by subtracting one equation from the other to find a value for y in
  % terms of x, then substituting this value in.

lon(37) :- solve(sin(8*x)^2 - sin(7*x)^2 = sin(x)^2).
  % Best done using difference of two squares.
```

```
/* FIXED:

                                                    Bernard Silver
                                                    Updated: 30 June 82
*/

 % Stuff from failed file that is now fixed
 % This file gives the text from the failed file, and then tells how the
 % problem was fixed

/* Problem 1 */
lon(30) :- solve(e^(log(e,x)) + log(e,e^x) = 8).

 % Text from Failed
 /*
 This question will be solved once the correct tidy axioms have been added.
 PRESS does not know that e^log(e,x) = x, but it does know that
 log(e,e^x) = x    */

 % Solved by adding tidy axioms as above

/* Problem 2 */
lon(29) :- solve(sin(x) = cos(a)).

 % Text from Failed

/* Need to solve for x with x and a both in degrees.  This is solved quite
happily  by PRESS, using isolation, to obtain
        x = 180.n + arcsin(cos(a))*(-1)^n.
 The point is that arcsin(cos(a)) needs simplification.  This could be added
 as a tidy axiom.  Actually nastymethod does know this rule, but of course is
 never called. */

 % Solved as above

/* Problem 3 */
aeb(20) :- solve(sin(150-x) = 2*sin(x-30)).


 % Text from Failed

/* Homogenization should be able to solve this but PRESS gets overloaded.
 If we first expand in terms of both sin and cos, then collect, the
 problem may clear up.  Collect needs rewriting! */

 % Solved by improving Collection and Attraction

/* Problem 4 */
dlon(5) :- solve(4*tan(2*x) + 3*cot(x)*sec(x)^2 = 0).

 % Text from Failed

/* Similar to lon(18).  In fact both should really be solved by selecting
 tan(x) as the reduced term.  Homogenization will in fact do this if
 its first choice is failed.  This should happen if the problem is left
 running long enough, but no-one has had enough patience! */

 % Solved by improving Homogenization to choose tan as the reduced term
```

```
/*      TESTEX.PRB
A Selection of Algebra Problems
Alan Bundy   10.5.79
Updated and modified by Leon Sterling 24.2.81
Changed 14.4.81
Renamed and reorganised 23.9.81         */

:-      assert((present(testex))).


/*SINGLE EQUATIONS*/

logeqn(Ans) :- solve(log(e,x+1) + log(e,x-1) = 3, x, Ans).

expeqn(Ans) :- solve((2^(x^2))^(x^3) = 2, x, Ans).

trigeqn(Ans) :-
  solve(((2^(cos(x)^2)*2^(sin(x)^2))^sin(x))^cos(x) = 2^(1/4), x, Ans).

negpolyeqn(Ans) :- solve(1/x^2 = 1/x, x, Ans).

homogeqn(Ans) :- solve(a^(x+1) + a^(2*x) = c, x, Ans).

chunkeqn(Ans) :- solve(cos(x)^2 + b*cos(x) = c, x, Ans).

invlogeqn(Ans) :- solve(log(x,4) + log(4,x) = 5/2, x, Ans).

nastyeqn(Ans) :- solve(y= (  (1+x^2)^(1/2)  ) / x, x, Ans).

cosapeqn(Ans) :- solve(cos(x) + cos(3*x) + cos(5*x) = 0, x, Ans).

acbseqn(Ans)  :- solve(3*cos(x) + 4*sin(x) = 5, x, Ans).

taklogeqn(Ans) :- solve(10^(x+3) = 2^(10+x), x, Ans).

/*single equation goals*/

coseqn(Ans) :- solve(cos(x-45) = sin(2*x) , x , Ans).

sqrteqn(Ans) :- solve(sqrt(5*x - 25) - sqrt(x-1) = 2 , x , Ans).

pow2eqn(Ans) :- solve(2^(2*x+8) - 32*2^x + 1 = 0, x , Ans).

quarteqn(Ans) :- solve(12*x^4 - 56*x^3 + 89*x^2 - 56*x + 12 = 0, x ,Ans).


/*SIMULTANEOUS EQUATIONS*/

/*trivial test equations*/
simpeqns(Ans) :- simsolve(
    a=b & b=c & c=1 & true , [a,c,b] , Ans).

/* some O level problems */

sim1 :- simsolve( y-2*x=0 & 3*x^2+x*y+y^2=144 , [x,y] , Ans).

sim2 :- simsolve( x+y=101 & x-y=1 , [x,y] , Ans).


/*CURRENT PROBLEMS*/
```

```
pb1(A) :- eval(arcsin(2^(-1)), A).

pb2 :- non_zero(0*2^(-1)).

pb4(Ans) :- solve( vc= (-(-t2)^(-1))*t1*(a0*(t1+t2)+(-vc))*t2^2,vc,Ans).

pb3( (-t2)^(-1)*(-(1+t1*(-(-t2)^(-1)))^(-1))*t1*a0*(t1+t2)).

pb5((r*s*sin(ans)+(-1)*((-1)*(s*2*r*(1+sin(ans)))))*(2^(-1)*s^(-i)) ).
```

```
/*                  Exam questions    Inequalities          */
/* Collected by Bernard Silver 19.9.81 */

/* Numbers continue from exam file */

/* O level  Additional Maths Oxford Board */

/* Autumn 1976 Paper1 */

oxf(14) :- solveineq(x-(6/x) > 1,x,Ans).

/* Summer 1977 */

oxf(15) :- solveineq(x+7 =< 2*y & 2*y =< 2*x+4,x,Ans).
/* Prove x>= 3 and find similar inequality for y

/* A level */

/* Summer 1977 Paper1 */

oxf(16) :- solveineq(sin(2*x) > cos(x) & x>= 0 & 180 >= x,x,Ans).

/* Summer 1979 Special Paper */

oxf(17) :- solveineq((x+3)/((x-1)*(x-3)) < 1,x,Ans).

oxf(18) :- solveineq(cos(x)^3 > 3*sin(x)^2*cos(x) & x>= -180 & 180 >=x,x,Ans).
/* In modulus form */

/* London */

/* Jan 1977 Paper2 */

lon(22) :- solveineq((2-x-x^2)/x^2 > 0 & 2 > (2-x-x^2)/x^2,x,Ans).

/* June 1977 Paper2 */

lon(23) :- solveineq((x^2+1)/(2*(2*x-1)) > 0 & 2 > (x^2+1)/(2*(2*x-1)),x,Ans).

/* June 1978 Paper1 */

lon(24) :- solveineq(x/(x-2) > 1/(x+1),x,Ans).

/* Paper2 */

lon(25) :- solveineq(1 > 1/(1+cos(x)^2) & 90 > x & x > -90,x,Ans).
/* In modulus form */

lon(26) :- solveineq(17 >= (4*cos(x) + sin(x))^2,x,Ans).
/* Prove inequality holds for all x */

/* Jan 1979 Paper2 */

lon(27) :- solveineq(x^2 - 9 > x^2 - 23 # -(x^2-9) > x^2 - 23 ,x,Ans).
/* In modulus form */

/* June 1979 Paper1 */

lon(28) :- solveineq(4*x/(x+2) > 1,x,Ans).
lon(29) :- solveineq((x+2)/((x+1)*(x-2)) > 0,x,Ans).

/* Special Paper */

lon(30) :- solveineq((4*x^2 -24*x +35)/3 > 1-x &
        1-x > -1*(4*x^2 -24*x +35)/3 ,x,Ans).
/* In modulus form */
/* LEWIS  - Equations used by Clayton Lewis in investigating
                the psychology of equation solving and
                Skill in Algebra
                                        Gathered by Leon
                                        Updated: 15 July 81
*/

:-      assert((present(lewis))).

                            % 14 equations used as the basic test set
                            %  in 'Skill in Algebra'
                            %  IBM Research Report RC 8359 (#36359)

a1(Ans) :- solve(a=p+p*r*t,p,Ans).

b1(Ans) :- solve(2*x=x^2,x,Ans).


a2(Ans) :- solve(1/3=1/x+1/7,x,Ans).

b2(Ans) :- solve(1/r=1/x+1/y+1/z,x,Ans).


a3(Ans) :- solve(9*(x+40)=5*(x+40),x,Ans).

b3(Ans) :- solve(7*(4*x-1)=3*(4*x-1)+4,x,Ans).


a4(Ans) :- solve(x*y+y*z=2*y,x,Ans).

b4(Ans) :- solve((x+3+x)/x^2=1,x,Ans).


a5(Ans) :- solve(5/10=(x-10)/(x+5),x,Ans).

b5(Ans) :- solve((1-x^2)/(1-x)=2,x,Ans).


a6(Ans) :- solve(x+2*(x+1)=4,x,Ans).

b6(Ans) :- solve(x+2*(x+2*(x+2))=x+2,x,Ans).


a7(Ans) :- solve(x-2*(x+1)=14,x,Ans).

b7(Ans) :- solve(6*(x-2)-3*(4-2*x)=x-12,x,Ans).



                                    % 8 additional harder problems

a1hard(Ans) :- solve((1/x+1/x^2)/(1/x+2*x^2)=3,x,Ans).

a2hard(Ans) :- solve(3*x+5*(x-3)=(5*x+3)-3*(x-2),x,Ans).


b1hard(Ans) :- solve(1/x=2/(4-x),x,Ans).

b2hard(Ans) :- solve(a/(x-b)=c+d,x,Ans).
```

```
   Ans) :- solve(2*(4*x+2)-3*(1+2*x)=0,x,Ans).

  (Ans) :- solve(3*(x+(a+b))+2*(b+(x+a))=1,x,Ans).


 d(Ans) :- solve((r+y+z)*x/(1/p+1/q)=d,x,Ans).

rd(Ans) :- solve(y+2*(3*x/(x+1))-(4+y)*(3*x/(x+1))=1,x,Ans).
```

```
/*      MECHO.PRB
A Selection of Algebra Problems taken from the mechanics project
Originally collected by Alan Bundy   10.5.79
Rephrased and updated for the new PRESS by Leon  23.9.81
*/

:-      assert((present(mecho))).

/*simple pulley*/
simppull(Ans) :- simsolve(
    m1*g*cos(180) + (1*tsn + 0) = m1*(a1*1)  &
    m2*g*1 + (cos(180)*tsn + 0) + 0 = m2*(a1*1)  &
    true , [tsn,a1] , Ans).

/*pulley and table with friction*/
pulltab(Ans) :- simsolve(
    m1*g*cos(270) + (1*tsn + (cos(-270)*reaction1 + 1*mu*reaction1 + 0))
      + 0 = m1*(a1*1)  &
    m2*g*1 + (cos(180)*tsn + 0 ) + 0=m2*(a1*1)  &
    m1*g*1 + (cos(270)*tsn + (reaction1 + cos(270)*mu*reaction1 + 0)) + 0
      = m1*(a1*cos(270))  &
    true,
  [reaction1, tsn, a1]  , Ans).

/*natural language problem four*/
nl4(Ans) :- simsolve(
    v^2=0^2 + 5*(60*60)^2 / (1760*3)*2000/1760  &
    true ,  [v] , Ans).

/*simple car problem*/
car(Ans) :- simsolve(
    1760*3*d0=0*60*t + 1/2*a*60*t^2  &
    v = 0 + a*60*t  &
    true ,  [t, v]  , Ans).

/*tower p21 no13 Palmer & Snell*/
tower1(Ans) :- simsolve(
    v = vel1 + 32*t2  &
    d2 = vel1*t2 + 1/2*32*t2^2  &
    true ,  [vel1, v]  , Ans).

/*train problem p18 Palmer & Snell*/
train(Ans) :- simsolve(
    t0 = t1+(t2+(t3+0))  &
    45/60 = 0 + 2^(-1)/60^2*t1  &
    45/60*t2 = d2  &
    0 = 45/60 + (-2)/60^2*t3  &
    7 = d1+(d2+(d3+0))  &
    d1 = 0*t1 + 1/2*2^(-1)/60^2*t1^2  &
    d3 = 45/60*t3 + 1/2*(-2)/60^2*t3^2  &
    true  ,  [t0, t1, t2, t3, d2, d1, d3]  , Ans).

/*tower to determine value of g*/
tower2(Ans) :- simsolve(
    v = 0 + a0*t0  &
    vc = 0 + a1*t1 &
    v = vc +a1*t2  &
    t0 = t1 + (t2 + 0)  &
    d2 = vc*t2 + 1/2*a1*t2^2  &
    d1 = 0*t1 + 1/2*a1*t1^2  &
    true  ,  [v, vc, a1, t0, t1, a0]  , Ans).

/*INEQUALITIES*/

stvineq(Ans) :- solveineq(x > 1/(1+sin(y)^2), x, Ans).

conjineq(Ans) :- solveineq(2*g*h1>0 & 2*g*(h1-h2)>=0 &
                        2*g*(h1-h2)>0 & 2*g*(h1-h2)>=0 & true, X, Ans).


%   Press cannot solve problems involving real(E^K) as it once used to,
%   to here are temporary formulations which avoid that pattern.
%   bloc fails because fixvar can't find a suitable variable, while
%   loop gets almost to the end and can't find the maximum.

bloc(Val) :- (X=h1;X=h2), solveineq(
        sqrt(2*g*h1) > 0 &
        2*g*(h1-h2) >= 0 &
        sqrt(2*g*(h1-h2)) > 0 &
        2*g*(h1-h2-h3*tan(t)) >= 0 & true, X, Val).

/* Old formulation
        bloc(Val) :- solveineq(sqrt(2*g*h1)>0 & real(sqrt(2*g*(h1-h2))) &
          sqrt(2*g*(h1-h2))>0 & real(sqrt(2*g*(h1-h2-l1*tan(t)))) & true,
          X,Val).        */

loop(Val) :- min(
        2*g*h-2*g*r >= 0 &
        sqrt(2*g*h-2*g*r) > 0 &
        2*g*h-4*g*r >= 0 &
        2*g*h - 2*g*r*(1+sin(ang)) >= r*g*sin(ang) &
        sqrt(2*g*h - 4*g*r) > 0 &
        2*g*h - 2*g*r*(1+sin(ang)) >= r*g*sin(ang) & true, h, Val).

/* Old formulation
        loop(Minval) :- min(real(sqrt(2*g*h-2*g*r)) & sqrt(2*g*h-2*g*r)>0 &
          real(sqrt(2*g*h-4*g*r)) & (2*g*h-2*g*r*(1+sin(ang))) >=
         r*g*sin(ang) &  sqrt(2*g*h-4*g*r)>0 &
          (2*g*h-2*g*r*(1+sin(ang))) >= r*g*sin(ang) & true,  h,Minval). */

dome(Minval) :- min(m*g*(3*sin(d)-2)>=0 & true,d,Minval).


/*CURRENT PROBLEMS*/

pb1(A) :- eval(arcsin(2^(-1)), A).

pb2 :- non_zero(0*2^(-1)).

pb4(Ans) :- solve( vc= (-(-t2)^(-1))*t1*(a0*(t1+t2)+(-vc))*t2^2,vc,Ans).

pb3( (-t2)^(-1)*(-(1+t1*(-(-t2)^(-1)))^(-1))*t1*a0*(t1+t2)).

pb5((r*g*sin(ang)+(-1)*((-1)*(g*2*r*(1+sin(ang)))))*(2^(-1)*g^(-1)) ).
```

```
/*CURRENT PROBLEMS*/

/* interval and eval problems */

pbe(I) :- interval(x^2,I).



/* tidy problems */

pbd( (-t2)^(-1)*(-(1+t1*(-(-t2)^(-1)))^(-1))*t1*a0*(t1+t2)).

pbb(d1+(-1)*1*2^(-1)*(t2*(-2)^(-1)*(t2^(-1)*d2*1)+t2^(-1)*d2*1*(t2*(-2)^(-1)))).

pba((-1)*1*2^(-1)*(t2*(-2)^(-1)*(t2*(-2)^(-1)))).

pbc((-1)*1*2^(-1)*(t2^(-1)*d2*1*(t2^(-1)*d2*1))).
```