::

Informal history of PRESS from mail messages sent about innovations

::

From Lawrence[400,441] on April 3, 1981 at 10:36 PM

New gimmick in Press (code lifted from Robs parser) for top level work :- the following procedures are provided (in extras:facile):

```
so          -  read equation and solve for x.
redo        - re-solve last equation.
show        -  show the remembered equations
               (Top ones are latest)
oops        -  remove last equation
bye         -  show all equations (and force into log) and halt.
```

STOP PRESS

Press can now solve reciprocal polynomial equations! (see Leon)

om Richard[400,422] on April 8, 1981 at 8:49 PM

I've implemented your proposed Log algorithm, with a variety of embellishments (so log(-3,-27) = 3 !!)

It works whenever the result should be an integer (I think), BUT it falls flat on its face for log(4,8).

The code finally gets around to asking the question "for gien B and X, find integers N, D such that B^N = X^D". I assumed that the answer was to find G=gcd(B,X), then D = log(G,B) and N = log(G,X). Well, this is true as far as it goes, but D and N aren't always integers. Then again, there may be a G for which N and D are integers, and it divides the gcd, but it isn't always equal to it. What the right answer is escapes me at the moment.

From Richard[400,422] on April 8, 1981 at 10:03 PM

There is a new version of LONG, currently available as LONGER.[400,422] The only change is that I've installed a new LOG algorithm, the outline of which was suggested by Leon. It has the interesting property that (if I were to use long integer arithmetic throughout) it will return the correct log if there is one, and fail only if there is not. As far as I can see the time to scover that a log doesn't exist shouldn't be markedly different from the ime to discover that a similar log does exist; it might even be less. There is still room for improvement, but not much.

Please test this stuff; I've tried it on a range of more or less plausible log(B,X)-es (some very implausible indeed, e.g. log(X,0)=+/-infinity, log(-3,-37) = 3, and so on) without finding any mistakes (but don't try log(10,10^3000); it doesn't take forever but it seems like it), but there may well be some obvious things I've missed.

From Richard[400,422] on April 9, 1981 at 3:04 PM

DSKA:LONGER.[400,422] now incorporates a new gcd algorithm, due in outline to Leon. It is now largely independent of the division code, so only "div", "mod", "fix", "++", and "--" should be vulnerable to remaining bugs in division (I think).

The outer wrapping of GCD now checks for A=1 or B=1 as well as for A=0 or B=0, since a majority of calls to GCD have one argument = 1.

From Lawrence[400,441] on April 9, 1981 at 7:15 PM

I shall think a bit about tidy over the weekend perhaps. The additions we made to mulfin and plusfin work but they are far from being the correct (ie general) solution. For example:

```
((a*2)+0)*5)            ==>             a * 10
((a*2)+0)*5*b           ==>             a * 2 * b * 5
```

(Ignore extra bracket in first!). The general problem is that I assumed that when bag sweepers bottomed out the tidied version of this bottom case would not be something that could be incorporated into the bag. This is false. The general solution is to take as invarient that all sub tidies will return something in 'tidy normal form' and to use combining rules which combine things in this form properly. In the cases we are having problems with both forms to be combined may be full bags of the same sort. In the code at the moment there is a distinction in that one bag is in a "final form" whereas the other is in an 'intermediate form' (ie Left and Ltag form) and these have slightly different properties. One would want to be normalised to the other (final -> intermediate I reckon). There is a thought crossing my mind that my current data stuctures for the intermediate form (and certainly for the final form) will not allow two complete bags to be merged without actually copying most of one of them. I wonder if I should care about this! I estimate that worrying about the amount of unneeded cons' performed has made the thing several orders of magnitude dirier to do than would otherwise have been the cease. So I have now proved to my satisfaction that such rubbish is not worth thinking about. Leave the system to sort it out.

I had thought briefly about combining objects already in tidy normal form to produce a new tidy normal form expression under various operations but thought I didn't really need it so save it up (other things to do). Oh well - now its needed!

From Bernard[400,4322] on April 11, 1981 at 7:42 PM
Hello,on Friday I discovered that when homos failed nasty function etc had a go on the old equation,ie the one before chunk was called.I remembered that we had previously overcome this by putting a cut after changeunknown.Looking (or even just looking!) at the code it seemed that the cut had got omitted from the go to be called directly by
solve1 for it to have the desired effect.In my copy of solve(in ,4322,homos)I have therefore put the full set of clauses back in homos and change of unknown I don't know about the call by poly so I left it.
Looking at the code for chunk it seems wrong.The call mentions subs, the rest doesn't.Must be wrong?Anyway,see you on Monday or Tuesday

From Richard[400,422] on April 11, 1981 at 8:37 PM
LONGER[,422] => ARITH:LONG => ARITH:LONG.OLD

Changes:
new log routine
new gcd routine
eval now has gcd(A,B) - takes gcds of numerator, denominator
eval now has fix(A) - gives integer part of A
number/1, number/5, and eval all know about xwd(,).
So eval(1000000,X) gives X=number(+,[0,100],[]) as it should. This turned out to be a very simple change.

From Sterling Hps[400,4321] on April 14, 1981 at 5:08 PM

Changes have been made to homogenisation calls and change of unknown calls. There is a new clause changevar which makes a given substitution and then solves the resultant equations as in change of unknown.

From Bernard[400,4322] on April 16, 1981 at 4:06 PM

I've found two 'buss'.One is to do with the interval package I think.
non_zero(29^(1/2)) fails.The other is more minor and is to do with poly_solve
.One equation I obtained was something like (x^2+3*x+7)*(2^(i

From Bernard[400,4322] on April 21, 1981 at 2:05 PM
non_zero(29^(1/2)) now succeeds.Alan had modified his version already to
sort out this problem.Presumably we should move his version to the press area.
My routine for 'helpins' collection+attraction seems to work fine,code in clean
+solve in my area.This also means that sin(2z)=sin(z) is solvable,not just
sin(2z)-sin(z)=0.

From Leon S[400,4321] on April 22, 1981 at 9:55 AM
Operator declarations now appear in press:press.ops
instead of init.mec.
If you are using your own read-in file, this file should
be changed to consult press.ops first.

p.s. The dreaded 119 has gone on vacation.

From Richard[400,422] on April 22, 1981 at 5:18 PM
There is a new version of ARITH:LONG.
1) There is new code to deal with xwd(_,_) which works compiled as well as
interpreted. The largest number you can type in is roughly 3.435*10^9.

2) I have renamed practically everything. Don't despair, I've just changed
the suffixes, so that a=rational, z=signed integer, n=natural(or 0).

3) A clause has been added to make 0^(1/N) = 0; I'd forgotten it before.

4) The radix is now 10^5 instead of 10^4.

5) various minor internal changes have been made, with no user impact (hope).

From Leon S[400,4321] on May 1, 1981 at 5:54 PM
The polynomial package in PRESS is now written up as note 82 in the MACHO
folder.

From Bernard[400,4322] on May 12, 1981 at 1:22 PM
A problem with odd_anti_symmetric.
got an **ERROR evaluate(number(+,[4096],[4543]) (es!!).
,racing it seems you use a X is Y to check if its anti_symm,and
this causes problems with rational coeffs.I can fix it if you don't
want to,but I don't know if the same problem occurs elsewhere
Bernard

From Leon S[400,4321] on May 8, 1981 at 12:56 PM
change poly_method so that multiplying through by last term
is only done once.

From Leon S[400,4321] on May 18, 1981 at 3:33 PM

The latest version of press (18 May) now contains Alan's simultaneous
equation code.
The old version of simeq is in press:simeq.old.
There are also slight changes to the polynomial stuff.
The new code appears in the press folder.

From Bundy Hps[400,405] on May 20, 1981 at 12:07 PM
Now I have fixed simsolve, so that it can find particular
solutions as well as general ones, ExPress can specialize the general

quartic as predicted,as well as the quadratic and cubic.

From Bernard[400,4322] on May 29, 1981 at 1:07 PM
Message as requested.
I've written a method called nasl that isolates function symbols in the
case where they dominate all occurrences of the unknown.
        This can now solve sin(x^2+x+2)=1/2,and (x^2+x+3)^6=89
It cannot solve (x+1)^(1/2)+x=4,this should be done by the old nasty function.
I've placed the call after isolate and before poly_method,so that examples
like the second one can be solved.

From Bernard[400,4322] on May 29, 1981 at 1:16 PM
Message 3!
Just a list problems.
1) Most important is the loopins the current nasty causes.Maybe this is easily
dealt with.I'm waitins to see what the AI2 suy produces.
2)The tidy problem with nested exponentiation.Probably a theoretical rather
than practical problem.
3)Tidy doesn't know that log(X,X)=1,and log(X,1)=0 (and unifying log(i,i)=0=1!).The
is reloaded from prolog.I
  n't know whether we can do it before.
4,Big problem!Collection and attraction and the anyone routine.Project!.
Anyway ,that's all I can think of for now,perhaps a problem mail file
should be set up.

Bernard

From Bernard[400,4322] on May 29, 1981 at 1:52 PM
Problem. sin(x^2+x)=1/2 solves fine,cos(x^2+x)=1/2 fails.To the unintiated the
two equations seem similar!.The problem lies in isolate,specifically
isolate1.With the cos problem the goal
isolate1([],x^2+x=blaa # x^2+x=blaa1,_434=_435) is set up and fails,no
matching.with ths sin the disJunction is not present so the goal
succeeds.It seems curious that this has not caused problems before.I don't
know what to do about it yet,but I'll think!
P.S. For reloading use my poly rather than the version you copied.

From Leon S[400,4321] on June 7, 1981 at 5:40 PM

The PRESS folder has been updated to be the latest version of PRESS
   archived this weekend.

Recent changes have been made to Bernard's homog files, the polynomial
methods. Bernard has added an isolation-type method in the file
press:nasl.
Also, the initialisations for Mecho which were in press:init.mec
have been moved to extras:init.mec, and are no longer in the
standard version of PRESS.

Leon

From Bernard[400,4322] on June 8, 1981 at 2:24 PM
I've changed the predicate norm in weaknf to init_norm to avoid a clash
with Long.The remainder theorem code has been added to polpak and poly.
A minor change has been made to factor_out in Polpak

From Richard[400,422] on June 8, 1981 at 4:11 PM
        I have fixed the comments to mention 99999 instead of 9999.
I have also renamed 'norm' to 'standardise'. I see that I have maligned
Lawrence, he had called it 'norm_number'.

From Lawrence[400,441] on June 24, 1981 at 7:50 PM

PRESS reloaded at 19:50:13 on 24 Jun 81

CC:Bundy Hps,Sterling Hps,Silver Hps,Okeefe Hps,Byrd Hps

From Lawrence[400,441] on July 1, 1981 at 8:38 PM

PRESS reloaded at 20:38:14 on 1 Jul 81

CC:Bundy Hps,Sterling Hps,Silver Hps,Okeefe Hps,Byrd Hps

From Lawrence[400,441] on July 8, 1981 at 2:26 PM

New version of UTIL with slight fix to LONG. Not worth reloading anything until necessary.

CC:Bundy Hps,Sterling Hps,Silver Hps,Okeefe Hps,Byrd Hps

From Leon S[400,4321] on July 15, 1981 at 5:43 PM
~ESS has exhibited some interesting behaviour when running some examples used by Clayton Lewis as test examples in a psychological study.
The equations can be found in extras:lewis.
Of particular note is a1hard.
The first step is a nasty fn. step to divide through by a denominator.
Then instead of recognising the resultant polynomial equation it tries many isolation steps until it eventually does recognise the polynomial.

Another point : tidy left (a*5+b*5-1)*(-1/5) alone.

the common subterm wasn't found in d2hard.

From Leon S[400,4321] on July 16, 1981 at 3:54 PM
Suggestion to improve isolation of a*x = b,
where non_zero(a) fails.
Have an alternative clause in isolate and/or vet solutions
rather than change the interval package.

From Leon S[400,4321] on July 18, 1981 at 3:49 PM
~an
I've changed the GOALS file so that the command tmprun runs a list of standard examples all successfully.
This is only a temporary measure as I think we should organise what the examples are a little more closely. Bernard has done something about this and has compiled a list of exam questions in file extras:exam. We should probably wait until September and have as an initial priority to sort the few remaining questions of this type.
Leon

From Leon S[400,4321] on July 21, 1981 at 1:45 PM

Bernard
I gave a math. reasoning seminar (see note 92). While running some of the examples in extras:lewis some interesting ideas on how to extend PRESS and what limitations the current methods have emerged.

One exale for you to think about is d2hard. It is essential for change of unknown to pick up the common subterm. I've traced why it doesn't - the problem is that it calculates that x* (x+1)^-1 only occurs once in x*(x+1)^-1 *6 + (y+4)*x*(x+1)^-1 *-3,

whereas it clearly occurs twice!
Any ideas for overcoming this?

Leon

From Lawrence[400,441] on August 2, 1981 at 10:34 PM

The files LONG and TIDY have been moved from their old home in arith; and they now live in util;

This is because they are now automatically present in UTIL, and I want to keep things together. Also, the contents of util; will now be placed on MasTapes containing the Prolog system when they are dispatched around the world.

I don't think this will affect anybody other than when they are looking for the sources. (/press in press; is unaffected).

Lawrence

CC:Bundy Hps,Sterling Hps,Silver Hps,Okeefe Hps,Byrd Hps

From Lawrence[400,441] on August 10, 1981 at 10:17 PM

PRESS reloaded at 22:17:04 on 10 aug 81

CC:Bundy Hps,Sterling Hps,Silver Hps,Okeefe Hps,Byrd Hps

From Lawrence[400,441] on September 3, 1981 at 5:50 PM

PressProject

I have updated PRESS;FLS, the list of component files, and tested the new XREF by running over PRESS. I archived the old pressx.mem (under 421) and the files pressx.rno and pressx.mem are the new cross reference listings. A copy has been sent to the printer and will appear in the terminal room shortly.

Lawrence

;Bundy Hps,Sterling Hps,Silver Hps,Okeefe Hps,Byrd Hps

From Bernard[400,4322] on September 4, 1981 at 2:09 PM
The following things have been done to press by me:
1) Log method added
2) New nasty method added
3) more printing out.In solve the changes are
printing out the tidied input equation if it doesn't match original,
printing out the polynomial forms of factorised polys
In poly printing 'By inspection' in remainder theorem and adding
(no I didn't!!).
4) Homog has been jazzed up.It can now deal with cos(x+45) etc,though
press as a whole can't solve any of the examples of this kind of thing.
Negative angles are now dealt with correctly.
Before all hyperbolics had e^(k*x) as reduced term,Now sech-tanh,cosh-sinh,
and coth-tanh pairs are dealt with in a similar way to the trig equivalents
(This change occured because of an exam question)
5) exam has been modified.All questions are called one of the following
aeb(N),lon(N),oxf(N),dlon(N),integer n
aeb has been extended,dlon are London syllabus D Alevel questions.
To run the AEB type aebrun,and similarly for the others.

To run aeb(N) onwards type aebrecurse(N).(run means time as well)
Someone should extract a subset of problems we wish to make standard.
6) A slight change has occured in arbint,instead of saying 'where n34 is an
arbitrary integer.'double line feed,it says 'Letting n34 be an arb etc',no
full stop and single line feed.This makes the output more logical.
        Thats all I can remember
7) apart from changing misc,chunk,homos.I've written in misc the
4 argument predicate correspond,which works as:
correspond(X,List1,List2,Y) :- nmember(X,List1,N),nmember(Y,List2,N),!.
although I've written it without nmember(so it works easily whether
X or Y is instantiated)This construction occurred a lot in previous code of
mine so I replaced it by correspond
                That really is all I can remember

From Bernard[400,4322] on September 5, 1981 at 3:25 PM
PRESS solves at the moment:
19 out of 36 AEB problems,
CC:Silver Hps,Sterling Hps

From Bernard[400,4322] on September 5, 1981 at 3:30 PM
  rry about that,to recap:
 'RESS solves:
19 out of 36 AEB problems,of the 27 single equations it solves 18
19 out of 21 London problems,18 out of 20 single equations (100% success on sim
solve!!)
11 out of 13 Oxford,(2 out of 3 Alevel ones)9 out of 10 single equations
7 out of 10 London syllabus D,7 out of 10 single equations(ie there wern't any sims
        This gives a total of 56 out of 80 problems
        52 out of 67 single equation problems
        Pretty good really.
CC:Silver Hps,Sterling Hps

From Bernard[400,4322] on September 5, 1981 at 3:39 PM
I've modified the suess routine in poirak,for roots it only considers
factors of the constant term between 9 and -9

From Bernard[400,4322] on September 7, 1981 at 1:47 PM
I've replaced the clauses in solve with cond_print(Old,New),which I've put in
misc,cond_print prints 'Tidying to New' if New doesn't match Old.
 'ote:We adopt the prolog convention of variable names begin with an
upper-case letter,EXCEPT in the case of 'Tidying' and 'EXCEPT'.(the former
is a word wheras 'New' is a variable,that is to say ...etc,etc.)

From Lawrence[400,441] on September 9, 1981 at 3:23 AM

PressProject

  Alan Bornings new matcher is now settled in [400,421,press,match].
This directory now holds every thing off the tape he sent.

  All the old matcher stuff has been archived under [400,421] as follows:

        OLDMAT.001   original mas file
        OLDMAT.002   new mas file of all the individual files
                                that were lying around. Includes 'cubic'
                                which is not in OLDMAT.001 and presumably
                                any other changes that people have made.

Nothing of the old matcher remains in the MATCH directory - ie all the stuff
in there is fresh from Alan Bornins.

[NB for reference: I now save all MAS files with three disit extensions
    (like 001 etc above). If you see one of these then it is a .MAS file
    and should be renamed if/when retrieved so that you can talk to SUBFIL
    about it. I recommend this naming strategy for the reason that it keeps
    the first part of the filename the same on FMS as it was on disk,
    which makes it easier to find with odir, amongst other advantages
]

Lawrence

CC:Bundy Hps,Sterling Hps,Silver Hps,Okeefe Hps,Byrd Hps

From Bernard[400,4322] on September 10, 1981 at 1:46 PM
If you do a reload could you move over the following files from my area
chunk,collec.ax,log,weaknf.
The changes are:
weaknf now uses decomp instead of parsei,this is quicker,and I've removed
some clauses that wern't needed.
chunk has been optimised (maybe!!),still doesn't sort out the lewis problem
  s has been slightly changed to deal with the changes to weaknf!!
it's all for the good as it makes the method more robust,I didn't
realise that I was assuming something before.
collec.ax has had the trig clauses removed.

I've changed the nasty on the press area to reflect the new names for
the weaknf method,but it works exactly the same way.The new nasty is
in nasty +mult.new on my area but I don't feel certain enough to have
it loaded into PRESS just yet.

From Bernard[400,4322] on September 10, 1981 at 1:54 PM
I ran the Lewis problems with Press (with non_zero deactivated) + the
result is in lewis.sol on my area.It solves all but one
                Bernard

From Bernard[400,4322] on September 12, 1981 at 4:12 PM
Hi,I've written a file,sim2,that does elimination in sim.eqns by
homogenization,it works on some of the outstanding A level questions
(Top level is sim1(Equation,Listofunknowns,Ans),rather than simsolve)

From Leon S[400,4321] on September 13, 1981 at 3:20 PM
find_int2 bug fixed. Due to exponentiation code.

From Richard[400,422] on September 13, 1981 at 8:22 PM
        Some of the examples in Press:Goals. don't work unless
Extras:Init.Mec has been loaded as well. Which do you prefer me
to do: put a comment on those examples, put checking code on those
examples to load Init.Mec when needed but not before, make Goals
load Init.Mec, or what?
        Even then, some of the examples still don't work due to
buss in the inequality routines. Somewhere along the line, the
code which used to cope with
        real(E^(N/D)) <- odd(D) -> true ; E >= 0
has been lost (I suspect this was Tidy), so I have taken the liberty
of rewriting the bloc/1 and loop/1 problems which used it, leaving the
old code as comments. I have NOT printed the new file.

From Bernard[400,4322] on September 14, 1981 at 2:06 PM
Adding missing attraction axioms,and using sim instead of simsolve
we can solve 6 more problems aeb:13,18,25,29,31 and 33.

I haven't yet moved los,solve,collec.ax,attrac.ax,nas1 over yet.
This system does run all of tmprun!.
Bernard

From Bernard[400,4322] on September 19, 1981 at 5:32 PM
I've reloaded Press with the new revised files.The error in INT seems
to be lack of a procedure measure/2,which is listed as an import from
the mecho database.I don't understand!.
Tmprun all run,but in one or two there are slight changes
from the previous methods of solution.
This is because I've modified isolate to take into account non_zero.
Now when it checks the condition of isolax,if the condition is
not 'non_zero(X)' it works as before.If non_zero(X) is true
then fine,if X is evaluated to zero,then fail (as before),
and now the new bit.If neither of these conditions hold,Press
outputs a message 'Assuming X to be non_zero'.If you don't like/want this
take it out of isolat
Bernard
CC:Byrd Hps,Sterling Hps

From Leon S[400,4321] on September 23, 1981 at 2:07 PM
I've modified the structure of the goals file, which no longer
exists. Instead, there is a file runex which runs the standard examples
and takes timings as before.
The clauses are smallrun, run (which has been changed),
mechorun, aebrun,lonrun,dlonrun,oxfrun and lewisrun.
Each run command consults the relevant files
(one of probs.tes, probs.mec&init.mec, exam and lewis which are all in
extras).
Leon

From Leon S[400,4321] on September 23, 1981 at 4:16 PM
The files extras:probs.tes and extras:probs.mec
have been renamed to extras:testex.prb and extras:mecho.prb
respectively, as suggested by Alan. Similarly, lewis is now lewis.prb
Leon
CC:Bundy Hps,Sterling Hps,Silver Hps,Okeefe Hps,Byrd Hps

TRACE

```
PRESS  (1 Jul 81)

| ?- [-soals].

soals reconsulted   158 words      1.25 sec.

yes
| ?- tmprun.
Solving log(e, x + 1) + log(e, x - 1) = 3 for x

    log(e, (x + 1) * (x + -1)) = 3


    e = ((x + 1) * (x + -1)) ^ (1/3)
        (by Isolation)

    log(e, x ^ 2 + -1) = 3


    x ^ 2 + -1 = e ^ 3
        (by Isolation)

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

    x = (e ^ 3 + -1 * -1) ^ (1/2) # x = -1 * (e ^ 3 + -1 * -1) ^ (1/2)
        (by Isolation)

Answer is :
( X1 # X2 )
  where :
    X1 =  x = (e ^ 3 + 1) ^ (1/2)
    X2 =  x = (e ^ 3 + 1) ^ (1/2) * -1


It then took 1492 milliseconds and produced answer
( X1 # X2 )
  where :
    X1 =  x = (e ^ 3 + 1) ^ (1/2)
    X2 =  x = (e ^ 3 + 1) ^ (1/2) * -1

Solving (2 ^ x ^ 2) ^ x ^ 3 = 2 for x

    2 ^ x ^ 5 = 2
      |
    x ^ 5 = log(2, 2)
      |   (by Isolation)
      |
    x = log(2, 2) ^ (1/5)
        (by Isolation)

Answer is :
X1
  where :
    X1 =  x = 1
```

```
expeen took 421 milliseconds and produced answer
X1
  where :
    X1 =  x = 1


Solvins ((2 ^ cos(x) ^ 2 * 2 ^ sin(x) ^ 2) ^ sin(x)) ^ cos(x) = 2 ^ (1 / 4) for x

    (2 ^ cos(x) ^ 2 * 2 ^ sin(x) ^ 2) ^ (sin(x * 2) * (1/2)) = 2 ^ (1/4)


    (2 ^ (cos(x) ^ 2 + sin(x) ^ 2)) ^ (sin(x * 2) * (1/2)) = 2 ^ (1/4)


    2 ^ (sin(x * 2) * 1 * (1/2)) = 2 ^ (1/4)


    sin(x * 2) * 1 * (1/2) = los(2, 2 ^ (1/4))
        (by Isolation)

    sin(x * 2) * 1 = los(2, 2 ^ (1/4)) * 2
        (by Isolation)

    sin(x * 2) = los(2, 2 ^ (1/4)) * 2 * 1
        (by Isolation)
        where n1 denotes an arbitrary inteser.

    x * 2 = n1 * 180 + -1 ^ n1 * arcsin(los(2, 2 ^ (1/4)) * 2 * 1)
        (by Isolation)

    x = (n1 * 180 + -1 ^ n1 * arcsin(los(2, 2 ^ (1/4)) * 2 * 1)) * (1/2)
        (by Isolation)

Answer is :
X1
  where :
    X1 =  x = -1 ^ n1 * 15 + n1 * 90


t  ieen took 6179 milliseconds and produced answer
X1
  where :
    X1 =  x = -1 ^ n1 * 15 + n1 * 90


Solvins 1 / x ^ 2 = 1 / x for x

Multiply throush by x ^ 2 to set a polynomial
 x = 1 is a solution

Answer is :
X1
  where :
    X1 =  x = 1


nespolveen took 143 milliseconds and produced answer
X1
  where :
    X1 =  x = 1
```

```
Solving a ^ (x + 1) + a ^ (2 * x) = c for x

Homogenized equation is a ^ 1 * a ^ x + (a ^ x) ^ 2 = c

New equation is a * x1 + x1 ^ 2 = c

Using quadratic equation formula
 x1 = (a * -1 + (a ^ 2 + c * 4) ^ (1/2)) * (1/2) # x1 = (a * -1 + (a ^ 2 + c * 4)
Applying substitution
    x1 = a ^ x

  to    :
    x1 = (a * -1 + (a ^ 2 + c * 4) ^ (1/2)) * (1/2) # x1 = (a * -1 + (a ^ 2 + c *

  gives :
    a ^ x = (a * -1 + (a ^ 2 + c * 4) ^ (1/2)) * (1/2) # a ^ x = (a * -1 + (a ^ 2


    x = log(a, (a * -1 + (a ^ 2 + c * 4) ^ (1/2)) * (1/2))
        (by Isolation)

    x = log(a, (a * -1 + (a ^ 2 + c * 4) ^ (1/2) * -1) * (1/2))
        (by Isolation)

Answer is :
( X1 # X2 )
  where :
    X1 =  x = log(a, (a ^ 2 + c * 4) ^ (1/2) * (1/2) + a * (-1/2))
    X2 =  x = log(a, (a ^ 2 + c * 4) ^ (1/2) * (-1/2) + a * (-1/2))


homogen took 2162 milliseconds and produced answer
( X1 # X2 )
  where :
    X1 =  x = log(a, (a ^ 2 + c * 4) ^ (1/2) * (1/2) + a * (-1/2))
    X2 =  x = log(a, (a ^ 2 + c * 4) ^ (1/2) * (-1/2) + a * (-1/2))


Solving cos(x) ^ 2 + b * cos(x) = c for x

Substituting x2 for cos(x)
 gives x2 ^ 2 + b * x2 = c

Using quadratic equation formula
 x2 = (b * -1 + (b ^ 2 + c * 4) ^ (1/2)) * (1/2) # x2 = (b * -1 + (b ^ 2 + c * 4)
Applying substitution
    x2 = cos(x)

  to    :
    x2 = (b * -1 + (b ^ 2 + c * 4) ^ (1/2)) * (1/2) # x2 = (b * -1 + (b ^ 2 + c *

  gives :
    cos(x) = (b * -1 + (b ^ 2 + c * 4) ^ (1/2)) * (1/2) # cos(x) = (b * -1 + (b ^

        where n2 denotes an arbitrary integer.

    x = 2 * n2 * 180 + arccos((b * -1 + (b ^ 2 + c * 4) ^ (1/2)) * (1/2)) # x = 2
 ^ (1/2)) * (1/2))
```

```
        (by Isolation)
        where n3 denotes an arbitrary integer.

    x = 2 * n3 * 180 + arccos((b * -1 + (b ^ 2 + c * 4) ^ (1/2) * -1) * (1/2)) # :
 * 4) ^ (1/2) * -1) * (1/2))
        (by Isolation)

Answer is :
( ( X1 # X2 ) # ( X3 # X4 ) )
  where :
    X1 =  x = n2 * 360 + arccos((b ^ 2 + c * 4) ^ (1/2) * (1/2) + b * (-1/2))
    X2 =  x = n2 * 360 + arccos((b ^ 2 + c * 4) ^ (1/2) * (1/2) + b * (-1/2)) * -
    X3 =  x = n3 * 360 + arccos((b ^ 2 + c * 4) ^ (1/2) * (-1/2) + b * (-1/2))
    X4 =  x = n3 * 360 + arccos((b ^ 2 + c * 4) ^ (1/2) * (-1/2) + b * (-1/2)) * -


chunkeen took 2522 milliseconds and produced answer
( ( X1 # X2 ) # ( X3 # X4 ) )
    are :
    X1 =  x = n2 * 360 + arccos((b ^ 2 + c * 4) ^ (1/2) * (1/2) + b * (-1/2))
    X2 =  x = n2 * 360 + arccos((b ^ 2 + c * 4) ^ (1/2) * (1/2) + b * (-1/2)) * -
    X3 =  x = n3 * 360 + arccos((b ^ 2 + c * 4) ^ (1/2) * (-1/2) + b * (-1/2))
    X4 =  x = n3 * 360 + arccos((b ^ 2 + c * 4) ^ (1/2) * (-1/2) + b * (-1/2)) * -


Simultaneously solving :
    a = b
    b = c
    c = 1
For [a, c, b],
Solving a = b for a

Answer is :
X1
  where :
    X1 =  a = b

Ap lying substitution
    a = b

  to    :
    b = c
    c = 1

  gives :
    b = c
    c = 1

Solving b = c for c

Answer is :
X1
  where :
    X1 =  c = b

Applying substitution
    c = b

  to    :
    c = 1
```

```
  gives :
    b = 1

Solving b = 1 for b

Answer is :
X1
  where :
    X1 =  b = 1

Substituting back in c solution
Applying substitution
    b = 1

  to    :
    c = b

    yes :
    c = 1

Substituting back in a solution
Applying substitution
    b = 1
    c = 1

  to    :
    a = b

  gives :
    a = 1


Final Answers are :
( ( X1 & X2 ) & X3 )
  where :
    X1 =  b = 1
    X2 =  c = 1
    X3 =  a = 1

simpeqns took 376 milliseconds and produced answer
( ( X1 & X2 ) & X3 )
  where :
    X1 =  b = 1
    X2 =  c = 1
    X3 =  a = 1


Simultaneously solving :
    m1 * s * cos(180) + (1 * tsn + 0) = m1 * (a1 * 1)
    m2 * s * 1 + (cos(180) * tsn + 0) + 0 = m2 * (a1 * 1)
For [tsn, a1].
Solving m1 * s * cos(180) + (1 * tsn + 0) = m1 * (a1 * 1) for tsn

    tsn = m1 * a1 + -1 * (m1 * s * -1)
        (by Isolation)

Answer is :
X1
  where :
```

```
    X1 =  tsn = (a1 + s) * m1

Applying substitution
    tsn = (a1 + s) * m1

  to    :
    m2 * s * 1 + (cos(180) * tsn + 0) + 0 = m2 * (a1 * 1)

  gives :
    m2 * s + (a1 + s) * m1 * -1 = m2 * a1

Solving m2 * s + (a1 + s) * m1 * -1 = m2 * a1 for a1
 a1 = (m1 * -1 + m2) * s * (m1 * -1 + m2 * -1) ^ -1 * -1 is a solution

Answer is :
X1
  where :
    X1 =  a1 = s * (m1 * -1 + m2 * -1) ^ -1 * m1 + s * (m1 * -1 + m2 * -1) ^ -1 *

Substituting back in tsn solution
f  lving substitution
    a1 = s * (m1 * -1 + m2 * -1) ^ -1 * m1 + s * (m1 * -1 + m2 * -1) ^ -1 * m2 *

  to    :
    tsn = (a1 + s) * m1

  gives :
    tsn = (s * (m1 * -1 + m2 * -1) ^ -1 * m1 + s * (m1 * -1 + m2 * -1) ^ -1 * m2 )


Final Answers are :
( X1 & X2 )
  where :
    X1 =  a1 = s * (m1 * -1 + m2 * -1) ^ -1 * m1 + s * (m1 * -1 + m2 * -1) ^ -1 *
    X2 =  tsn = (s * (m1 * -1 + m2 * -1) ^ -1 * m1 + s * (m1 * -1 + m2 * -1) ^ -1

si ePull took 1456 milliseconds and produced answer
(   . & X2 )
  where :
    X1 =  a1 = s * (m1 * -1 + m2 * -1) ^ -1 * m1 + s * (m1 * -1 + m2 * -1) ^ -1 *
    X2 =  tsn = (s * (m1 * -1 + m2 * -1) ^ -1 * m1 + s * (m1 * -1 + m2 * -1) ^ -1


Simultaneously solving :
    v ^ 2 = 0 ^ 2 + 5 * (60 * 60) ^ 2 / (1760 * 3) * 2000 / 1760
For [v].
Solving v ^ 2 = 0 ^ 2 + 5 * (60 * 60) ^ 2 / (1760 * 3) * 2000 / 1760 for v

    v = (1687500/121) ^ (1/2) # v = -1 * (1687500/121) ^ (1/2)
        (by Isolation)

Answer is :
( X1 # X2 )
  where :
    X1 =  v = (1687500/121) ^ (1/2)
    X2 =  v = (1687500/121) ^ (1/2) * -1

Applying substitution
    v = (1687500/121) ^ (1/2)
```

```
  to    :

  sives :

Applysins substitution
    v = (1687500/121) ^ (1/2) * -1

  to    :

  sives :

Substitutins back in v solution
Applysins substitution

  to    :
    v = (1687500/121) ^ (1/2)

  sives :
    v = (1687500/121) ^ (1/2)

S  stitutins back in v solution
Ap-plysins substitution

  to    :
    v = (1687500/121) ^ (1/2) * -1

  sives :
    v = (1687500/121) ^ (1/2) * -1


Final Answers are :
( X1 # X2 )
  where :
    X1 =  v = (1687500/121) ^ (1/2)
    X2 =  v = (1687500/121) ^ (1/2) * -1

nl4 took 976 milliseconds and produced answer
( X1 # X2 )
  where :
    X1 =  v = (1687500/121) ^ (1/2)
    X2 =  v = (1687500/121) ^ (1/2) * -1


Simultaneously solvins :
    m1 * s * cos(270) + (1 * tsn + (cos(-270) * reaction1 + 1 * mu * reaction1 +
    m2 * s * 1 + (cos(180) * tsn + 0) + 0 = m2 * (a1 * 1)
    m1 * s * 1 + (cos(270) * tsn + (reaction1 + cos(270) * mu * reaction1 + 0)) +
For [reaction1, tsn, a1].
Solvins m1 * s * cos(270) + (1 * tsn + (cos(-270) * reaction1 + 1 * mu * reaction

    mu * reaction1 = m1 * a1 + -1 * tsn
        (by Isolation)
Solvins m1 * s * 1 + (cos(270) * tsn + (reaction1 + cos(270) * mu * reaction1 + 0

    reaction1 = 0 + -1 * (m1 * s)
        (by Isolation)

Answer is :
X1
  where :
```

```
    X1 =  reaction1 = m1 * s * -1

Applying substitution
    reaction1 = m1 * s * -1

  to    :
    m1 * s * cos(270) + (1 * tsn + (cos(-270) * reaction1 + 1 * mu * reaction1 + (
    m2 * s * 1 + (cos(180) * tsn + 0) + 0 = m2 * (a1 * 1)

  gives :
    tsn + mu * m1 * s * -1 = m1 * a1
    m2 * s + tsn * -1 = m2 * a1

Solving tsn + mu * m1 * s * -1 = m1 * a1 for tsn

    tsn = m1 * a1 + -1 * (mu * m1 * s * -1)
        (by Isolation)

Answer is :
X1
    ere :
    X1 =  tsn = (a1 + mu * s) * m1

Applying substitution
    tsn = (a1 + mu * s) * m1

  to    :
    m2 * s + tsn * -1 = m2 * a1

  gives :
    m2 * s + (a1 + mu * s) * m1 * -1 = m2 * a1

Solving m2 * s + (a1 + mu * s) * m1 * -1 = m2 * a1 for a1
 a1 = (mu * m1 * -1 + m2) * s * (m1 * -1 + m2 * -1) ^ -1 * -1 is a solution

Answer is :
X1
  where :
    X1 =  a1 = mu * s * (m1 * -1 + m2 * -1) ^ -1 * m1 + s * (m1 * -1 + m2 * -1) ^

Substituting back in tsn solution
Applying substitution
    a1 = mu * s * (m1 * -1 + m2 * -1) ^ -1 * m1 + s * (m1 * -1 + m2 * -1) ^ -1 * m

  to    :
    tsn = (a1 + mu * s) * m1

  gives :
    tsn = (mu * s * (m1 * -1 + m2 * -1) ^ -1 * m1 + s * (m1 * -1 + m2 * -1) ^ -1 *

Substituting back in reaction1 solution
Applying substitution
    a1 = mu * s * (m1 * -1 + m2 * -1) ^ -1 * m1 + s * (m1 * -1 + m2 * -1) ^ -1 * m
    tsn = (mu * s * (m1 * -1 + m2 * -1) ^ -1 * m1 + s * (m1 * -1 + m2 * -1) ^ -1 *

  to    :
    reaction1 = m1 * s * -1

  gives :
    reaction1 = m1 * s * -1
```

```
Final Answers are :
( ( X1 & X2 ) & X3 )
  where :
    X1 =  a1 = mu * g * (m1 * -1 + m2 * -1) ^ -1 * m1 + g * (m1 * -1 + m2 * -1) ^
    X2 =  tsn = (mu * g * (m1 * -1 + m2 * -1) ^ -1 * m1 + g * (m1 * -1 + m2 * -1)
    X3 =  reaction1 = m1 * g * -1

Fulltab took 3118 milliseconds and produced answer
( ( X1 & X2 ) & X3 )
  where :
    X1 =  a1 = mu * g * (m1 * -1 + m2 * -1) ^ -1 * m1 + g * (m1 * -1 + m2 * -1) ^
    X2 =  tsn = (mu * g * (m1 * -1 + m2 * -1) ^ -1 * m1 + g * (m1 * -1 + m2 * -1)
    X3 =  reaction1 = m1 * g * -1


Trying to solve
    x > 1 / (1 + sin(y) ^ 2)

S   ing x > (sin(y) ^ 2 + 1) ^ -1 for x

Answer is :
X1
  where :
    X1 =  x > (sin(y) ^ 2 + 1) ^ -1

Isolating x on the lhs gives
    x > (sin(y) ^ 2 + 1) ^ -1

x > (sin(y) ^ 2 + 1) ^ -1 dominates the other inequalities.

stvineq took 180 milliseconds and produced answer
X1
  where :
    X1 =  x > (sin(y) ^ 2 + 1) ^ -1


Simultaneously solving :
    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
For [t0, t1, t2, t3, d2, d1, d3],
Solving t0 = t1 + (t2 + (t3 + 0)) for t0

Answer is :
X1
  where :
    X1 =  t0 = t1 + t2 + t3

Applying substitution
    t0 = t1 + t2 + t3

  to    :
    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

  gives :
    (3/4) = t1 * (1/7200)
    t2 * (3/4) = d2
    0 = t3 * (-1/1800) + (3/4)
    7 = d1 + d2 + d3
    d1 = t1 ^ 2 * (1/14400)
    d3 = t3 * (3/4) + t3 ^ 2 * (-1/3600)

Solving (3/4) = t1 * (1/7200) for t1

    t1 = (3/4) * 7200
        (by Isolation)

Answer is :
X1
  where :
    X1 =  t1 = 5400

Applying substitution
    t1 = 5400

  to    :
    d1 = t1 ^ 2 * (1/14400)
    t2 * (3/4) = d2
    0 = t3 * (-1/1800) + (3/4)
    7 = d1 + d2 + d3
    d3 = t3 * (3/4) + t3 ^ 2 * (-1/3600)

  gives :
    d1 = 2025
    t2 * (3/4) = d2
    0 = t3 * (-1/1800) + (3/4)
    7 = d1 + d2 + d3
    d3 = t3 * (3/4) + t3 ^ 2 * (-1/3600)

Solving t2 * (3/4) = d2 for t2

    t2 = d2 * (4/3)
        (by Isolation)

Answer is :
X1
  where :
    X1 =  t2 = d2 * (4/3)

Applying substitution
    t2 = d2 * (4/3)

  to    :
    d1 = 2025
    0 = t3 * (-1/1800) + (3/4)
    7 = d1 + d2 + d3
    d3 = t3 * (3/4) + t3 ^ 2 * (-1/3600)

  gives :
```

```
    d1 = 2025
    0 = t3 * (-1/1800) + (3/4)
    7 = d1 + d2 + d3
    d3 = t3 * (3/4) + t3 ^ 2 * (-1/3600)

Solving 0 = t3 * (-1/1800) + (3/4) for t3

    t3 * (-1/1800) = 0 + -1 * (3/4)
        (by Isolation)

    t3 = (0 + -1 * (3/4)) * -1800
        (by Isolation)

Answer is :
X1
  where :
    X1 =  t3 = 1350

Applying substitution
    t3 = 1350

  to    :
    d3 = t3 * (3/4) + t3 ^ 2 * (-1/3600)
    d1 = 2025
    7 = d1 + d2 + d3

  gives :
    d3 = (2025/4)
    d1 = 2025
    7 = d1 + d2 + d3

Solving 7 = d1 + d2 + d3 for d2

    d1 + d2 = 7 + -1 * d3
        (by Isolation)

    d2 = 7 + -1 * d3 + -1 * d1
        (by Isolation)

Answer is :
X1
  where :
    X1 =  d2 = d3 * -1 + d1 * -1 + 7

Applying substitution
    d2 = d3 * -1 + d1 * -1 + 7

  to    :
    d3 = (2025/4)
    d1 = 2025

  gives :
    d3 = (2025/4)
    d1 = 2025

Solving d1 = 2025 for d1

Answer is :
X1
  where :
```

```
    X1 =  d1 = 2025

Applying substitution
    d1 = 2025

  to    :
    d3 = (2025/4)

  gives :
    d3 = (2025/4)

Solving d3 = (2025/4) for d3

Answer is :
X1
  where :
    X1 =  d3 = (2025/4)

Substituting back in d1 solution
Applying substitution
    d3 = (2025/4)

  to    :
    d1 = 2025

  gives :
    d1 = 2025

Substituting back in d2 solution
Applying substitution
    d3 = (2025/4)
    d1 = 2025

  to    :
    d2 = d3 * -1 + d1 * -1 + 7

  gives :
    d2 = (-10097/4)

Substituting back in t3 solution
Applying substitution
    d3 = (2025/4) & d1 = 2025
    d2 = (-10097/4)

  to    :
    t3 = 1350

  gives :
    t3 = 1350

Substituting back in t2 solution
Applying substitution
    (d3 = (2025/4) & d1 = 2025) & d2 = (-10097/4)
    t3 = 1350

  to    :
    t2 = d2 * (4/3)

  gives :
    t2 = (-10097/3)
```

```
Substitutins back in t1 solution
Applyins substitution
    ((d3 = (2025/4) & d1 = 2025) & d2 = (-10097/4)) & t3 = 1350
    t2 = (-10097/3)

  to    :
    t1 = 5400

  sives :
    t1 = 5400

Substitutins back in t0 solution
Applyins substitution
    (((d3 = (2025/4) & d1 = 2025) & d2 = (-10097/4)) & t3 = 1350) & t2 = (-10097/3
    t1 = 5400

  to    :
    t0 = t1 + t2 + t3

  ves :
    t0 = (10153/3)


Final Answers are :
( ( ( ( ( ( X1 & X2 ) & X3 ) & X4 ) & X5 ) & X6 ) & X7 )
  where :
    X1 =  d3 = (2025/4)
    X2 =  d1 = 2025
    X3 =  d2 = (-10097/4)
    X4 =  t3 = 1350
    X5 =  t2 = (-10097/3)
    X6 =  t1 = 5400
    X7 =  t0 = (10153/3)

train took 2499 milliseconds and produced answer
( ( ( ( ( ( X1 & X2 ) & X3 ) & X4 ) & X5 ) & X6 ) & X7 )
  where :
    X1 =  d3 = (2025/4)
    X2 =  d1 = 2025
    X3 =  d2 = (-10097/4)
    X4 =  t3 = 1350
    X5 =  t2 = (-10097/3)
    X6 =  t1 = 5400
    X7 =  t0 = (10153/3)


Solvins 2 ^ (2 * x + 8) - 32 * 2 ^ x + 1 = 0 for x

Homosenized eauation is (2 ^ x) ^ 2 * 2 ^ 8 + 2 ^ x * -32 = -1

New eauation is x3 ^ 2 * 256 + x3 * -32 = -1

Usins auadratic equation formula
 x3 = (1/16) # x3 = (1/16) is a solution
Applyins substitution
    x3 = 2 ^ x

  to    :
    x3 = (1/16) # x3 = (1/16)
```

```
  sives :
    2 ^ x = (1/16)


    x = los(2, (1/16))
        (by Isolation)

Answer is :
X1
  where :
    X1 =  x = -4


Pow2ean took 1584 milliseconds and Produced answer
X1
  where :
    X1 =  x = -4


S[illegible]ins 12 * x ^ 4 - 56 * x ^ 3 + 89 * x ^ 2 - 56 * x + 12 = 0 for x

Substitutins x4 for x + 1 / x
 sives x4 * -56 + x4 ^ 2 * 12 + 65 = 0

Usins quadratic equation formula
 x4 = (5/2) # x4 = (13/6) is a solution
Applyins substitution
    x4 = x + 1 / x

  to    :
    x4 = (5/2) # x4 = (13/6)

  sives :
    x + x ^ -1 = (5/2) # x + x ^ -1 = (13/6)


Multiply throush by x ^ 1 to set a polynomial

U[illegible]s quadratic equation formula
 x = 2 # x = (1/2) is a solution

Multiply throush by x ^ 1 to set a polynomial

Usins quadratic equation formula
 x = (3/2) # x = (2/3) is a solution
 (x = 2 # x = (1/2)) # x = (3/2) # x = (2/3) is a solution

Answer is :
( ( X1 # X2 ) # ( X3 # X4 ) )
  where :
    X1 =  x = 2
    X2 =  x = (1/2)
    X3 =  x = (3/2)
    X4 =  x = (2/3)


quarteen took 1215 milliseconds and Produced answer
( ( X1 # X2 ) # ( X3 # X4 ) )
  where :
```

```
    X1 =  x = 2
    X2 =  x = (1/2)
    X3 =  x = (3/2)
    X4 =  x = (2/3)



yes
| ?- halt.
Utilities Package  (8 Jul 81)
[Now includes LONG and TIDY]

| ?-  :- [filin].

extras:words compiled: 654 words,     0.85 sec.

extras:sportr compiled: 580 words,     0.85 sec.

press:solve consulted   921 words      0.34 sec.

press:isolat consulted   300 words      0.13 sec.

press:isolat.ax consulted   1674 words      0.67 sec.

press:ineqis.ax consulted   756 words      0.34 sec.

press:weaknf consulted   375 words      0.15 sec.

press:collec consulted   186 words      0.06 sec.

press:nasl consulted   286 words      0.15 sec.

press:collec.ax consulted   771 words      0.36 sec.

press:attrac consulted   1667 words      0.21 sec.

press:attrac.ax consulted   321 words      0.13 sec.

press:chunk consulted   332 words      0.14 sec.

press:homos.top consulted   1278 words      0.51 sec.

press:homos.rew consulted   2219 words      0.98 sec.

press:homos.trs consulted   3057 words      1.28 sec.

press:homos.msc consulted   2302 words      1.03 sec.

press:nasty consulted   195 words      0.09 sec.

press:simeq consulted   426 words      0.20 sec.

press:ineq consulted   650 words      0.25 sec.

press:misc consulted   508 words      0.21 sec.

press:match consulted   1655 words      0.70 sec.

press:prover consulted   558 words      0.24 sec.
```

SRRWT

```
integrate(Term,Term*X,X):-freeof(Term,X),!.


integrate(Term,Soln,X)
        :-simplify(Term,NTerm),
          integrate1(NTerm,Ans,X),
          simplify(Ans,Soln).


integrate1(Y+Z,YInt+ZInt,X)
        :-!,
        integrate(Y,YInt,X),
        integrate(Z,ZInt,X).



integrate1(Term,Soln,X)
            :-derivdivides(Term,Soln,X),
            !.



derivdivides(Term,Soln,X)
        :-decompl(Term,[*,..Elements]),
          ddivides(Elements,[],Soln,X).

derivdivides(Term,Const*2:(-1)*(Fn:2),X)
       :-match(Term,Fn*Rest),
         decompl(Fn,[*,..Elements]),
         decompl(Rest,[*,..RestBas]),
         check(Fn,RestBas,Const,X).



ddivides([El,..Rest],Scanned,Const*Ans,X)
        :-lookup(El,Ars,Ans),
          append(Rest,Scanned,Restofterm),
          check(Ars,Restofterm,Const,X).

 ividesCCEl,..Rest],Scanned,Ans,X)
        :-ddivides(Rest,[El,..Scanned],Ans,X).


check(Ux,Remainder,Const,X)
        :-diffwrt(Ux,DUx,X),
          simplify(DUx,NewDUx),
          decompl(NewDUx,[*,..DUxBas]),
          ksame(DUxBas,Remainder,Constbas,X),
          recompl(Const,[*,..Constbas]).


ksame(L1,L2,Const,X)
        :-freeof(L1,X),
          freeof(L2,X),
          specapp(L1,L2,Const),
          !.

ksame([FL1,..RestL1],L2,Const,X)
        :-select(FL1,L2,NewL2),
```

```
            ksame(RestL1,NewL2,Const,X),
            !.

ksame([FL1|RestL1],L2,Const,X)
          :-freeof(FL1,X),
            not freeof(RestL1,X),
            append(RestL1,[FL1],NewL1),
            ksame(NewL1,L2,Const,X),
            !.

specapp(L1,L2,Ans)
          :-inverse(L1,NewL1),
            append(NewL1,L2,Ans).


inverse([],[]).

inverse([FList|RList],[FList^(-1)|RNes])
            :-inverse(RList,RNes).



lookup(cos(Ars),Ars,sin(Ars)).
lookup(sin(Ars),Ars,-cos(Ars)).
lookup(tan(Ars),Ars,log(sec(Ars))).
lookup(cot(Ars),Ars,log(sin(Ars))).
lookup(sec(Ars),Ars,log(sec(Ars)+tan(Ars))).
lookup(artan(Ars),Ars,Ars*artan(Ars)-log(1+Ars^2)*2^(-1)).
lookup(arsin(Ars),Ars,Ars*arsin(Ars)+(1-Ars^2)^2^(-1)).
lookup(log(Ars),Ars,Ars*log(Ars)-Ars).
lookup(Ars^(-1),Ars,log(Ars)).
lookup(Ars^D,Ars,(D+1)^(-1)*Ars^(D+1)):-freeof(D,x),
                                        D\==(-1).
lookup(D^Ars,Ars,(log(D)^(-1))*D^Ars):-freeof(D,x),
                                        D\==(-1).


decomp1(X*(Y*E),L)
          :-!,decomp1(X*Y*E,L).
  comp1(E*X*Y,[*,Y,..L]):-!,decomp1(E*X,[*,..L]).
decomp1(Y*E,[*,Y,E]):-!.
decomp1(X,[*,X]).



recomp1(E,[*,E]):-!.
recomp1(E*X,[*,X,..L]):-!,recomp1(E,[*,..L]).
recomp1(1,[*]):-!.
recomp1(X,[*,X]).



diffwrt1(C^X,C^X*log(C)^(-1),X):-freeof(C,X),!.
diffwrt1(log(X),X^(-1),X):-!.
diffwrt1(tan(X),sec(X)^2,X):-!.
diffwrt1(cot(X),-cosec(X)^2,X):-!.
diffwrt1(sec(X),sec(X)*tan(X),X):-!.
diffwrt1(arsin(X),(1-X^2)^(-2^(-1)),X):-!.
diffwrt1(cosec(X),-cos(X)*cosec(X)^2,X):-!.
diffwrt1(artan(X),(1+X^2)^(-1),X):-!.
```

```
tsimpax(-U,(-1)*U).
tsimpax(log(e),1).

sensym(var,var1).
```

Old stuff. Replaced by TIDY & LONG

```
/* Evaluation routines */


  eval(Exp,Exp) :- atomic(Exp), !.

  eval(Exp,Ans)
        :- nonvar(Exp),
           Exp =.. [P|Args],
           maplist(eval,Args,Nargs),
           Nexp =.. [P|Nargs],
           ( checklist(integer,Nargs) -> eval1(Nexp,Ans)
                                      ;  Ans = Nexp       ),
           !.


  eval1(A=B,true) :- A=:=B, !.

  eval1(A=B,false) :- A=\=B, !.

  eval1(A=:=B,Ans) :- eval1(A=B), !.

  eval1(A=\=B,true) :- A=\=B, !.

  eval1(A=\=B,false) :- A=:=B, !.

  eval1(A>B,true) :- A>B, !.

  eval1(A>B,false) :- A=<B, !.

  eval1(A<B,Ans) :- eval1(B>A,Ans), !.

  eval1(A>=B,Ans) :- eval1(B>A,Ans1), negate(Ans1,Ans), !.

  eval1(A=<B,Ans) :- eval1(B>=A,Ans), !.

  eval1(A+B,Ans) :- Ans is A+B, !.

  eval1(A-B,Ans) :- Ans is A-B, !.

  eval1(A*B,Ans) :- Ans is A*B, !.

  eval1(A/B,Ans) :- 0 is A mod B, !, Ans is A/B.

  eval1(A/B,Ans^(-1)) :- 0 is B mod A, !, Ans is B/A.

  eval1(A/B,A1/B1) :-
        gcd(A,B,D), !, A1 is A/D, B1 is B/D.

  eval1(-A,Ans) :- Ans is -A, !.

  eval1(A ++ B,Ans) :- Res is A+B, Ans is Res mod 360, !.

  eval1(A -- B,Ans) :- Res is A-B, Ans is Res mod 360, !.

  eval1(A^B,Ans) :- intexp(A,B,Ans), !.

  eval1(sort(X),Ans) :- sorteval(X,Ans), !.
```

```
  eval1(sin(X),Ans) :- sineval(X,Ans), !.

  eval1(cos(X),Ans) :- coseval(X,Ans), !.

  eval1(tan(X),Ans) :- taneval(X,Ans), !.

  eval1(arcsin(X),Ans) :- arcsineval(X,Ans), !.

  eval1(arccos(X),Ans) :- arccoseval(X,Ans), !.

  eval1(arctan(X),Ans) :- arctaneval(X,Ans), !.


  eval1(Exp,Exp).


  negate(true,false).
  negate(false,true).


  /* Greatest Common Divisor */

gcd(A,0,A) :- !.
gcd(A,B,D) :- !,
        C is A mod B,
        gcd(B,C,D).

  sarteval(4,2).
  sarteval(9,3).
  sarteval(16,4).
  sarteval(25,5).
  sarteval(36,6).
  sarteval(49,7).
  sarteval(64,8).
  sarteval(81,9).
  sarteval(100,10).
  sarteval(121,11).
  sarteval(144,12).
  sarteval(169,13).


  intexp(-1,-1,-1) :- !.

  intexp(L,M,-1*(N^M))
        :- L < 0,
           M < 0,
           P is -M,
           odd(P),
           N is -L,
           !.

  intexp(L,M,N^M)
        :- L < 0,
           M < 0,
           N is -L,
           !.

  intexp(L,M,L^M) :- M < 0, !.
```

```
intexp(L,1,L) :- !.

intexp(L,M,N)
      :- P is M-1,
         intexp(L,P,Q),
         N is L * Q.



sineval(X,S)
      :- X =< 90,
         X >= 0,
         !,
         sineval1(X,S).

sineval(X,S)
      :- X > 90,
         !,
         Y is 180-X,
         sineval(Y,S).

sineval(X,S)
      :- X < 0,
         !,
         Y is -X,
         sineval(Y,S1),
         eval(-1*S1,S).


  sineval1(0,0).

  sineval1(30,2^(-1)).

  sineval1(45,(2^(2^(-1)))^(-1)).

  sineval1(60,(3^(2^(-1)))*2^(-1)).

  sineval1(90,1).



coseval(X,C)
      :- Y is 90-X,
         sineval(Y,C).



taneval(X,T)
      :- sineval(X,S),
         coseval(X,C),
         eval(S*C^(-1),T).


arcsineval(S,X)
      :- non_neg(S),
         !,
         sineval1(X,S).

arcsineval(S,X)
```

```
        :- negative(S),
           eval(-1*S,S1),
           sineval1(X,S1).


arccoseval(C,X)
        :- non_neg(C),
           !,
           sineval1(Y,C),
           X is 90-Y.

arccoseval(C,X)
        :- negative(C),
           eval(-1*C,C1),
           sineval1(Y,C1),
           X is 90-Y.


arctaneval(T,X)
        :- tidy(T*((1+T^2)^(2^(-1)))^(-1),S),
           arcsineval(S,X).


natnum(X) :- integer(X), X>0.


odd(N) :- eval(N,N1), natnum(N1), 1 is N1 mod 2.

even(N) :- eval(N,N1), natnum(N1), 0 is N1 mod 2.



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



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

wordsin([HD|TL],List)
        :- wordsin(HD,L1),
           wordsin(TL,L2),
           union(L1,L2,List),
           !.

wordsin(E,List)
        :- atomic(E) -> fail
                     ;  E =.. [P|Args],
                        wordsin(Args,List),
                        !.

wordsin(E,[])
        :- integer(E),
           !.

wordsin(E,[E]) :- atom(E).
```

OLD stuff - see INT

```
/*type*/
/*Finds types of variables in PRESS*/
/*Alan Bundy 19.12.79*/


/* Use type information - top level*/

in(X,[L,B,T,R]) :-
        interval(X,[Lx,Bx,Tx,Rx]), !,
        less_than([B,L1],[Bx,Lx]), less_than([Tx,Rx],[T,R1]),
        opposite(L1,L), opposite(R1,R).

positive(X) :- interval(X,[L,B,T,R]), !, less_than([0,closed],[B,L]).

negative(X) :- interval(X,[L,B,T,R]), !, less_than([T,R],[0,closed]).

non_neg(X) :- interval(X,[L,B,T,R]), !, less_than([0,open],[B,L]).

non_zero(X^N) :- negative(N), !.          /*temporary patch*/

   n_zero(X) :- interval(X,[L,B,T,R]), !,
        (less_than([0,closed],[B,L]); less_than([T,R],[0,closed])).

acute(X) :- interval(X,[L,B,T,R]), !,
        less_than([0,open],[B,L]), less_than([T,R],[90,open]).

obtuse(X) :- interval(X,[L,B,T,R]), !,
        less_than([90,open],[B,L]), less_than([T,R],[180,open]).

non_reflex(X) :- interval(X,[L,B,T,R]), !,
        less_than([0,open],[B,L]), less_than([T,R],[180,open]).


/* X lies in in closed or open interval*/

interval(X+Y,[L,B,T,R]) :- !,
        interval(X,[Lx,Bx,Tx,Rx]), interval(Y,[Ly,By,Ty,Ry]),
        plus(Bx,By,B), plus(Tx,Ty,T),
        comb(Lx,Ly,L), comb(Rx,Ry,R).

/* This is general case. When we have real numbers we can
use it for any function monotonic on an interval*/
interval(X*Y,[L,B,T,R]) :- !,
        interval(X,[Lx,Bx,Tx,Rx]), interval(Y,[Ly,By,Ty,Ry]),
        mlmaplist(mult,[[Bx,Bx,Tx,Tx], [Lx,Lx,Rx,Rx],
                        [By,Ty,By,Ty], [Ly,Ry,Ly,Ry], AnsList]),
        pick_lower(AnsList,[B,L]), pick_upper(AnsList,[T,R]).

interval(X^Y,[Lx,1,inf,Rx]) :-
        in(X,[Lx,1,inf,Rx]), in(Y,[Ly,0,inf,Ry]), !.

interval(X^Y,[Rx,1,inf,Lx]) :-
        in(X,[Lx,0,1,Rx]), in(Y,[Ly,-inf,0,Ry]), !.

interval(X^Y,[Lx,0,1,Rx]) :-
        in(X,[Lx,0,1,Rx]), in(Y,[Ly,0,inf,Ry]), !.

interval(X^Y,[Rx,0,1,Lx]) :-
        in(X,[Lx,1,inf,Rx]), in(Y,[Ly,-inf,0,Ry]), !.
```

```
interval(sin(X),[L,0,1,R]) :-
        (in(X,[L,0,90,R]) ; in(X,[R,90,180,L]) ), !.


interval(sin(X),[closed,-1,1,closed]) :- !.

interval(cos(X),[L,0,1,R]) :-
        ( in(X,[R,0,90,L]); in(X,[L,270,360,R]) ), !.

interval(cos(X),[closed,-1,1,closed]) :- !.

interval(tan(X),[L,0,inf,R]) :-
        ( in(X,[L,0,90,R]); in(X,[L,180,270,R]) ), !.

/*integers and fractions have fixed range*/
interval(P/Q,[open,N,N1,open]) :-
        integer(P), integer(Q), !,
        whole_part(P,Q,N), N1 is N+1.

interval(X,[closed,X,X,closed]) :- integer(X), !.

/* Range known from type of curve*/
interval(X,Interval) :- atom(X), classify(X,Interval), !.

/* ad hoc patch for gravity - proper solution means allowing
equations between quantities and defining g as measure(g,32,ft/sec^2) */
interval(g,[open,1,inf,open]) :- !.

/* All quantities assumed positive (NB change defn of drop!!) */
interval(M,[open,0,inf,open]) :- measure(Q,M), quantity(Q), !,
        (said(M) -> true;
        assert(said(M)) & trace('I assume %t positive.\n',[M],1)).

/* Default case */
interval(X,[open,-inf,inf,open]) :- !.

/* Find lower bounding integer of P/Q */
whole_part(P,Q,N) :-
        M is P/Q,
        (M<0 -> N is M-1; N=M).

/* Add 2 boundaries*/
plus(X,Y,Z) :- integer(X), integer(Y), !, Z is X+Y.

plus(X,Y,Z) :- either_infinite(X,Y,Z), !.

/* minus a boundary */
minus(X,Y) :- integer(X), !, Y is -X.
minus(inf,-inf).
minus(-inf,inf).

/*multiply 2 boundaries*/
mult([B1,M1,B2,M2, [B,M]]) :-
        mult(B1,B2,B), comb(M1,M2,M).

mult(0,Y,0) :- !.                mult(X,0,0) :- !.

mult(X,Y,Z) :- integer(X),integer(Y), !, Z is X*Y.

mult(X,Y,Z) :- either_infinite(X,Y,_),
```

```
        sign(X,Sx), sign(Y,Sy), signed_inf(Sx,Sy,Z), !.


/* Orderins of boundaries (assumes ranses are consecutive)*/
less_than([X,Mx],[Y,My]) :-
        comb(Mx,My,M), less_than(X,Y,M), !.

less_than(X,Y,M) :-
        inteser(X), inteser(Y), !,
        (M=closed -> X<Y; X=<Y).

less_than(X,X,open) :- !.

less_than(X,inf,M) :- X \== inf, !.

less_than(-inf,Y,M) :- Y \== -inf, !.


/* Is either X or Y infinite*/
  ither_infinite(inf,Y,inf) :- !.
_ither_infinite(-inf,Y,-inf) :- !.
either_infinite(X,inf,inf) :- !.
either_infinite(X,-inf,-inf) :- !.



/* Combine 2 boundaries*/
comb(closed,closed,closed).
comb(closed,open,open).
comb(open,closed,open).
comb(open,open,open).

/*open and closed are opposites*/
opposite(open,closed).
opposite(closed,open).

/*sisn of number*/

  sn(-inf,'-').
_sn(inf,'+').
sisn(Int,'+') :- inteser(Int), Int>=0.
sisn(Int,'-') :- inteser(Int), Int<0.

/*produce inf of risht sisn*/
sisned_inf(S,S,inf) :- !.
sisned_inf(S1,S2,-inf).


/* Pick upper and lower bounds from list*/

pick_upper([Hd],Hd) :- !.

pick_upper([Hd|Tl],Ans) :-
        pick_upper(Tl,Up),
        (less_than(Up,Hd) -> Hd=Ans; Up=Ans), !.

pick_lower([Hd],Hd) :- !.

pick_lower([Hd|Tl],Ans) :-
```

```
        pick_lower(T1,Lwr),
        (less_than(Hd,Lwr) -> Hd=Ans; Lwr=Ans) , !.


/*Find range that angle lies in */
classify(Angle ,Range ) :- measure(Q ,Angle ),
        angle(Point ,Q ,Curve ), !, range(angle ,Curve ,Range ).

classify(Angle ,Range ) :- measure(Q ,Angle ),
        incline(Curve ,Q ,Point ), !, range(incline ,Curve ,Range ).

/*Find range from curve shape */
/*For simple curves */
range(AI ,Curve ,Range ) :-
        concavity(Curve ,Conv ), cnorm(Conv ,Nconv ),
        slope(Curve ,Slope ), snorm(Slope ,Nslope ), !,
        quad(AI ,Nslope ,Nconv ,Range ).

/*For complex curve */
range(AI ,Curve ,Range ) :-
        partition(Curve ,Clist ), !, maplist(range(AI) ,Clist ,Rlist ),
        unionlist(Rlist ,Range ).

/*Find range given concavity and slope */
quad(angle,left,right,[open,0,90,closed]) :- !.
quad(incline,left,right,[closed,90,180,closed]) :- !.
quad(angle,right,right,[closed,90,180,closed]) :- !.
quad(incline,right,right,[closed,180,270,closed]) :- !.
quad(angle,left,left,[closed,180,270,closed]) :- !.
quad(incline,left,left,[closed,270,360,closed]) :- !.
quad(angle,right,left,[closed,270,360,closed]) :- !.
quad(incline,right,left,[open,0,90,closed]) :- !.


/*normal forms for concavities and slopes */

/*slopes */
snorm(hor ,right ) :- !.
snorm(S,S) :- !.

/*concavities*/
cnorm(stline,left) :- !.
cnorm(C,C) :- !.

/*Union of list of intervals*/

/*basis*/
unionlist([Range],Range) :- !.

/*recusive step*/
unionlist([Range1 | Rest], Range) :-
        unionlist(Rest,Range2), !,
        (combine(Range1,Range2,Range) ; combine(Range2,Range1,Range)).

/*Combine two intervals, if its easy*/
combine([M1,N1,N2,M2],[M3,N3,N4,M4],[M1,N1,N4,M4]) :-
        N2>=N3, !.


/* JOBS TO DO
```

```
        improve as in note 62 i.e. use monotonic reasoning
        write symbolic version for finding max/mins
*/

:- true.
```

```
Start of save set MATCHER on
System  Unknown monitor  APR$0
Unknown BPI 9 track 18-Aug-81 23:27:57 BACKUP  tape format 1
Tape number  1

 ALGEB           2                 2-Feb-80
 APPLIC PL       13                8-Mar-80
 BAGS   PL       39               29-Jul-81
 BASIC           52               28-Feb-80
 CHANGE PL       15               29-Jul-81
 CMISCE PL       8                 8-Mar-80
 COLLAX PAT      8                11-Jul-81
 COLLEC PL       2                27-Jul-81
 DECOMP PAT      4                21-Apr-80
 EVAL            36               20-Feb-80
 EXAM            7                13-Mar-80
 EXPR   PL       38               29-Jul-81
 FEATUR PL       8                18-Apr-80
 FLAGRO PL       6                16-Apr-80
 FUZZY  PL       7                14-May-80
 `?ALS           45               20-Feb-80
 _)RD   PL       26               30-Apr-80
 IMISCE PL       8                 9-May-80
 INIT            2                28-Feb-80
 INST   PL       5                11-Jul-81
 INTERV PAT      2                10-Apr-80
 INTEXP PL       2                22-Feb-81
 INVOCA PL       15                8-Mar-80
 IOROUT PL       39                8-Mar-80
 ISOLAX PL       6                20-Apr-80
 LEARN  PL       23               21-Apr-80
 LEAST  PAT      3                 6-Jul-81
 LISTRO PL       17                8-Mar-80
 LOG    PL       7                21-Mar-80
 LTEST  PL       4                21-Apr-80
 MATCH           7                 6-Jul-81
 MEMO   PL       8                23-Feb-81
 MISC   PL       5                 6-Jul-81
 MLSUPP PL       6                 8-Mar-80
 `ULTIL PL       10                8-Mar-80
 _PS             2                 8-Mar-80
 PATHS  PL       5                26-Mar-80
 PICK   PL       19               13-May-80
 POLY   PAT      2                18-Mar-80
 PORTRA PL       7                23-Apr-80
 POWERF PL       41               29-Jul-81
 PRESS           140               3-Mar-80
 READIN PL       15                8-Mar-80
 RECORD PL       1                22-Feb-81
 SCOAP           52                5-Dec-79
 SETROU PL       13                8-Mar-80
 SIMP            55                3-Mar-80
 SPRINT          24               29-May-79
 SQRT   PL       7                20-Mar-80
 STRUCT PL       16                8-Mar-80
 TEST   PL       8                11-Jul-81
 TIDY   PL       4                11-Mar-80
 TRFORM PL       27               23-Apr-80
 TYPE            51               19-Feb-80
 UTIL            3                22-Feb-81
```

```
End of save set  on
System  Unknown monitor  APR$0
Unknown BPI 9 track 0M-Kt-Jan--41 00:00:00 BACKUP  tape format 1
Tape number  1

Start of save set PAPERS on
System  Unknown monitor  APR$0
Unknown BPI 9 track 18-Aus-81 23:28:25 BACKUP  tape format 1
Tape number  1

 CUBIC  MSS      43                18-Aus-81
 IJCAI  MSS      183               10-Jun-81
 MATCH  BIB      20                18-Aus-81
 REPORT MSS      400                4-Jun-81

End of save set  on
System  Unknown monitor  APR$0
Unknown BPI 9 track 0M-Kt-Jan--41 00:00:00 BACKUP  tape format 1
Tape number  1
```

```
/* FILE IN POWERFUL MATCHER */

/* new procedures */
 :- consult('powerf.pl'),
    consult('hard.pl'),
    consult('bags.pl'),
    consult('expr.pl'),
    consult('memo.pl'),
    consult('pick.pl'),
    consult('fuzzy.pl'),
    consult('trform.pl'),
    consult('portra.pl'),
    consult('misc.pl'),
    consult('featur.pl'),
    consult('inst.pl'),
    consult('test.pl').

/* stuff to learn how to solve specialized kinds of equations */
 :- consult('learn.pl'),
    consult('ltest.pl').

/* additions to PRESS */
 :- consult('collec.pl'),
    consult('change.pl'),
    consult('tidy.pl'),
    consult('isolax.pl'),
    consult('sort.pl').

/* patches to PRESS */
 :- reconsult('collax.pat'),
    reconsult('decomp.pat'),
    reconsult('interv.pat'),
    reconsult('least.pat'),
    reconsult('polv.pat').
```

```
/* POWERFUL ALGEBRAIC MATCHER */


try_hard_to_solve(Eqn,Unknown,Ans) :-
    /* solve the equation using powerful matcher */
    flag(try_hard,Old,true),
    solve(Eqn,Unknown,Ans),
    flag(try_hard,_,Old).

:- flag(try_hard,_,false).



/* COLLECTION ROUTINE THAT USES POWERFUL MATCHER */

try_hard_to_collect(X,Old,New1) :-
    trace('\ntrying to use powerful matcher to collect %p in %c\n',
      [X,Old],3),
    features(Old,X,FExpr),
    trace('features of expression are %p\n',[FExpr],4),
    trace('looking for a collection rule with matching features\n\n',4),
    /* select a collection axiom */
    collax(Us,LHS,RHS),
    /* bind all the rule variables to random atoms,
       guarding against accidental overlap with atoms in original expression */
    wordsin(Old,BadNames),
    instantiate(LHS,BadNames,PatternVars),
    /* choose a collection variable and work out features with respect to it */
    wordsin(Us,Ulist), member(U,Ulist),
    /* make sure the features of the expression and the rule match */
    match_features(Old,X,LHS,U,Subst,AddedPVars),
    append(AddedPVars,PatternVars,PatternVars1),
    /* apply substitution found in features match to rule
       and put in polynomial normal form */
    subst(Subst,LHS,LHS1), subst(Subst,RHS,RHS1),
    poly_form([X],LHS1,LHS2),
    /* prepare and apply rule */
    make_description(Old,X,[],[],Old_D),
    make_description(LHS2,X,PatternVars1,AddedPVars,LHS_D),
    make_description(RHS1,X,PatternVars1,AddedPVars,RHS_D),
    apply_rule( Old_D , rule(LHS_D,RHS_D) , New_D),
    expr(New_D,New),
    tidy(New,New1),
    trace('%p collected in %p gives  %c\n',[X,Old,New1],2).


/* MATCH FEATURES OF EXPRESSION AND RULE AND RETURN SUBSTITUTION */

/* features of expression and rule match simply with change in variable */
match_features(Expr, Expr_X, Rule, Rule_X, Rule_X=Expr_X, []) :-
    features(Expr,Expr_X,FExpr),  features(Rule,Rule_X,FRule),
    subst(Rule_X=Expr_X, FRule, NewRule),
    match(NewRule, FExpr),
    !.

/* try change of unknown ... the initial attempts may fail, but leave
    advice on extra factors or terms to include in change of unknown */
match_features(Expr, Expr_X, Rule, Rule_X, Subst1&Subst2, AddedPatternVars) :-
    features(Expr,Expr_X,FExpr),  features(Rule,Rule_X,FRule),
```

```
    subterm(Term,FRule),
    diff(Term,FRule), diff(Term,Rule_X),
    contains(Rule_X,Term), occ(Term,FRule,N), N>1,
    subst(Term=Expr_X, FRule, NewRule),
    /* make sure that the only occurrence of the unknown in the rule was in
       the given subexpression */
    freeof(Rule_X,NewRule),
    match(NewRule, FExpr),
    trace('trying change of unknown of the form %p\n', [Term=Expr_X], 2),
    !,   /* find actual substitution to try */
    set_advice(Expr, Expr_X, Rule, Rule_X, Term, Subst1, AddedPatternVars),
    /* find other substitution */
    assert(particular_solution(Rule_X)),
    solve(Subst1,Rule_X,Subst2),
    trace('actual change of unknown is %c\n',[Subst1&Subst2],2).


/* first try obvious substitution */
set_advice(Expr, Expr_X, Rule, Rule_X, Term, Term=Expr_X, []).

 : if this doesn't work, check for advice in data base */
set_advice(Expr, Expr_X, Rule, Rule_X, Term, New_Subst, AddedPatternVars) :-
    recorded( when_matching(Expr,Expr_X),
         try_new_substitution(Term=Expr_X , New_Subst , AddedPatternVars), ID),
    trace('\n\nfinding advice to try new substitution %p\n' , [New_Subst],4),
    trace('   in place of old substitution %p\n' , [Term=Expr_X],4).




/* APPLY_RULE */
/* The arguments to "apply_rule" are as follows:

     Expr - the expression being transformed
     Rule - the rewrite rule being applied
     New_Expr - the result of applying the rule to Expr
   The symbolic quantities in the expression and the rule are assumed
   to be standardized apart. */

apply_rule( Expr , rule(Pattern,Replacement) , New_Expr ) :-
    expr(Expr,EE), expr(Pattern,EP), expr(Replacement,ER),
    trace('trying to apply rule %p -> %p\n   to %p\n\n',[EP,ER,EE],4),
    match(Expr,Pattern,Transform),
    apply_transform(Transform,Replacement,New_Expr),
    !.


/* match is called as follows:
     match(Expr,Pattern,Transform)
        where
     Expr is the expression or subexpression being matched
     Pattern is the left hand side of the rule (or a subpart of it)
     Transform is returned - it is a transformation (functions to be applied,
        substitutions, and possibly change of unknown)
        that makes Expr=Pattern      */


/* SIMPLE CASES -- IMMEDIATE MATCH OR SIMPLE SUBSTITUTION */
match(Expr,Pattern,Transform) :-
```

```
    expr(Expr,E), expr(Pattern,E),
    null_transform(Transform),
    trace('trivially matching %p and %p\n',[E,E],4),
    !.

match(Expr,Pattern,Transform) :-
    expr(Pattern,Var),
    atom(Var),
    pattern_vars(Pattern,PatternVars),
    member(Var,PatternVars),
    expr(Expr,E),
    make_substitution_transform(Var=E,Transform),
    trace('matching %p and %p by using substitution\nreturning %p\n',
       [E,Var,Transform],4),
    !.



/* HARD MATCH - USE MEMO */
match(Expr,Pattern,Transform) :-
    memo( hard_match(Expr,Pattern,Transform) ),
    !.
```

```
/* PROCEDURES FOR NON-TRIVIAL MATCHES */

/* There are two ways of accomplishing a hard match:
      by matching subexpressions ("match1") -- this may involve
         base matches, and applying functions to the RHS of the rule;
      or by solving for a variable in the rule ("match2").
   The subgoals match1 and match2 are used to prevent backtracking
   among the cases of match1. */

hard_match(Expr,Pattern,Transform) :-
   match1(Expr,Pattern,Transform),
   expr(Expr,E), expr(Pattern,P),
   trace('match succeeded on expression %p and pattern %p\nreturning %p\n\n',
      [E,P,Transform],4).

hard_match(Expr,Pattern,Transform) :-
   expr(Expr,E), expr(Pattern,P),
   trace('match failed on %p and %p\n',[E,P],4),
   match2(Expr,Pattern,Transform),
   trace('solving for a variable succeeded in matching expression %p\n',
      [E],4),
   trace('   and pattern %p\nreturning %p\n\n',[P,Transform],4).



/* match1 procedures to convert to base for + and *  */

match1(Expr,Pattern,Transform) :-
   expr(Expr,E), expr(Pattern,P),
   ( E=_+_ ; P=_+_ ),
   !,
   convert_and_match(+,Expr,Pattern,Transform).

match1(Expr,Pattern,Transform) :-
   expr(Expr,E), expr(Pattern,P),
   ( E=_*_ ; P=_*_ ),
   !,
   convert_and_match(*,Expr,Pattern,Transform).


/* MATCHING OTHER KINDS OF FUNCTIONS */

match1(Expr,Pattern,Transform) :-
   expr(Expr,E), expr(Pattern,P),
   trace('trying to match expression %p and pattern %p\n',
      [E,P],4),
   /* Expr and Pattern must have the same functor */
   functor(E,F,N), functor(P,F,N),
   match_parts(Expr,Pattern,1,N,Transform).



match_parts(Expr,Pattern,J,N,Transform) :-
   J>N,
   null_transform(Transform),
   !.
```

```
match_parts(Expr,Pattern,J,N,T3) :-
    subpart(Expr,J,E1), subpart(Pattern,J,P1),
    match(E1,P1,T1),
    apply_transform(T1,Pattern,P2),
    J1 is J+1,
    match_parts(Expr,P2,J1,N,T2),
    concat_transforms(T1,T2,T3).



/* "MATCH2" -- SOLVING FOR A VARIABLE IN THE RULE */

match2(Expr,Pattern,_) :-
    /* don't allow solving for a variable at top level */
    owners(Expr,[]),
    !,
    fail.

match2(Expr,Pattern,Transform) :-
    pattern_var(Pattern,V),
    expr(Expr,E), expr(Pattern,P),
    contains(V,P),
    trace('\ntrying to solve for a variable\n',4),
    trace('calling equation solver to solve for %p in %p\n\n',[V,E=P],4),
    /* assert that a particular rather than a general solution
       for V is desired */
    assert(particular_solution(V)),
    solve(E=P,V,SS),
    or_to_list(SS,SList),
    select(V=Soln,SList,_),
    make_substitution_transform(V=Soln,T1),
    /* If the unknown in Pattern is the same as the unknown in Expr (i.e.
       there is no new unknown), then the solution must be free of the
       unknown; if solving for the new unknown then the solution must
       contain the unknown; otherwise, the solution must be free of both
       the unknown and the new unknown */
    unknown(Expr,UExpr), unknown(Pattern,UPattern),
    (UPattern=UExpr -> freeof(UExpr,Soln), Transform=T1 ;
     V=UPattern -> contains(UExpr,Soln), change_unknown(UExpr,T1,Transform) ;
     freeof(UExpr,Soln), freeof(UPattern,Soln), Transform=T1 ),
    trace('using solution %p\n',[V=Soln],4).
```

```
/* BAG PROCEDURES FOR POWERFUL ALGEBRAIC MATCHER */


convert_and_match(Op,Expr,Pattern,Transform) :-
    to_bas(Op,Expr,ExprBas),
    to_bas(Op,Pattern,PatternBas),
    bas_match(ExprBas,PatternBas,Transform).


/* TRIVIAL CASE - EMPTY BAGS */
bas_match(Expr,Pattern,Transform) :-
    expr( Expr , bas(_,[]) ),
    expr( Pattern , bas(_,[]) ),
    null_transform(Transform),
    !.


/* USE MEMO FOR OTHER CASES */
bas_match(Expr,Pattern,Transform) :-
    expr(Expr,E), expr(Pattern,P),
    E=bas(Op,_),
    (Op= + -> Name=plus ; Op= * -> Name=times ),
    trace('trying to match %t bags for expression %p\n',[Name,E],4),
    trace('    and pattern %p\n',[P],4),
    memo( bas_match1(Expr,Pattern,Transform) ).



/* TRY PICKING A TERM FROM EACH BAG AND MATCHING THESE TERMS */
bas_match1(Expr,Pattern,Transform) :-
    pick_terms(Expr,Pattern,E,P,ERest,PRest),
    trace(
'picking terms from expression & pattern bags and trying to match them\n',4),
    match(E,P,T1),
    apply_transform(T1,PRest,PR1),
    expr(PR1,E1),
    poly_form(E1,E2),    /* CROCK! */
    poly_form(E2,E3),
    new_expr(PR1,E3,PR2),
    (null_transform(T1) -> true ;
     expr(PR2,RR),
     trace('applying transform to remaining terms in pattern bag\n',4),
     trace('    yielding %p\n\n',[RR],4) ),
    bas_match(ERest,PR2,T2),
    concat_transforms(T1,T2,Transform).



/* IF THERE IS JUST A VARIABLE LEFT, TRY MAKING IT THE IDENTITY ELEMENT FOR
    THE BAG.  This may not work, so be prepared to backtrack. */
bas_match1(Expr,Pattern,Transform) :-
    expr(Expr,bas(Op,[])), expr(Pattern,bas(Op,[V])),
    atom(V),
    pattern_vars(Pattern,PatternVars), member(V,PatternVars),
    unknown(Pattern,PUnknown), V\==PUnknown,
    /* make sure this isn't a pattern variable added by the matcher.
       Otherwise we would be undoing the effect of introducing it */
    not added_pattern_var(Pattern,V),
    identity(Op,Ident),
```

```
    make_substitution_transform(V=Ident,Transform),
    trace('trying making %p the bas identity element %p\n',[V,Ident],4).


/* SEE IF THERE'S JUST A PATTERN VARIABLE IN THE PATTERN, AND RANDOM
    JUNK LEFT IN THE EXPRESSION THAT'S FREE OF THE UNKNOWN */
bas_match1(Expr,Pattern,Transform) :-
    expr(Expr,Bas), expr(Pattern,bas(Op,[V])),
    pattern_var(Pattern,V), not unknown(Pattern,V),
    /* make sure this isn't a pattern variable added by the matcher.
       Otherwise we would be undoing the effect of introducing it */
    not added_pattern_var(Pattern,V),
    unknown(Expr,EUnknown), freeof(EUnknown,Bas),
    from_bas(Expr,Junk), expr(Junk,J),
    make_substitution_transform(V=J,Transform),
    trace('substituting %p for %p\n',[J,V],4).



/* TRY ELIMINATING A TERM FROM EITHER THE EXPRESSION OR THE RULE BY MOVING
    IT OR ITS INVERSE TO THE OTHER SIDE OF THE RULE */
bas_match1(Expr,Pattern,Transform) :-
    perm2(Expr,Pattern,Try,_),
    select_term(Try,T,Rest),
    op_distributes(T),
    expr(T,TT),
    unknown(Expr,EUnknown), freeof(EUnknown,TT),
    unknown(Pattern,PUnknown), freeof(PUnknown,TT),
    trace('dealing with term %p\n', [TT],4),
    trace('   by applying a function to each side of the rule\n',4),
    expr(Try,bas(Op,_)),
    (Try=Expr ->
       make_function_transform(Op,TT,T1), !, bas_match(Rest,Pattern,T2) ;
       make_inv_function_transform(Op,TT,T1), !, bas_match(Expr,Rest,T2) ),
    concat_transforms(T1,T2,Transform).


/* TRY INVOKING SOLVE-FOR-VARIABLE MATCH */
bas_match1(Expr,Pattern,Transform) :-
    from_bas(Expr,E), from_bas(Pattern,P),
    memo( match2(E,P,Transform) ).



/* SEE IF HAVING ANOTHER PATTERN VARIABLE AROUND WOULD HELP */
/* Another pattern variable might be available by choosing a different
    change of unknown, if both bases contain the unknown, and otherwise
    are free of the unknown.  (Perhaps the matcher can
    add or multiply by a new pattern variable.)  Record some advice, and
    then fail. */

bas_match1(Expr,Pattern,Transform) :-
    pick_terms(Expr,Pattern,E,P,ERest,PRest),
    /* see if picked terms are both equal to the unknown */
    unknown(Expr,X), expr(E,X), expr(P,X),
    /* ... and that remaining stuff is free of the known */
    expr(ERest, EE), freeof(X,EE),
    expr(PRest, PP), freeof(X,PP),
    /* leave advice */
```

```
    top(Expr,ETop), top(Pattern,PTop),
    sensum(a,Q), EE=bas(Op,_),
    (Op= + -> New_Subst = (Old_Term=X+ -1*Q) ;
         Op= * -> New_Subst = (Old_Term=X*Q^ -1)),
    recordz( when_matching(ETop,X) ,
                  try_new_substitution(Old_Term=X,New_Subst,[Q]) , ID),
    trace('failing, but leaving advice to try substitution %p\n',[New_Subst],4),
    trace('    in place of old substitution %p\n',[Old_Term=X],4),
    trace('    when matching expression %p and rule %p\n',
         [ETop,PTop],4),
    fail.


/* FAILURE - OUTPUT A MESSAGE */
bas_match1(Expr,Pattern,Transform) :-
    expr(Expr,E), expr(Pattern,P),
    trace('bas match failed on %p and %p\n',[E,P],4),
    fail.
```

```
/* EXPRESSION DESCRIPTIONS

An expression description is a data structure for describing expressions
and subexpressions for the powerful matcher, along with some associated
access procedures.

data structure format:
   expr_description(Expr,Top,Unknown,PatternVars,AddedPatternVars,Owners)
      where
   Expr is the current expression
   Top is the entire expression tree, of which Expr is a subexpression
   Unknown is the current unknown
   PatternVars is a list of the pattern variables in Top
        (including AddedPatternVars)
   AddedPatternVars is a list of the pattern variables in Top introduced by
        the matcher when performing a change of unknown
   Owners is a list of owners of Expr (sort of like a path from the top)

*/


/* ACCESS TO PARTS */

expr( expr_description(Expr,Top,Unknown,PVars,AddedPVars,Owners) , Expr) :- !.

top( expr_description(Expr,Top,Unknown,PVars,AddedPVars,Owners) , Top) :- !.

unknown( expr_description(Expr,Top,Unknown,PVars,AddedPVars,Owners) ,
      Unknown) :- !.

symbols( expr_description(Expr,Top,Unknown,PVars,AddedPVars,Owners) , Symbols) :
   wordsin(Top,Symbols),
   !.

pattern_vars( expr_description(Expr,Top,Unknown,PVars,AddedPVars,Owners) ,
      PVars) :- !.

owners( expr_description(Expr,Top,Unknown,PVars,AddedPVars,Owners) , Owners) :-



/* REPLACING PARTS */

new_expr( expr_description(Expr,Top,Unknown,PVars,AddedPVars,Owners) ,
      New_Expr ,
      expr_description(New_Expr,Top,Unknown,PVars,AddedPVars,Owners) ) :- !.

new_owners( expr_description(Expr,Top,Unknown,PVars,AddedPVars,Owners) ,
      New_Owners ,
      expr_description(Expr,Top,Unknown,PVars,AddedPVars,New_Owners) ) :- !.

new_unknown( expr_description(Expr,Top,Unknown,PVars,AddedPVars,Owners) ,
      New_Unknown ,
      expr_description(Expr,Top,New_Unknown,PVars,AddedPVars,Owners) ) :- !.

new_pattern_vars( expr_description(Expr,Top,Unknown,PVars,AddedPVars,Owners) ,
      New_PVars ,
      expr_description(Expr,Top,Unknown,New_PVars,New_Added,Owners) ) :-
   /* remove non-existent vars from AddedPVars */
   intersect(AddedPVars,New_PVars,New_Added).
```

```
/* ROUTINE TO ADD A MATCHER-GENERATED PATTERN VARIABLE TO THE LIST */
add_pattern_var( expr_description(Expr,Top,Unknown,PVars,AddedPVars,Owners) ,
        Var ,
        expr_description(Expr,Top,Unknown,PVars,[Var|AddedPVars],Owners) ) :- !.


/* TEST IF SOMETHING'S A PATTERN VARIABLE,
   OR RETURN ONE NONDETERMINISTICALLY */
pattern_var( expr_description(Expr,Top,Unknown,PVars,AddedPVars,Owners) ,
        V ) :-
   member(V,PVars).

/* TEST IF SOMETHING'S A PATTERN VARIABLE ADDED BY THE MATCHER,
   OR RETURN ONE NONDETERMINISTICALLY */
added_pattern_var( expr_description(Expr,Top,Unknown,PVars,AddedPVars,Owners) ,
        V ) :-
   member(V,AddedPVars).

/* TEST FOR EXPR THAT'S AN EMPTY BAG */
 mpty( expr_description(bag(_,[]),Top,Unknown,PVars,AddedPVars,Owners) ).



/* MAKE A DESCRIPTION GIVEN AN EXPRESSION AND AN UNKNOWN */

make_description( Expr, Unknown , PVars , AddedPVars ,
        expr_description(Expr,Expr,Unknown,PVars,AddedPVars,[]) ) :- !.



subpart( expr_description(Expr,Top,Unknown,PVars,AddedPVars,Owners) , N ,
        expr_description(SubExpr,Top,Unknown,PVars,AddedPVars,NewOwners) ) :-
   ars(N,Expr,SubExpr),
   add_owner(Owners,Expr,NewOwners),
   !.


 % SELECT NONDETERMINISTICALLY A TERM FROM A BAG */

select_term(Expr,T,Rest) :-
   Expr = expr_description( E, Top,Unknown,PVars,AddedPVars,Owners),
   E = bag(Op,Args),
   select(A,Args,ARest),
   add_owner(Owners,E,New_Owners),
   T = expr_description( A, Top,Unknown,PVars,AddedPVars,New_Owners),
   Rest = expr_description( bag(Op,ARest),
 Top,Unknown,PVars,AddedPVars,New_Owners).



/* ROUTINES TO KEEP TRACK OF OWNERS
As the matcher is recursively called on expressions, it keeps track
of the enclosing expressions in a list of owners.  Each item in the list
is a pair such as pair(first,t), pair(other,t), or pair(first,sin).  "first"
or "other" indicates whether the term being considered is the first element
of a bag being matched. */
```

```
/* procedures for bass */
add_owner( [pair(_,Op)|Rest], bas(Op,_), [pair(other,Op)|Rest] ) :- !.
add_owner( Owners, bas(Op,_), [pair(first,Op)|Owners] ) :- !.

/* procedures for other expressions */
add_owner( Owners, Expr, [pair(first,Op)|Owners] ) :-
    Expr=..[Op|_],
    !.


op_distributes(Expr) :-
    owners(Expr,Owners),
    distributes_over_owners(Owners),
    !.



distributes_over_owners([]) :- !.

 distributes_over_owners([P]) :- !.

distributes_over_owners([P1,P2|Rest]) :-
    dist1(P1,P2),
    distributes_over_owners([P2|Rest]).


dist1( pair(_,Op1) , pair(first,Op2) ) :-
    distributes(Op1,Op2).

distributes(*,+).
```

```
/* PROCEDURE THAT REMEMBERS PREVIOUSLY COMPUTED RESULTS

   memo(Pred)
      Pred is the predicate beins evaluated */


memo(Pred) :-
   recorded(Pred,memo(Pred,Result),_),
   ( Result=fail ->
        trace('lookins up failure for\n   %p\n',[Pred],4), !, fail ;
     trace('lookins up result for\n   %p\n',[Pred],4) ).

memo(Pred) :-
   call(Pred),
   /* Record result in data base if not there already.
      If it is already there, fail and try for another answer.
      This check is necessary -- the predicate may have been called
      previously without all possible results (including the final fail) beins
      senerated and recorded.  In this case, the previously recorded results
      will be re-senerated before new ones.  Mumble mumble. */
   ( recorded(Pred,memo(Pred,Result),_) -> fail ;
        recordz(Pred,memo(Pred,true),_) ).

memo(Pred) :-
   /* all calls have failed -- record failure */
   recordz(Pred,memo(Pred,fail),_),
   !, fail.
```

```
/* AUXILIARY PROCEDURES FOR POWERFUL MATCHER
   SELECT BEST TERMS TO MATCH FROM BAGS */

pick_terms(Expr,Pattern,E,P,ERest,PRest) :-
   pick_term(Expr,E,ERest),
   features(E,EFeatures),
   select_term(Pattern,P,PRest),
   features(P,PFeatures),
   unknown(Expr,EUnknown), unknown(Pattern,PUnknown),
   fuzzy_match(EFeatures,PFeatures,EUnknown,PUnknown),
   /* extra check for polynomials -- check that powers are the same */
   expr(E,EE), expr(P,PP),
   power(EE,EUnknown,N1),
   power(PP,PUnknown,N2),
   (EUnknown=PUnknown, integer(N1), integer(N2) -> N1=N2 ; true),
   /* reject if the match is non-trivial and
      moving the terms to the other side would succeed */
   (PFeatures=mumble, op_distributes(E), not(match(EE,PP)) -> fail ; true).



/* PICK THE BEST TERM FROM A BAG TO TRY MATCHING NEXT */

pick_term(Expr,T,Rest) :-
   expr(Expr,E), unknown(Expr,Unknown),
   pick1(E,Unknown,TT,RR),
   owners(Expr,Owners),
   add_owner(Owners,E,New_Owners),
   new_expr(Expr,TT,T1), new_owners(T1,New_Owners,T),
   new_expr(Expr,RR,R1), new_owners(R1,New_Owners,Rest).



pick1( bag(Op,[Term]) , Unknown , Term , bag(Op,[]) ) :- !.

pick1( bag(Op,[T1,T2|Others]) , Unknown , Term , bag(Op,[TBad|Rest]) ) :-
   pick_from_pair(T1,T2,Unknown,TGood,TBad),
   pick1( bag(Op,[TGood|Others]) , Unknown , Term , bag(Op,Rest) ).


/* Crock to handle polynomials - Just pick term with unknown
   to highest power.  This also handles terms free of the unknown. */

pick_from_pair(T1,T2,Unknown,TGood,TBad) :-
   power(T1,Unknown,P1),
   power(T2,Unknown,P2),
   (integer(P1), integer(P2), P1<P2 ->
        TGood=T2, TBad=T1 ; TGood=T1, TBad=T2).


/* power(Term,Unknown,N) unifies N with the highest power to which
   unknown occurs in term if the unknown is to an integer power, or
   to "mumble" if to a non-integer power.
   All the cuts and junk are to prevent unwanted backtracking. */

power(Unknown,Unknown,N) :-
```

```
    !,
    N=1.

power(X,Unknown,N) :-
    atomic(X),
    !,
    N=0.

power(Unknown^N,Unknown,N1) :-
    !,
    (integer(N) -> N1=N ; N1=mumble).

power([],Unknown,N) :-
    !,
    N=0.

power([H|T],Unknown,N) :-
    !,
    power(H,Unknown,P1),
    power(T,Unknown,P2),
    (P1=mumble -> N=mumble; P2=mumble -> N=mumble;
     P1>P2 -> N=P1; N=P2).

power(Expr,Unknown,N) :-
    !,
    Expr=..[Op|Args],
    power(Args,Unknown,N).
```

```
/* FUZZY MATCHER FOR USE WITH 'FEATURE' STUFF */

fuzzy_match(Expr1,Expr2,Unknown,New_Unknown) :-
    New_Unknown=false,
    !,
    match(Expr1,Expr2),
    !.

fuzzy_match(Expr1,Expr2,Unknown,New_Unknown) :-
    freeof(New_Unknown,Expr1),
    freeof(New_Unknown,Expr2),
    !,
    match(Expr1,Expr2),
    !.

fuzzy_match(Expr1,Expr2,Unknown,New_Unknown) :-
    /* At this point, one and only one of the expr's should contain
       New_Unknown.  For fuzzy match, just see that the other
       expression contains Unknown */
    perm2(Expr1,Expr2,E1,E2),
    contains(New_Unknown,E2),
    contains(Unknown,E1),
    !.


/* "features" will retain integer powers -- make sure that the match
    "mumble" */

:- asserta(( match(I,M) :- integer(I), atom(M), M=mumble )).
:- asserta(( match(M,I) :- integer(I), atom(M), M=mumble )).
```

```
/* TRANSFORMS */

/* Transforms are data structures that represent functions, substitutions,
   and possibly a change of unknown to be applied to an expression.
   Format:
     transform(FunctionList,SubstitutionList,New_Unknown)
   New_Unknown is "false" if the unknown isn't being changed */


/* CREATING TRANSFORMS */

null_transform( transform([],[],false) ).

make_substitution_transform( S , transform([],[S],false) ).

make_function_transform( Op , Expr ,
      transform( [function(Op,Expr)] , [] , false) ).

make_inv_function_transform( + , Expr ,
      transform( [function(+,NExpr)] , [] , false) ) :-
   tidy(-1*Expr,NExpr).

make_inv_function_transform( * , Expr ,
      transform( [function(*,InvExpr)] , [] , false) ) :-
   tidy(Expr^ -1,InvExpr).



change_unknown( New_Unknown , transform(FList,SList,false) ,
      transform(FList,SList,New_Unknown) ).


apply_transform( transform(FList,SList,New_Unknown) , Descr1 , Descr6 ) :-
   expr(Descr1,E1),
   apply_functions(FList,E1,E2),
   new_expr(Descr1,E2,Descr2),
   /*remove pattern vars that have been substituted for */
   subst_vars(SList,SVars), pattern_vars(Descr2,PVars),
   subtract(PVars,SVars,New_PVars),
   apply_substitutions(SList,Descr2,Descr3),
   new_pattern_vars(Descr3,New_PVars,Descr4),
   expr(Descr4,E4),
   tidy(E4,E5),
   new_expr(Descr4,E5,Descr5),
   (New_Unknown=false -> Descr6=Descr5 ;
      new_unknown(Descr5,New_Unknown,Descr6) ),
   !.

concat_transforms( transform(F1,S1,U1) , transform(F2,S2,U2) ,
      transform(F3,S3,U3) ) :-
   append(F1,F2,F3), append(S1,S2,S3),
   (U1=false -> U3=U2 ; U2=false -> U3=U1 ; fail),
   !.



/*    apply_functions(Functions,Expr,New)
   takes a list of functions "Functions"  and an expression "Expr".
```

```
   Returns the result of applying the functions to the expression in "New".
   The functions are of the form function(+,Args) or function(*,Args). */

apply_functions([],Expr,Expr) :- !.

apply_functions([H|T],Expr,New) :-
   apply_function(H,Expr,E1),
   apply_functions(T,E1,New).


/* These clauses handle bags.  If the function has the same operator as the
   bag, just add the new argument to the bag.  If the function's operator
   distributes over the bag, apply the function to each element.
   Otherwise fail. */

apply_function( function(Op,Func_Args), bag(Op,Argss),
   bag(Op,[Func_Arg|Argss]) ) :- !.


apply_function( function(Func_Op,Func_Arg), bag(Bag_Op,[]),
      bag(Bag_Op,[]) ) :-
   distributes(Func_Op,Bag_Op),
   !.

apply_function( F, bag(Bag_Op,[Arg1|Argss]),
      bag(Bag_Op,[New_Arg1|New_Argss]) ) :-
   F=function(Func_Op,_),
   !,  /* fail completely if Func_Op doesn't distribute */
   distributes(Func_Op,Bag_Op),
   apply_function(F,Arg1,New_Arg1),
   apply_function(F,bag(Bag_Op,Argss),bag(Bag_Op,New_Argss)).



/* now clauses to handle expressions not in bag form */

apply_function( function(+,Arg), Expr, Expr+Arg).

apply_function( function(*,Arg), Expr, Expr*Arg).



apply_substitutions([],X,X) :- !.

apply_substitutions([H|T],X,Z) :-
   subst(H,X,Y),
   apply_substitutions(T,Y,Z),
   !.


/* return a list of the variables from a substitution list */

subst_vars( [] , []).
subst_vars( [V=_|Rest] , [V|VList] ) :-
   subst_vars(Rest,VList).
```

```
/* PORTRAY */

portray( bas(Op,[A]) ) :-
    write(A),
    !.

portray( bas(Op,[A|Rest]) ) :-
    write(A),
    write(Op),
    portray( bas(Op,Rest) ),
    !.

portray( bas(Op,[]) ) :-
    write('<empty bas>'),
    !.



portray( transform(FList,SList,New_Unknown) ) :-
    writef('transform:\n'),
    portray_functions(FList),
    portray_subst(SList),
    (New_Unknown=false -> true ;
     writef('    change unknown to %t\n',[New_Unknown]) ),
    !.

portray_functions([]) :- !.
portray_functions( [function(*,A)|TL] ) :-
    writef('    * %p\n',[A]),
    portray_functions(TL),
    !.
portray_functions( [function(+,A)|TL] ) :-
    writef('    + %p\n',[A]),
    portray_functions(TL),
    !.

portray_subst([]) :- !.
portray_subst( [Var=E|TL] ) :-
    writef('    %p  ->  %p\n',[Var,E]),
    portray_subst(TL),
    !.



portray(X) :-
    write(X),
    !.
```

```
/* MISCELLANEOUS ROUTINES FOR POWERFUL ALGEBRAIC MATCHER */


identity(+,0).
identity(*,1).



to_bas(Op,Descr1,Descr2) :-
    expr(Descr1,Expr),
    decomp(Expr,[Op|A1]),
    rev(A1,Args),
    new_expr(Descr1,bas(Op,Args),Descr2),
    !.

to_bas(Op,Descr1,Descr2) :-
    expr(Descr1,Expr),
    new_expr(Descr1,bas(Op,[Expr]),Descr2),
    !.


from_bas(Descr1,Descr2) :-
    expr(Descr1,bas(Op,A1)),
    rev(A1,Args),
    recomp(Expr,[Op|Args]),
    new_expr(Descr1,Expr,Descr2),
    !.


or_to_list(H#T,[H|TT]) :-
    !,
    or_to_list(T,TT).

or_to_list(Expr,[Expr]) :- !.
```

```
/* PROCEDURES FOR EXTRACTING FEAURES FROM EXPRESSIONS
   calling protocol:
      features(Expr,Features)
 */


features(Expr,Features) :-
   expr(Expr,E), unknown(Expr,U),
   features(E,U,Features),
   !.



features([],U,[]) :- !.

features([H|T],U,[FH|FT]) :-
   features(H,U,FH),
   features(T,U,FT),
   !.


features(U,U,U) :- !.

features(Expr,U,mumble) :-
   freeof(U,Expr),
   !.

features(E^N,U,F^N) :-
   integer(N),
   features(E,U,F),
   !.

features(E1+E2,U,Features) :-
   features(E1,U,F1),
   features(E2,U,F2),
   (F1=mumble -> Features=F2 ;
    F2=mumble -> Features=F1 ;
    Features=F1+F2),
   !.

features(E1*E2,U,Features) :-
   features(E1,U,F1),
   features(E2,U,F2),
   (F1=mumble -> Features=F2 ;
    F2=mumble -> Features=F1 ;
    Features=F1*F2),
   !.

features(Expr,U,Features) :-
   Expr=..[Op|Args],
   features(Args,U,FArgs),
   Features=..[Op|FArgs],
   !.
```

```
/* Procedure to instantiate all the variables in a rule.
   call:       instantiate( Rule , BadNames , PatternVars )
   Uses names u,v,w if possible; otherwise use gensym'd names */

instantiate(Rule,BadNames,PatternVars) :-
   variables(Rule,PatternVars),
   bind_list(PatternVars,BadNames).


bind_list([],BadNames) :- !.

bind_list([S|T],BadNames) :-
   preferred_symbol(S),
   not member(S,BadNames), !,
   append(BadNames,[S],Bad1),
   bind_list(T,Bad1).


preferred_symbol(u).
preferred_symbol(v).
preferred_symbol(w).
preferred_symbol(X) :-
   gensym(g,X).
```

```
/* tests for powerful matcher */

 :- tlim(5).

quadratic :- try_hard_to_solve( a*x^2+b*x+c=0, x, Ans).

cubic :- try_hard_to_solve( a*x^3+b*x^2+c*x+d=0, x, Ans).

specialized_cubic :- try_hard_to_solve( z^3+h*z+s=0, z, Ans).

distrib1 :- try_hard_to_solve( a*x+x = x^2, x, Ans).

distrib2 :- try_hard_to_solve( a*x-x = x^2, x, Ans).

distrib3 :- try_hard_to_solve( x+x = x^2, x, Ans).

trig1 :- try_hard_to_solve( a*sin(x)+b*cos(x)=c, x, Ans).

change1 :- try_hard_to_solve( 5^(2*y) + 5^y + 5 = 0, y, Ans).

 hange2 :- try_hard_to_solve( 5^(2*y) - 5^(y+1) + 6 = 0, y, Ans).

change3 :- try_hard_to_solve( 3^(2*y) - 2*3^(y+2) + 81 = 0, y, Ans).

/* the following two examples are from McArthur & Keith,
      Intermediate Algebra */

change4 :- try_hard_to_solve( (3*y^2-2*y-2)^2 = 21*y^2 - 14*y - 20, y, Ans).

change5 :- try_hard_to_solve( y^6 + 7*a^3*y^3 - 8*a^6 = 0, y, Ans).
```

```
/* PROCEDURE FOR LEARNING TO SOLVE PARTICULAR FORMS OF EQUATIONS */


learn_to_solve(Form,Unknown,Eqn,Conditions) :-
   trace('trying to learn to solve %p for %p\n\n',[Eqn,Unknown],1),
 /* solve the equation using powerful matcher */
   try_hard_to_solve(Eqn,Unknown,A1),
 /* convert Eqn to the normal form */
   C=..[Form,Unknown,Eqn,Norm_Eqn],
   call(C),
 /* change symbols in equation etc. to variables */
   wordsin(A1,Symbols),
   variablize( [Eqn,Norm_Eqn,Unknown,A1,Symbols,Conditions] ,
        [EqnVar,NormVar,UnknownVar,AnsVar,SymVars,CondVars] ),
 /* make up a conversion command to execute when the new method is run */
   Convert=..[Form,UnknownVar,E1,E2],
 /* assert the new method */
   trace('asserting new method for solving %p for %p\n\n',[Eqn,Unknown],1),
   asserta((
      solve1(E1,UnknownVar,Ans) :-
         Convert,
         match(E2,NormVar),
         trace(
          'using learned method for solving equations of the form  %p\n\n',
          [Eqn],1),
         CondVars,
         tidy(AnsVar,Ans),
         !
      )),
   !.



variablize(A,B) :-
   wordsin(A,W),
   variablize(A,W,B),
   !.

variablize(A,[],A) :- !.

variablize(A,[H|T],B) :-
   /* crock - don't variablize arbitrary integers */
   (integral(H) -> A1=A ; var_subst( _=H , A, A1) ),
   variablize(A1,T,B),
   !.



/* SUBSTITUTE THAT DOESN'T BIND OLD VARS */
var_subst(Var=Const,Old,Old) :-
   var(Old),
   !.

var_subst(Var=Const,Const,Var) :- !.

var_subst(Var=Const,X,X) :-
   atomic(X),
   !.

var_subst(Var=Const,[],[]) :- !.
```

```
var_subst(Var=Const,[H|T],[H1|T1]) :-
    var_subst(Var=Const,H,H1),
    var_subst(Var=Const,T,T1),
    !.

var_subst(Var=Const,Old,New) :-
    Old=..[Op|Args],
    var_subst(Var=Const,Args,NArgs),
    New=..[Op|NArgs],
    !.



polynomial(X, L=R, poly_sum(X,PList)) :-
    poly_norm(X, L+ -1*R, P1),
    poly_sort(P1,P2),
    tidy(P2,PList),    /* kludge - clean up after normalization */
    !.


/* bubble sort for polynomial coefficients */
poly_sort(P1,P3) :-
    poly_sort1(P1,P2),
    (P1=P2 -> P3=P2 ; poly_sort1(P2,P3)),

poly_sort1( [A,B|Rest] , [X|S] ) :-
    perm2(A,B,X,Y),
    X=pair(NX,_), Y=pair(NY,_),
    NX>NY,
    poly_sort1( [Y|Rest] , S ),
    !.

poly_sort1(P1,P1) :- !.



/* general class doesn't change the expression */
  neral(X,Exp,Exp) :- !.


/* add a clause for matching stuff in polynomial normal form */
:- asserta((
match(poly_sum(X,L1),poly_sum(X,L2)) :-
    !, poly_match(L1,L2), !
)),



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

poly_match( [pair(N,0)] , [] ) :- !.

poly_match( [] , [pair(N,0)] ) :- !.

poly_match( [pair(N1,C1)|R1], [pair(N2,C2)|R2] ) :-
    (N1=N2 -> !, C1=C2, poly_match(R1,R2) ;
```

```
 N1>N2 -> !, C1=0, poly_match(R1,[pair(N2,C2)|R2]) ;
 /* N1<N2 */ !, C2=0, poly_match([pair(N1,C1)|R1],R2) ),
!.
```

```
learn_quad :- learn_to_solve( polynomial, x, a*x^2+b*x+c=0 , non_zero(a)  ).

qtest1 :- solve( x^2=9 , x, Ans ).

qtest2 :- solve( x^2-x-6=0, x, Ans).

qtest3 :- solve( (x+3)*(x+2)= 6, x, Ans).


learn_tris :- learn_to_solve( general, x, a*sin(x)+b*cos(x)=c ,
      non_zero(a) ).

ttest1 :- solve( 1*sin(x)+0*cos(x)=1 , x, Ans ).

ttest2 :- solve( 1*sin(x)+1*cos(x)=2^ (2^ -1) , x, Ans ).
```

```
/* COLLECTION ROUTINE TO INVOKE POWERFUL MATCHER */


collect(X,Old,New1) :-
    least_dom(X,Old),
    flas(try_hard,true,true),
    try_hard_to_collect(X,Old,New1),
```

```
/* CHANGE OF UNKNOWN ROUTINE USING POWERFUL MATCHER
   tries to change equation to a quadratic */

solve1(LHS=RHS,Unknown,Ans) :-
   /* move everything in equation to LHS and put in poly form */
   poly_form(LHS+ -1*RHS,Expr),
   /* cheap test to see if change of unknown is appropriate */
   quad_test(Expr,Unknown),
   trace('trying change of unknown to make equation into a quadratic\n',[],4),
   /* match against the general quadratic equation
      The "_zzz" junk is to ensure that the names in the expr and
      the rule are standardized apart. */
   make_description(Expr,Unknown,[],[],EDescr),
   make_description( a_zzz*x_zzz^2 + b_zzz*x_zzz + c_zzz ,
      x_zzz , [x_zzz,a_zzz,b_zzz,c_zzz] , [] , Q ),
   match(EDescr,Q,Transform),
   /* substitute for a,b,c,x in solution to general quadratic */
   Sqrt = (b_zzz^2+ -4*a_zzz*c_zzz)^(2^ -1),
   Sol1 = (x_zzz=(-1*b_zzz+Sqrt)*(2*a_zzz)^ -1),
   Sol2 = (x_zzz=(-1*b_zzz+(-1*Sqrt))*(2*a_zzz)^ -1),
   make_description( Sol1 , x_zzz , [] , [] , Sol1Descr ),
   make_description( Sol2 , x_zzz , [] , [] , Sol2Descr ),
   apply_transform(Transform,Sol1Descr,New1Descr),
   apply_transform(Transform,Sol2Descr,New2Descr),
   expr(New1Descr,New1), expr(New2Descr,New2),
   trace(
'\napplying transform to solution to quadratic equation yielding %e\n',
      [New1$New2],4),
   try_hard_to_solve(New1$New2,Unknown,Ans).



/* Test if the expression could be made into a quadratic with
   a change of unknown.  This test consists of seeing if the
   expression is a sum, with two terms containing the unknown,
   and one of them involving exponentiation. */
quad_test(Expr,Unknown) :-
   decomp(Expr,[+|Terms]),
   select(T1,Terms,Rest),
   contains(Unknown,T1),
   subterm(A^B,T1),
   select(T2,Rest,_),
   contains(Unknown,T2),
   !.
```

```
/* ADDITIONS TO TIDY */


/* additional tidy axioms */

nt_tidyax( (U^V)^W , U^X ) :- poly_form(V*W,X).

nt_tidyax( U^(N^ -1) , Ans ) :- eval( U^(N^ -1) , Ans).



/* new bas flushins procedure to combine like items to powers -
   put in before other procedures */
:- asserta((
f12([X|L],New) :- twofrom(L,X1^A,X2^B,R), match(X1,X2), tidy(A+B,C),
    !, f12([X,X1^C|R],New)
)).
```

```
/* ISOLATION AXIOMS THAT RETURN PARTICULAR SOLUTIONS
   When solving for a variable in a rule using the powerful matcher,
   particular rather than general solutions are desired.  */

:- asserta((
isolax( 1 , sin(U)=V , U=arcsin(V) , particular_solution(U) )
)).

:- asserta((
isolax( 1 , cos(U)=V , U=arccos(V) , particular_solution(U) )
)).

:- asserta((
isolax( 1 , tan(U)=V , U=arctan(V) , particular_solution(U) )
)).

:- asserta((
isolax( 1 , cosec(U)=V , U=arccosec(V) , particular_solution(U) )
)).

:- asserta((
isolax( 1 , sec(U)=V , U=arcsec(V) , particular_solution(U) )
)).

:- asserta((
isolax( 1 , cot(U)=V , U=arccot(V) , particular_solution(U) )
```

```
/* SQUARE ROOT EVALUATION */

/* put the new eval before the old ones */
:- asserta((
eval( X^(N^ -1), Ans) :-
    eval(X,X1),
    eval(N,N1),
    inteser(X1),
    inteser(N1),
    !,
    (N1=0 -> Ans=X ;
     N1<0 -> N2 is -N1, eval(X^(N2^ -1),A1), eval(A1^ -1,Ans) ;
     /* N1 > 0 */
     remove_powers(X1,N1,2,IPart,Residue),
     (Residue=1 -> Ans=IPart ;
      IPart=1 -> Ans=Residue^(N1^ -1) ;
      Ans=IPart*Residue^(N1^ -1) )),
    !
)),

  `move_powers(X,Power,J,1,X) :-
    intexp(J,Power,A),
    A>X,
    !,

remove_powers(X,Power,J,IPart,Residue) :-
    intexp(J,Power,A),
    0 is X mod A,
    X1 is X/A,
    remove_powers(X1,Power,J,IP1,Residue),
    IPart is IP1*J,
    !.

remove_powers(X,Power,J,IPart,Residue) :-
    J1 is J+1,
    remove_powers(X,Power,J1,IPart,Residue),
    !.
```

```
collax( W , U*W+V*W , (U+V)*W ) ,

/* collax( W , W+V*W , (V+1)*W ) ,   */

/* collax( W , V*W+(-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)*2^ (-1) ) ,

collax( U , cos(U)^2+ -1*(sin(U)^2) , cos(2*U) ) ,

  llax( 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 , sin(U)*cos(U)^(-1) , tan(U) ),

collax( U , cos(U)*sin(U)^(-1) , cot(U) ),

collax( U , U^2+2*U*V+V^2 , (U+V)^2 ),

/* tris rule for cubic */
collax( U , cos(U)^3 + (-3)*4^(-1)*cos(U) , (4^(-1))*cos(3*U) ),

collax( U, U^3 + 3*U^2*V + 3*U*V^2 + V^3 , (U+V)^3 ),
```

```
decomp(A+(B+C),L) :- !, decomp(A+B+C,L).
decomp(A+B+C,[+,C|L]) :- !, decomp(A+B,[+|L]).
decomp(A+B,[+,B,A]) :- !.

decomp(A*(B*C),L) :- !, decomp(A*B*C,L).
decomp(A*B*C,[*,C|L]) :- !, decomp(A*B,[*|L]).
decomp(A*B,[*,B,A]) :- !.

decomp(A&(B&C),L) :- !, decomp(A&B&C,L).
decomp(A&B&C,[&,C|L]) :- !, decomp(A&B,[&|L]).
decomp(A&B,[&,B,A]) :- !.

decomp(A#(B#C),L) :- !, decomp(A#B#C,L).
decomp(A#B#C,[#,C|L]) :- !, decomp(A#B,[#|L]).
decomp(A#B,[#,B,A]) :- !.


decomp(E,F) :- E=..F, !.
```

```
/* KLUDGES!  interval stuff is screwing up - just bypass it */

non_zero(X) :- !.

acute(X) :- !, fail.

non_reflex(X) :- !, fail.

non_nes(X) :- !, fail.
```

```
/* Exp is a least dominating expression of X (i.e. 2 args contain X) */
/* This patches the old "least_dom" by converting to bas representation */

least_dom(X,Exp) :-
        decomp(Exp,[F|Args]),
        sublist(contains(X),Args,XArgs),
        length(XArgs,N), N>1, !.
```

```
/* disable existing method for linear and quadratic equations so that
   the program can try to learn them */

poly_method(X,Plist, X=Ans) :-
    !,
    fail.
```

```
/* PATHS */
/* Paths are a way of describing subparts of things.
   A path consists of a list of part-extracting functions to be applied to
   some object.  The functions are either integers (arg numbers), or lists
   consisting of a functor plus n-1 arguments.  The last argument is supplied
   by apply_path, and is a variable to hold the result */

apply_path([],Expr,Expr) :- !.

apply_path([H|T],E1,E3) :-
   (integer(H) -> arg(H,E1,E2) ;
      append(H,[E2],L), C=..L, call(C) ),
   apply_path(T,E2,E3),
   !.
```

```
/* SQUARE ROOT EVALUATION */

/* put the new intexp before the old ones */
:- asserta((
intexp(L,E,A^ -1) :-
    L>0,
    E<0,
    NE is 0-E,
    intexp(L,NE,A),
    !
)).
```

```
/* LOGARITHM EVALUATION */

:- asserta((
eval1(log(Base,X),Ans) :-
    logeval(Base,X,IPart,Fraction),
    (IPart=0 -> Ans=Fraction ;
     Fraction=0 -> Ans=IPart ;
     /* return an improper fraction */
     Fraction=Denominator^ -1,
     Numerator is IPart*Denominator+1,
     Ans=Numerator*Denominator^ -1),
    !
)).



/* logeval will succeed if the logarithm can be expressed as an integer
   plus 1 over an integer */
logeval(Base,1,0,0) :- !.

  geval(Base,X,IPart,Fraction) :-
    0 is X mod Base,
    X1 is X/Base,
    logeval(Base,X1,I1,Fraction),
    IPart is I1+1,
    !.

logeval(Base,X,0,Power^ -1) :-
    X<Base,
    findpower(X,Base,2,Power),
    !.



findpower(X,Base,Test,Power) :-
    intexp(X,Test,K),
    (K<Base -> T1 is Test+1, findpower(X,Base,T1,Power), ! ;
     K=Base -> Power=Test, ! ;
     !, fail).
```

```
memo(Pred) :- call(Pred).
```

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

Neil Davey's Identity Assimilator
```

PREDICATE	FILE	CALLED BY
append(3)	utility	foosin(2)
arsn(3)	scope	position(3)
.om(3)	scope	use_for_collection(2) use_for_attraction(3) use_for_tidy1(2) use_for_normalize1(2) use_for_normalize2(2)
check1(2)	scope	
common_ancestor(3)	scope	distance(3) common_ancestor(3)
distance(3)	scope	use_for_attraction(3)
foosin(2)	scope	use_for_normalize1(2) use_for_normalize2(2) susaryist(2) foosin(2)
footerm(3)	scope	use_for_normalize2(2) footerm(3)
forall(2)	utility	use_for_normalize2(2) heisht(2)
.isht(2)	scope	heisht(2) check1(2)
inner(1)	undefined	use_for_normalize2(2)
member(2)	utility	use_for_isolation(2) use_for_collection(2) use_for_tidy1(2) use_for_tidy2(2) soln(2) pair(3) footerm(3) heisht(2)
nmember(3)	utility	arsn(3) partition(5)
occ(3)	utility	use_for_collection(2) use_for_attraction(3) use_for_tidy1(2) use_for_normalize1(2) use_for_normalize2(2)
pair(3)	scope	use_for_attraction(3) pair(3)
partition(5)	scope	qsort(4) partition(5)

position(3)	scope	use_for_attraction(3) position(3)
qsort(3)	scope	susaryist(2)
qsort(4)	scope	qsort(3) qsort(4)
soln(2)	scope	use_for_isolation(2) soln(2)
susarlist(1)	undefined	susaryist(2)
susaryist(2)	scope	use_for_normalize1(2)
union(3)	utility	wordsin(2)
use(1)	scope	
use_for_attraction(3)	scope	use(1)
use_for_collection(2)	scope	use(1)
use_for_isolation(2)	scope	use(1) use_for_isolation(2)
use_for_normalize1(2)	scope	use(1)
use_for_normalize2(2)	scope	use(1)
use_for_tidy1(2)	scope	use(1)
use_for_tidy2(2)	scope	use(1)
wordsin(2)	scope	use_for_isolation(2) use_for_collection(2) use_for_attraction(3) use_for_tidy1(2) use_for_tidy2(2) soln(2) wordsin(2)
writef(2)	utility	use(1) use_for_isolation(2) use_for_collection(2) use_for_attraction(3) use_for_tidy1(2) use_for_tidy2(2) use_for_normalize1(2) use_for_normalize2(2)

```
/* THMS.

Identities of Alsebra for use as tests for Davey's SCOPE prosram

Alan Bundy 15.7.81 */

thm1 :- use( u+v=w -> u=w-v ).

thm2 :- use( sin(2*u)/2 = sin(u)*cos(v) ).

thm3 :- use( u*(v+w) = u*v + u*w ).

thm4 :- use( u^2=v -> (u=sart(v) # u= -sart(v)) ).

thm5 :- use( sin(u)=v -> u=n*180 + ((-1)^n)*arcsin(v) ).

thm6 :- use( u^(v*w) = (u^v)^w ).

/* Run them all */
go :- thm1, thm2, thm3, thm4, thm5, thm6.
```

```
yes
| ?- go.
u+v=w->u=w-v has been used as an isolation axiom on the variable u .

u+v=w->u=w-v has now been used in as many cases as possible.

sin(2*u)/2=sin(u)*cos(v) has been used as a tidy axiom from right to left
 on the variable v .

sin(2*u)/2=sin(u)*cos(v) has now been used in as many cases as possible.

u*(v+w)=u*v+u*w has been used as an attraction axiom from right to left
 on the variables v and w .

u*(v+w)=u*v+u*w has been used as a collection axiom from right to left
 on the variable u .

u*(v+w)=u*v+u*w has now been used in as many cases as possible.

u^2=v->u=sqrt(v)#u= -sqrt(v) has been used as an isolation axiom on the variable u

u^2=v->u=sqrt(v)#u= -sqrt(v) has now been used in as many cases as possible.

sin(u)=v->u=n*180+ -1^n*arcsin(v) has been used as an isolation axiom on the variab

sin(u)=v->u=n*180+ -1^n*arcsin(v) has now been used in as many cases as possible.

u^(v*w)=(u^v)^w has been used as an attraction axiom from left to right
 on the variables u and v .

u^(v*w)=(u^v)^w has been used as an attraction axiom from right to left
 on the variables v and w .

u^(v*w)=(u^v)^w has now been used in as many cases as possible.


yes
| ?- core     50176  (20992 lo-ses + 29184 hi-ses)
heap     15872 =  14505 in use +    1367 free
global    1175 =     16 in use +    1159 free
local     1024 =     16 in use +    1008 free
trail      511 =      0 in use +     511 free
    0.01 sec. for 1 GCs gaining 416 words
    0.03 sec. for 2 local shifts and 4 trail shifts
    2.49 sec. runtime
```

```
/* SCOPE.

Neil Davy's Identity Assimilator

*/

 :- op(950,xfy,(->)).
 :- op(950,xfy,<->).



/*************************************************

    TRY E IN ALL THE SYNTACTIC GROUPS

*************************************************/

use(E) :-
        use_for_attraction(E,X,Y),
        fail.

use(E) :-
        use_for_isolation(E,X),
        fail.

use(E) :-
        use_for_collection(E,X),
        fail.

use(E) :-
        use_for_tidy1(E,X),
        fail.

use(E) :-
        use_for_tidy2(E,X),
        fail.

use(E) :-
        use_for_normalize1(E,X),
        fail.

use(E) :-
        use_for_normalize2(E,Foo),
        fail.

use(E) :- writef('%t has now been used in as many cases as possible.\n\n',[E]).


/*************************************************

    TRY E AS AN ISOLATION AXIOM

*************************************************/

use_for_isolation(Cond->(L->R), U) :-
        use_for_isolation(L->R,U),
```

```
use_for_isolation(L->R,U) :-
        wordsin(L,S),
        member(U,S),
        soln(R,U),
        assert(isolax(L->R,U)),
        writef('%t->%t has been used as an isolation axiom on the variable %t .\n\n',[L,R,U]).

/*************************************************

     TRY E AS A COLLECTION AXIOM

*************************************************/


use_for_collection(E,U) :-
        axiom(E,L,R),
        wordsin(E,S),
        member(U,S),
        occ(U,L,N1),
        occ(U,R,N2),
        N1>0,
        N2>0,
        (N1>N2,
        assert(collax(L,R,U)),
        writef('%t has been used as a collection axiom from left to right \n on the variable %t .\n\n',[E,U]);
        N1<N2,
        assert(collax(R,L,U)),
        writef('%t has been used as a collection axiom from right to left \n on the variable %t .\n\n',[E,U])).

/*************************************************

     TRY E AS AN ATTRACTION AXIOM

*************************************************/


use_for_attraction(E,X,Y) :-
        axiom(E,L,R),
        wordsin(E,S),
        pair(X,Y,S),
        position(X,L,XL),
        position(Y,L,YL),
        position(X,R,XR),
        position(Y,R,YR),
        occ(X,L,1),
        occ(X,R,1),
        occ(Y,L,1),
        occ(Y,R,1),
        distance(XL,YL,DL),
        distance(XR,YR,DR),
        (DL>DR,
        assert(attrax(X,Y,L,R)),
        writef('%t has been used as an attraction axiom from left to right \n on the variables %t and %t .\n\n',[E,X,Y]);
        DL<DR,
        assert(attrax(X,Y,R,L)),
        writef('%t has been used as an attraction axiom from right to left \n on the variables %t and %t .\n\n',[E,X,Y])).
```

```
/************************************************

     TRY E AS A TIDY AXIOM

************************************************/

/*   TRY THE FIRST TYPE OF TIDY AXIOM   */

use_for_tidy1(E,U) :-
        axiom(E,L,R),
        wordsin(E,S),
        member(U,S),
        occ(U,L,NL),
        occ(U,R,NR),
        (NL>0,
        NR=0,
        assert(tidyax(L,R,U)),
        writef('%t has been used as a tidy axiom from left to right \n on the variable %t .\n\n',[E,U]);
        NL=0,
        NR>0,
        assert(tidyax(R,L,U)),
        writef('%t has been used as a tidy axiom from right to left \n on the variable %t .\n\n',[E,U])).

/*   TRY THE SECOND TYPE OF TIDY AXIOM   */


use_for_tidy2(L=R,U) :-
        wordsin(L,S),
        member(U,S),
        (R=U,
        assert(tidy_ax(L,R,U)),
        writef('%t has been used as a tidy axiom from left to right \n on the variable %t .\n\n',[E,U]);
        L=U,
        assert(tidy_ax(R,L,U)),
        writef('%t has been used as a tidy axiom from right to left \n on the variable %t .\n\n',[E,U])).

/************************************************

     TRY E AS A NORMALIZE AXIOM

************************************************/

/*   TRY THE FIRST TYPE OF NORMALIZE AXIOM   */

use_for_normalize1(E,Foo) :-
        axiom(E,L,R),
        subsyist(E,Foo),
        foosin(L,SL),
        occ(Foo,SL,NL),
        foosin(R,SR),
        occ(Foo,SR,NR),
        (NL>0,
        NR=0,
        assert(normax(L,R,Foo)),
        writef('%t has been used as a normalize axiom from left to right \n on the function %t .\n\n',[E,Foo]);
        NR>0,
        NL=0,
```

```
        assert(normax(R,L,Foo)),
        writef('%t has been used as a normalize axiom from right to left \n on the function %t .\n\n',[E,Foo])).

/*   TRY THE SECOND TYPE OF NORMALIZE AXIOM   */

use_for_normalize2(E,Foo) :-
        axiom(E,L,R),
        inner(Foo),
        (foosin(L,SL),
        occ(Foo,SL,1),
        footerm(T,Foo,L),
        forall(footerm(S,Foo,R), check1(S,T)),
        assert(normax(L,R,Foo)),
        writef('%t has been used for a normalize axiom from \n left to right on the funtion %t .\n\n',[E,Foo]);
        foosin(R,SR),
        occ(Foo,SR,1),
        footerm(T,Foo,R),
        forall(footerm(S,Foo,L), check1(S,T)),
        assert(normax(R,L,Foo)),
        writef('%t has been used for a normalize axiom from \n right to left on the function %t .\n\n',[E,Foo])).

/*****************************************************

     THE SUBROUTINES USED IN THE PROGRAM

*****************************************************/

soln(U=R,U) :-
        wordsin(R,S),
        not(member(U,S)).

soln(A#B,U) :-
        soln(A,U),
        soln(B,U).


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

wordsin([First|Rest],Set) :- !,
        wordsin(First,S1),
        wordsin(Rest,S2),
        union(S1,S2,Set).

wordsin(X,[]) :- integer(X), !.

wordsin(X,[X]) :- atom(X), !.

wordsin(Term,Set) :-

        Term=..[Functor|Args],
        wordsin(Args,Set).


axiom(L=R,L,R).
axiom(L<->R,L,R).


position(X,E,[]) :-
```

```
        E=..[X|Args].
position(X,E,L) :-
        arsn(N,E,T),
        L=[N|L1],
        position(X,T,L1).


distance(X,Y,D) :-
        common_ancestor(X,Y,L),
        lensth(X,N1),
        lensth(Y,N2),
        lensth(L,N3),
        D is (N1-N3)+(N2-N3).


common_ancestor([F1|R1],[F2|R2],[]) :- F1=\=F2.
common_ancestor([F|R1],[F|R2],[F|R]) :-
        common_ancestor(R1,R2,R).


arsn(N,T,X) :- T=..[_|L],nmember(X,L,N).


susaryist(E,Foo) :-
        foosin(E,S),
        susarlist(D),
        asort(S,L,D),
        L=[Foo|Rest].

foosin([],[]) :- !.
foosin([F|R],S) :- !,
        foosin(F,S1),
        foosin(R,S2),
        appsnd(S1,S2,S).

foosin(X,[]) :-
        atomic(X), !.
foosin(Term,S) :-
        Term=..[Functor|Args],
        foosin(Args,S1),
        S=[Functor|S1].



asort(L,R,D) :-
        asort(L,[],R,D).

asort([X|L],R0,R,D) :-
        partition(L,X,L1,L2,D),
        asort(L2,R0,R1,D),
        asort(L1,[X|R1],R,D).
asort([],R,R,D).

partition([X|L],Y,[X|L1],L2,D) :-
        nmember(X,D,NX),
        nmember(Y,D,NY),
        NX=<NY,
        partition(L,Y,L1,L2,D).
partition([X|L],Y,L1,[X|L2],D) :-
```

```
        nmember(X,D,NX),
        nmember(Y,D,NY),
        NX>NY,
        partition(L,Y,L1,L2,D).
partition([],_,[],[],D).


pair(X,Y,[X|R]) :-
        member(Y,R).

pair(X,Y,[_|R]) :-
        pair(X,Y,R).




footerm(T,Foo,E) :-
        E=..[Foo|_],
        E=T.

footerm(T,Foo,E) :-
        E=..[_|Args],
        member(X,Args),
        footerm(T,Foo,X).
height(T,H) :-
        atomic(T),
        H is 0.

height(T,H) :-
        T=..[_|Args],
        member(X,Args),
        forall(member(Y,Args), check1(X,Y)),
        height(X,HX),
        !,
        H is HX+1.

check1(X,Y) :- height(X,HX),
        height(Y,HY),
        HX =< HY.
```