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 extra!facile)!
  
  go - 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 los) and halt.

STOP PRESS

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

From Richard[400,422] on April 8, 1981 at 8:49 PM
I've implemented your proposed los algorithm, with a variety of embellishments (so los(-3,-27) = 3 !!)

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

The code finally gets around to asking the question 'for sien R and X, find intesers N, D such that B*N = X^D'.
I assumed that the answer was to find G=gcd(B,X), then D = los(G,B) and N = los(G,X). Well, this is true as far as it goes, but D and N aren't always intesers. Then again, there may be a G for which N and D are intesers, and it divides the scd, 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 LUNG, 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 inteser arithmetic throughout) it will return the correct los if there is one, and fail only if there is not. As far as I can see the time to discover that a los doesn't exist shouldn't be markedly different from the time to discover that a similar los 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 los(B,X)-es (some very implausible indeed, e.g. los(X,0)=+-infinity, los(-3,-37) = 3, and so on) without finding any mistakes (but don't try los(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

LONGER.[400,422] now incorporates a new scd 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 wrappings 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 bad 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 invariant 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 has it in a "final form" whereas the other is in an "intermediate form" (ie Left and Ltas 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 worriesing about the amount of unneeded cons' performed has made the things several orders of magnitude harder to do than would otherwise have been the case. 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 so on the old equation, ie the one before chunk was called. I remembered that we had previously overcome this by putting a cut after change unknown, looking (or even just looking!) at the code it seemed that the cut had got omitted from the o be called directly by solve 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, a 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 los routine
new scd routine
eval now has scd(A,B) - takes scds 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(+100,100,100) 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 homosanisation 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 ‘bugs’. 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 
\[(x^2+3x+7)\times(2^1)
\]

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.

...The dreaded 119 has gone on vacation.

From Richard[400,4221] on April 22, 1981 at 5:18 PM

- There is a new version of ARITHMONG.
- There is new code to deal with `xgcd(...)` which works compiled as well as interpreted. The largest number you can type in is roughly \(3.435\times10^{19}\).

2) I have renamed practically everything. Don’t despair, I’ve just changed the suffixes, so that \(\alpha=\text{rational}, \pi=\text{signed integer}, n=\text{natural (or 0)}\).

3) A clause has been added to make \(O^\cdot(N)\) = 0; I’d forgotten it before.

4) The radix is now \(10^{15}\) instead of \(10^{14}\).

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 MECHO folder.

From Bernard[400,4322] on May 12, 1981 at 1:22 PM

A problem with odd_anti_symmetric, not an **ERROR evaluate(number(+,[4096],[4543])) (eg!).

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 `simea` is in `press!simea!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 11:07 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 \( (x^2 + x + 3)^r = 8 \)
It cannot solve \((x+1)^r (1/2) + x = 4\), 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 11:16 PM
Message 3!
Just a list problems.
1) Most important is the looping the current nasty causes. Maybe this is easily dealt with. I'm waiting to see what the AI2 sys produces.
3) Tidy doesn't know that \( \log(x, x) = 1 \), and \( \log(x, 1) = 0 \) (and unifying \( \log(i, i) = 0 \)), the thing is reloaded from 프로선. I don't know whether we can do it before.
4) Big problem! Collection and attraction and the anyone routine, 프로젝트! 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 11:52 PM
Problem. \( \sin(x^2 + x) = 1/2 \) solves fine, \( \cos(x^2 + x) = 1/2 \) fails. To the uninitiated the two equations seem similar. The problem lies in isolate, specifically isolate1. With the cos problem the goal isolate1(\{x^2 + x = bla \} x^2 + x = bla1, 434 = 435) is set up and fails, no matching, with this 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 homos 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 Lons. 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 on June 24, 1981 at 7:50 PM
PRESS reloaded at 19:50:13 on 24 Jun 81
CC:Bundy Hps,Sterlins Hps,Silver Hps,Okeefe Hps,Byrd Hps

From Lawrence on July 1, 1981 at 8:38 PM
PRESS reloaded at 20:38:14 on 1 Jul 81
CC:Bundy Hps,Sterlins Hps,Silver Hps,Okeefe Hps,Byrd Hps

From Lawrence on July 8, 1981 at 9:26 PM
New version of UTIL with slight fix to LONG. Not worth reloading anything until necessary.
CC:Bundy Hps,Sterlins Hps,Silver Hps,Okeefe Hps,Byrd Hps

From Leon on July 15, 1981 at 5:43 PM
PRESS 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 alhard.
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 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 on July 18, 1981 at 3:49 PM
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 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 MassTapes containing the Prolos 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,Burd 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,Burd 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

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

From Bernard[400,4322] on September 4, 1981 at 2:09 PM

The following things have been done to Press by me:
1) Los 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) Homos 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 trigs equivalents
   (This change occurred because of an exam question)
5) exam has been modified. All questions are called one of the following
   aeb(N),lon(N),ofx(N),dloN(N),integer n
   aeb has been extended, dloN are London syllabus D A level questions.
   To run the AEB type aebrun, and similarly for the others.
To run aeb\(N\) onwards type aeb\text{recurse}(N)\,(\text{run means time as well})

Someone should extract a subset of problems we wish to make standard.

6) A slight change has occurred in arhint\textendash;instead of saying \textquote{where n34 is an arbitrary integer,\textit{double line feed} it says \textquote{letting n34 be an arb etc},\textit{no full stop} and single line feed.\textit{This makes the output more logical.}

Thats all I can remember

7) Apart from changing \textit{misc\textunderscore{chunk\textunderscore{homog}}, I've written in \textit{misc} the 4 \textit{argument predicate correspond} which works as:

\texttt{correspond(X,List1, List2, Y) :- nmember(X,List1,N), nmember(Y,List2,N), !,}

although I've written it without \textit{nmember} (so it works easily whether \(X\) or \(Y\) is instantiated).\textit{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

\textit{trying about that, to recap:}

PRESS solves:
19 out of 36 AEB Problems, \textit{of the 27 single equations it solves 18}
19 out of 21 London Problems, \textit{18 out of 20 single equations (100\% success on single solve!!)}
11 out of 13 Oxford, (2 out of 3 A level ones)
9 out of 10 single equations
7 out of 10 London syllabus D, 7 out of 10 single equations \textit{ie there weren't any single solves!!}

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 guess routine in \texttt{polpak}, 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 \texttt{cond\textunderscore{Print}(Old,New)}, which I've put in \texttt{misc\textunderscore{cond\textunderscore{Print}} prints \textquote{Tidying to New} if New doesn't match Old.

\textit{Note: We adopt the Prolog convention of variable names begin with an upper-case letter, \textit{EXCEPT} in the case of \textquote{Tidying} and \textquote{EXCEPT}. \textit{The former is a word whereas \textquote{New} is a variable, that is to say \ldots \textit{etc}, \textit{etc}.)}

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

PressProject

Alan Borning's 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:

\begin{verbatim}
OLDMAT.001 original mas file
OLDMAT.002 new mas file of all the individual files
\end{verbatim}

\textit{that were lying around. Includes \textquote{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 Borning.
[NB for reference: I now save all MAS files with three digit 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 SURFIL 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,Burd Hps

From Bernard[400,4322] on September 10, 1981 at 11:46 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 parse!, this is quicker, and I've removed some clauses that weren't needed.
chunk has been optimised (maybe!!), still doesn't sort out the lewis problem
'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 tris 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 11:54 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.eons by homogenization. It works on some of the outstanding A level questions (Top level is sim1(Equation, Listofunknowns, Ans), rather than sim+solve)

From Leon SC[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 buggs 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 sim+solve we can solve 6 more problems aeb:13, 18, 25, 29, 31 and 33.
I haven’t yet moved los,solve,coller,ax,attrac,ax,na81 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 IN1 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 isolate, 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 isolate

Bernard

CC:Bundu Hps, Sterling Hps

From Leon SL[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 SL[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:Bundu Hps, Sterling Hps, Silver Hps, Okeefe Hps, Byrd Hps
PRESS (1 Jul 81)

! ?- [goals].

goals reconsulted 158 words 1.25 sec.

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

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

\[
e = (((x + 1) * (x - 1)) \right)^{(1/3)}
\text{(by Isolation)}
\]

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

\[
x ^ 2 + -1 = e ^ 3
\text{(by Isolation)}
\]

\[
x ^ 2 = e ^ 3 + -1 * -1
\text{(by Isolation)}
\]

\[
x = (e ^ 3 + -1 * -1) ^ (1/2) # x = -1 * (e ^ 3 + -1 * -1) ^ (1/2)
\text{(by Isolation)}
\]

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

Trac ran took 1492 milliseconds and produced answer
(X & 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 \( x \)

\[
2 ^ x ^ 5 = 2
\]

\[
x ^ 5 = \log(2, 2)
\text{(by Isolation)}
\]

\[
x = \log(2, 2) ^ (1/5)
\text{(by Isolation)}
\]

Answer is:
X1
where:
X1 = x = 1
Solving \((2 \cdot \cos(x) \cdot 2 \cdot \sin(x) \cdot 2) \cdot \sin(x) = 2 \cdot (1/4)\) for \(x\):

\[
(2 \cdot \cos(x) \cdot 2 \cdot \sin(x) \cdot 2) \cdot (\sin(x \cdot 2) \cdot (1/2)) = 2 \cdot (1/4)
\]

\[
(2 \cdot (\cos(x) \cdot 2 + \sin(x) \cdot 2)) \cdot (\sin(x \cdot 2) \cdot (1/2)) = 2 \cdot (1/4)
\]

\[
2 \cdot (\sin(x \cdot 2) \cdot 1 \cdot (1/2)) = 2 \cdot (1/4)
\]

\[
\sin(x \cdot 2) \cdot 1 \cdot (1/2) = \log(2, 2 \cdot (1/4))
\]

(by Isolation)

\[
\sin(x \cdot 2) \cdot 1 = \log(2, 2 \cdot (1/4)) \cdot 2
\]

(by Isolation)

\[
\sin(x \cdot 2) = \log(2, 2 \cdot (1/4)) \cdot 2 \cdot 1
\]

(by Isolation)

where \(n_1\) denotes an arbitrary integer.

\[
x \cdot 2 = n_1 \cdot 180 + -1 \cdot n_1 \cdot \arcsin(\log(2, 2 \cdot (1/4)) \cdot 2 \cdot 1)
\]

(by Isolation)

\[
x = (n_1 \cdot 180 + -1 \cdot n_1 \cdot \arcsin(\log(2, 2 \cdot (1/4)) \cdot 2 \cdot 1)) \cdot (1/2)
\]

(by Isolation)

Answer is:

\[
X_1
\]

\[
\text{where } \downarrow
\]

\[
X_1 = x = -1 \cdot n_1 \cdot 15 + n_1 \cdot 90
\]

nvoteen took 6179 milliseconds and produced answer

\[
X_1
\]

\[
\text{where } \downarrow
\]

\[
X_1 = x = -1 \cdot n_1 \cdot 15 + n_1 \cdot 90
\]

Solving \(1 / x = 2 = 1 / x\) for \(x\):

Multiply through by \(x \cdot 2\) to set a polynomial

\(x = 1\) is a solution

Answer is:

\[
X_1
\]

\[
\text{where } \downarrow
\]

\[
X_1 = x = 1
\]

nspolyeon took 143 milliseconds and produced answer

\[
X_1
\]

\[
\text{where } \downarrow
\]

\[
X_1 = x = 1
\]
Solving $a^x + 1 + a^x = c$ for $x$

Homogenized equation is $a^x + a^x + (a^x) - 2 = c$

New equation is $a^x + 1 + a^x - 2 = c$

Using quadratic equation formula
$x_1 = (a^x - 1 + (a^2 + c * 4) - (1/2)) * (1/2) * x_1 = (a^x - 1 + (a^2 + c * 4)$

Applying substitution
$x_1 = a^x$

to
$x_1 = (a^x - 1 + (a^2 + c * 4) - (1/2)) * (1/2) * x_1 = (a^x - 1 + (a^2 + c * 4)$

solves
$a^x = (a^x - 1 + (a^2 + c * 4) - (1/2)) * (1/2) * x_1 = (a^x - 1 + (a^2 + c * 4)$

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

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

Answer is:
$(X_1 \neq X_2)$

where:

$X_1 = x = \log(a, (a^x - 1 + (a^2 + c * 4) - (1/2) * (1/2) + a^x (-1/2))$

$X_2 = x = \log(a, (a^x - 1 + (a^2 + c * 4) - (1/2) * (-1/2) + a^x (-1/2))$

Homogen took 2162 milliseconds and produced answer
$(X_1 \neq X_2)$

where:

$X_1 = x = \log(a, (a^x - 1 + (a^2 + c * 4) - (1/2) * (1/2) + a^x (-1/2))$

$X_2 = x = \log(a, (a^x - 1 + (a^2 + c * 4) - (1/2) * (-1/2) + a^x (-1/2))$

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

Substituting $x_2$ for $\cos(x)$

solves $x_2^2 + b * x_2 = c$

Using quadratic equation formula
$x_2 = (b^x - 1 + (b^2 + c * 4) - (1/2)) * (1/2) * x_2 = (b^x - 1 + (b^2 + c * 4)$

Applying substitution
$x_2 = \cos(x)$

to
$x_2 = (b^x - 1 + (b^2 + c * 4) - (1/2)) * (1/2) * x_2 = (b^x - 1 + (b^2 + c * 4)$

solves
$\cos(x) = (b^x - 1 + (b^2 + c * 4) - (1/2)) * (1/2) * \cos(x) = (b^x - 1 + (b^2$

where $n_2$ denotes an arbitrary integer.

$x = 2 * n_2 * 180 + \arccos((b^x - 1 + (b^2 + c * 4) - (1/2)) * (1/2)) * x = 2$

$(1/2)) * (1/2))$
(by Isolation)
where \( n_3 \) denotes an arbitrary integer.

\[
x = 2 \times n_3 \times 180 + \arccos((b \times -1 + (b^2 + c^4)^{1/2} - 1) \times (1/2)) \quad \text{and} \quad \arccos((b^2 + c^4)^{1/2} + b \times (-1/2)) \quad \text{are}.
\]

\[
\text{Simultaneously solving:}
\]
\[
a = b
\]
\[
b = c
\]
\[
c = 1
\]

For \( \{a, c, b\} \),

Solving \( a = b \) for \( a \)

Answer is:

\[
X1
\]

where:

\[
X1 = a = b
\]

Applying substitution

\[
a = b
\]

\[
b = c
\]

\[
c = 1
\]

Solving \( b = c \) for \( c \)

Answer is:

\[
X1
\]

where:

\[
X1 = c = b
\]

Applying substitution

\[
c = b
\]

\[
to:
\]

\[
c = 1
\]
\[ b = 1 \]

Solving \( b = 1 \) for \( b \)

Answer is:
\[ X1 \]
where:
\[ X1 = b = 1 \]

Substituting back in \( c \) solution
Applying substitution
\[ b = 1 \]
to:
\[ c = b \]

Substituting back in \( a \) solution
Applying substitution
\[ b = 1 \]
\[ c = 1 \]
to:
\[ a = b \]

Final Answers are:
\( (\{ X1 \land X2 \} \land X3) \)
where:
\[ X1 = b = 1 \]
\[ X2 = c = 1 \]
\[ X3 = a = 1 \]

Simultaneous solving:
\[ m1 \land s1 \land \cos(180) + (1 \land tsn + 0) = m1 \land (a1 \land 1) \]
\[ m2 \land s1 \land 1 + (\cos(180) \land tsn + 0) + 0 = m2 \land (a1 \land 1) \]
For \( \{tsn, a1\} \).
Solving \( m1 \land s1 \land \cos(180) + (1 \land tsn + 0) = m1 \land (a1 \land 1) \) for \( tsn \):
\[ tsn = m1 \land a1 + -1 \land (m1 \land s1 \land -1) \]
(by Isolation)

Answer is:
\[ X1 \]
where:
\[ X_1 = tsn = (a_1 + s) \times m_1 \]

Applying substitution
\[ tsn = (a_1 + s) \times m_1 \]
to
\[ m_2 \times s \times 1 + (\cos(180) \times tsn + 0) + 0 = m_2 \times (a_1 \times 1) \]
gives
\[ m_2 \times s + (a_1 + s) \times m_1 \times -1 = m_2 \times a_1 \]

Solving \( m_2 \times s + (a_1 + s) \times m_1 \times -1 = m_2 \times a_1 \) for \( a_1 \)
\[ a_1 = (m_1 \times -1 + m_2) \times s \times (m_1 \times -1 + m_2 \times -1) \times -1 \times -1 \text{ is a solution} \]

Answer is \( a_1 \)

where
\[ X_1 = a_1 = s \times (m_1 \times -1 + m_2 \times -1) \times -1 \times m_1 + s \times (m_1 \times -1 + m_2 \times -1) \times -1 \times m_2 \times -1 \]

Substituting back in \( tsn \) solution

\( tsn = (a_1 + s) \times m_1 \)
to

gives
\[ tsn = (s \times (m_1 \times -1 + m_2 \times -1) \times -1 \times m_1 + s \times (m_1 \times -1 + m_2 \times -1) \times -1 \times m_2 \times -1) \times -1 \times m_2 \times -1 \]

Final Answers are:

\( \) (\( X_1 \) & \( X_2 \))
where
\[ X_1 = a_1 = s \times (m_1 \times -1 + m_2 \times -1) \times -1 \times m_1 + s \times (m_1 \times -1 + m_2 \times -1) \times -1 \times m_2 \times -1 \]
\[ X_2 = tsn = (s \times (m_1 \times -1 + m_2 \times -1) \times -1 \times m_1 + s \times (m_1 \times -1 + m_2 \times -1) \times -1 \times m_2 \times -1) \times -1 \times m_2 \times -1 \]

Simultaneously solving:
\[ v \times 2 = 0 \times 2 + 5 \times (60 \times 60) \times -2 / (1760 \times 3) \times 2000 / 1760 \]
For \( |v| \).

Solving \( v \times 2 = 0 \times 2 + 5 \times (60 \times 60) \times -2 / (1760 \times 3) \times 2000 / 1760 \) for \( v \)

\[ v = (1687500/121) \times (1/2) \times v = -1 \times (1687500/121) \times (1/2) \]
(by Isolation)

Answer is:

\( \) (\( X_1 \) & \( X_2 \))
where
\[ X_1 = v = (1687500/121) \times (1/2) \]
\[ X_2 = v = (1687500/121) \times (1/2) \times -1 \]

Applying substitution
\[ v = (1687500/121) \times (1/2) \]
Applying substitution
\[ v = \left(\frac{1687500}{121}\right)^{(1/2)} \times -1 \]
to 
\[ v = \left(\frac{1687500}{121}\right)^{(1/2)} \times -1 \]
solves 

Substituting back in \( v \) solution
Applying substitution
\[ v = \left(\frac{1687500}{121}\right)^{(1/2)} \]
to 
\[ v = \left(\frac{1687500}{121}\right)^{(1/2)} \]
solves 
\[ v = \left(\frac{1687500}{121}\right)^{(1/2)} \]

Final Answers are:
( \( X_1 \neq X_2 \)
where:
\[ X_1 = v = \left(\frac{1687500}{121}\right)^{(1/2)} \]
\[ X_2 = v = \left(\frac{1687500}{121}\right)^{(1/2)} \times -1 \]

mul took 976 milliseconds and produced answer
( \( X_1 \neq X_2 \)
where:
\[ X_1 = v = \left(\frac{1687500}{121}\right)^{(1/2)} \]
\[ X_2 = v = \left(\frac{1687500}{121}\right)^{(1/2)} \times -1 \]

Simultaneously solving:
\[ m_1 \ast s \ast \cos(270) + (1 \ast tsn + (\cos(-270) \ast reaction1 + 1 \ast mu \ast reaction1 + m_2 \ast s \ast 1 + (\cos(180) \ast tsn + 0) + 0 = m_2 \ast (a_1 \ast 1) \]
\[ m_1 \ast s \ast 1 + (\cos(270) \ast tsn + (reaction1 + \cos(270) \ast mu \ast reaction1 + 0)) \]

For [reaction1, tsn, a1].
Solving \( m_1 \ast s \ast \cos(270) + (1 \ast tsn + (\cos(-270) \ast reaction1 + 1 \ast mu \ast reaction1 + mu \ast reaction1 = m_1 \ast a_1 + -1 \ast tsn \)
(by Isolation)
Solving \( m_1 \ast s \ast 1 + (\cos(270) \ast tsn + (reaction1 + \cos(270) \ast mu \ast reaction1 + 0 \]
reaction1 = \( 0 + -1 \ast (m_1 \ast s) \)
(by Isolation)

Answer is:
\[ X_1 \]
where ;
\[ X_1 = \text{reaction1} = m_1 \times s \times -1 \]

**Applying substitution**
\[ \text{reaction1} = m_1 \times s \times -1 \]

to
\[ m_1 \times s \times \cos(270) + (1 \times tsn + (\cos(-270)) \times \text{reaction1} + 1 \times \mu \times \text{reaction1} + \]
\[ m_2 \times s \times 1 + (\cos(180) \times tsn + 0) + 0 = m_2 \times (a_1 \times 1) \]

\[ \text{solves} \]
\[ tsn + \mu \times m_1 \times s \times -1 = m_1 \times a_1 \]
\[ m_2 \times s + tsn \times -1 = m_2 \times a_1 \]

**Solving**
\[ tsn + \mu \times m_1 \times s \times -1 = m_1 \times a_1 \]
\[ tsn = m_1 \times a_1 \times -1 \times (\mu \times m_1 \times s \times -1) \]

(by Isolation)

Answer is:
\[ X_1 \]

\[ tsn = (a_1 + \mu \times s) \times m_1 \]

**Applying substitution**
\[ tsn = (a_1 + \mu \times s) \times m_1 \]

to
\[ m_2 \times s + tsn \times -1 = m_2 \times a_1 \]

\[ \text{solves} \]
\[ m_2 \times s + (a_1 + \mu \times s) \times m_1 \times -1 = m_2 \times a_1 \]

**Solving**
\[ m_2 \times s + (a_1 + \mu \times s) \times m_1 \times -1 = m_2 \times a_1 \]
\[ a_1 = (\mu \times m_1 \times -1 + m_2) \times s \times (m_1 \times -1 + m_2 \times -1) \]
\[ -1 \times -1 \text{ is a solution} \]

Answer is:
\[ X_1 \]

\[ a_1 = \mu \times s \times (m_1 \times -1 + m_2 \times -1) \]
\[ -1 \times m_1 + s \times (m_1 \times -1 + m_2 \times -1) \]
\[ -1 \times -1 \]

**Substituting back in tsn solution**

**Applying substitution**
\[ a_1 = \mu \times s \times (m_1 \times -1 + m_2 \times -1) \]
\[ -1 \times m_1 + s \times (m_1 \times -1 + m_2 \times -1) \]

\[ tsn = (a_1 + \mu \times s) \times m_1 \]

\[ \text{solves} \]
\[ tsn = (\mu \times s \times (m_1 \times -1 + m_2 \times -1) \]
\[ -1 \times m_1 + s \times (m_1 \times -1 + m_2 \times -1) \]
\[ -1 \]

**Substituting back in reaction1 solution**

**Applying substitution**
\[ a_1 = \mu \times s \times (m_1 \times -1 + m_2 \times -1) \]
\[ -1 \times m_1 + s \times (m_1 \times -1 + m_2 \times -1) \]

\[ tsn = (\mu \times s \times (m_1 \times -1 + m_2 \times -1) \]
\[ -1 \times m_1 + s \times (m_1 \times -1 + m_2 \times -1) \]
\[ -1 \]

\[ \text{to} \]
\[ \text{reaction1} = m_1 \times s \times -1 \]

\[ \text{solves} \]
\[ \text{reaction1} = m_1 \times s \times -1 \]
Final Answers are:
({X1 & X2) & X3)

where:
X1 = a1 = mu * s * (m1 * -1 + m2 * -1) ^ -1 * m1 + s * (m1 * -1 + m2 * -1)
X2 = tsn = (mu * s * (m1 * -1 + m2 * -1) ^ -1 * m1 + s * (m1 * -1 + m2 * -1)
X3 = reaction1 = m1 * s * -1

Pulltab took 3118 milliseconds and produced answer
({X1 & X2) & X3)

where:
X1 = a1 = mu * s * (m1 * -1 + m2 * -1) ^ -1 * m1 + s * (m1 * -1 + m2 * -1)
X2 = tsn = (mu * s * (m1 * -1 + m2 * -1) ^ -1 * m1 + s * (m1 * -1 + m2 * -1)
X3 = reaction1 = m1 * s * -1

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

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

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

Isolatins x on the lhs sives
x > (sin(y) ^ 2 + 1) ^ -1

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

stvinea 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},
Solvins t0 = t1 + (t2 + (t3 + 0)) for t0

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

Applying substitution
t0 = t1 + t2 + t3

t0
45 / 60 = 0 + 2 ^ -1 / 60 ^ 2 * t1
45 / 60 * t2 = d2
\[
0 = \frac{45}{60} + \frac{-2}{60} \sim 2 \times t3
\]
\[
7 = \frac{d1}{(d2 + (d3 + 0))}
\]
\[
d1 = \frac{0}{t1} + 1 / 2 \times 2 \sim -1 / 60 \sim 2 \times t1 \sim 2
\]
\[
d3 = \frac{45}{60} \times t3 + 1 / 2 \times -2 / 60 \sim 2 \times t3 \sim 2
\]

Solves:
\[
(3/4) = t1 \times (1/7200)
\]
\[
t2 \times (3/4) = d2
\]
\[
0 = t3 \times (-1/1800) + (3/4)
\]
\[
7 = d1 + d2 + d3
\]
\[
d1 = t1 \sim 2 \times (1/14400)
\]
\[
d3 = t3 \times (3/4) + t3 \sim 2 \times (-1/3600)
\]

Solving \((3/4) = t1 \times (1/7200)\) for \(t1\)

\[
t1 = (3/4) \times 7200
\]

(by Isolation)

Answer is:
\[
X1
\]
where:
\[
X1 = t1 = 5400
\]

Applying substitution
\[
t1 = 5400
\]

to:
\[
d1 = t1 \sim 2 \times (1/14400)
\]
\[
t2 \times (3/4) = d2
\]
\[
0 = t3 \times (-1/1800) + (3/4)
\]
\[
7 = d1 + d2 + d3
\]
\[
d3 = t3 \times (3/4) + t3 \sim 2 \times (-1/3600)
\]

Solves:
\[
d1 = 2025
\]
\[
t2 \times (3/4) = d2
\]
\[
0 = t3 \times (-1/1800) + (3/4)
\]
\[
7 = d1 + d2 + d3
\]
\[
d3 = t3 \times (3/4) + t3 \sim 2 \times (-1/3600)
\]

Solving \((3/4) \times d2\) for \(t2\)

\[
t2 = d2 \times (4/3)
\]

(by Isolation)

Answer is:
\[
X1
\]
where:
\[
X1 = t2 = d2 \times (4/3)
\]

Applying substitution
\[
t2 = d2 \times (4/3)
\]

to:
\[
d1 = 2025
\]
\[
0 = t3 \times (-1/1800) + (3/4)
\]
\[
7 = d1 + d2 + d3
\]
\[
d3 = t3 \times (3/4) + t3 \sim 2 \times (-1/3600)
\]

Solves:
\[ d_1 = 2025 \]
\[ 0 = t_3 * (-1/1800) + (3/4) \]
\[ 7 = d_1 + d_2 + d_3 \]
\[ d_3 = t_3 * (3/4) + t_3 ^ 2 * (-1/3600) \]

Solving \[ 0 = t_3 * (-1/1800) + (3/4) \] for \( t_3 \)

\[ t_3 * (-1/1800) = 0 + -1 * (3/4) \]
(by Isolation)

\[ t_3 = (0 + -1 * (3/4)) * -1800 \]
(by Isolation)

Answer is:
\[ X_1 \]
where:
\[ X_1 = t_3 = 1350 \]

Applying substitution
\[ t_3 = 1350 \]
\[ d_3 = t_3 * (3/4) + t_3 ^ 2 * (-1/3600) \]
\[ d_1 = 2025 \]
\[ 7 = d_1 + d_2 + d_3 \]

Solves:
\[ d_3 = (2025/4) \]
\[ d_1 = 2025 \]
\[ 7 = d_1 + d_2 + d_3 \]

Solving \[ 7 = d_1 + d_2 + d_3 \] for \( d_2 \)

\[ d_1 + d_2 = 7 + -1 * d_3 \]
(by Isolation)

\[ d_2 = 7 + -1 * d_3 + -1 * d_1 \]
(by Isolation)

Answer is:
\[ X_1 \]
where:
\[ X_1 = d_2 = d_3 * -1 + d_1 * -1 + 7 \]

Applying substitution
\[ d_2 = d_3 * -1 + d_1 * -1 + 7 \]

\[ d_3 = (2025/4) \]
\[ d_1 = 2025 \]

Solves:
\[ d_3 = (2025/4) \]
\[ d_1 = 2025 \]

Solving \[ d_1 = 2025 \] for \( d_1 \)

Answer is:
\[ X_1 \]
where:
X1 = d1 = 2025

Applying substitution
d1 = 2025
to
  d3 = (2025/4)
saves:
  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
saves:
  d1 = 2025

Substituting back in d2 solution
Applying substitution
  d3 = (2025/4)
  d1 = 2025
to
  d2 = d3 * -1 + d1 * -1 + 7
saves:
  d2 = (-10097/4)

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

Substituting back in t2 solution
Applying substitution
  (d3 = (2025/4) & d1 = 2025) & d2 = (-10097/4)
  t3 = 1350
to
  t2 = d2 * (4/3)
saves:
  t2 = (-10097/3)
Substituting back in to solution

Applying substitution
$$((d_3 = (2025/4) \land d_1 = 2025) \land d_2 = (-10097/4)) \land t_3 = 1350$$
$$t_2 = (-10097/3)$$

$$t_1 = 5400$$

Substituting back in to solution

Applying substitution
$$((d_3 = (2025/4) \land d_1 = 2025) \land d_2 = (-10097/4)) \land t_3 = 1350 \land t_2 = (-10097/3)$$
$$t_1 = 5400$$

$$t_0 = t_1 + t_2 + t_3$$

$$t_0 = (10153/3)$$

Final Answers are:

$$X_1 = d_3 = (2025/4)$$
$$X_2 = d_1 = 2025$$
$$X_3 = d_2 = (-10097/4)$$
$$X_4 = t_3 = 1350$$
$$X_5 = t_2 = (-10097/3)$$
$$X_6 = t_1 = 5400$$
$$X_7 = t_0 = (10153/3)$$

Train took 2499 milliseconds and produced answer

$$X_1 = d_3 = (2025/4)$$
$$X_2 = d_1 = 2025$$
$$X_3 = d_2 = (-10097/4)$$
$$X_4 = t_3 = 1350$$
$$X_5 = t_2 = (-10097/3)$$
$$X_6 = t_1 = 5400$$
$$X_7 = t_0 = (10153/3)$$

Solving \(2 ^ (2 \cdot x + 8) - 32 \cdot 2 ^ x + 1 = 0\) for \(x\)

Homogenized equation is \((2 ^ x) ^ 2 \cdot 2 ^ 8 + 2 ^ x \cdot -32 = -1\)

New equation is \(x^3 - 2 \cdot 256 + x^3 \cdot -32 = -1\)

Using quadratic equation formula
$$x^3 = (1/16) \# x^3 = (1/16)$$ is a solution

Applying substitution
$$x_3 = 2 ^ x$$

$$x_3 = (1/16) \# x_3 = (1/16)$$
\[ 2^x = \frac{1}{16} \]

\[ x = \log_2 \left( \frac{1}{16} \right) \]
\[ \text{(by Isolation)} \]

**Answer is**:

**X1**

**where**:

**X1** = \( x = -4 \)

Pow2can took 1584 milliseconds and produced answer

**X1**

**where**:

**X1** = \( x = -4 \)

Substituting \( x^4 \) for \( x + 1/x \)

\[ x^4 - 56 * x^3 + 89 * x^2 - 56 * x + 12 = 0 \] for \( x \)

Using quadratic equation formula

\( x^4 = \frac{5}{2} \) \# \( x^4 = \frac{13}{6} \) is a solution

Applying substitution

\[ x^4 = x + 1/x \]

**to**:

\[ x^4 = \frac{5}{2} \# x^4 = \frac{13}{6} \]

**gives**:

\[ x + x^{-1} = \frac{5}{2} \# x + x^{-1} = \frac{13}{6} \]

Multiply through by \( x^{-1} \) to set a polynomial

Using quadratic equation formula

\( x = 2 \# x = \frac{1}{2} \) is a solution

Multiply through by \( x^{-1} \) to set a polynomial

Using quadratic equation formula

\( x = \frac{3}{2} \# x = \frac{2}{3} \) is a solution
\( (x = 2 \# x = \frac{1}{2}) \# x = \frac{3}{2} \# x = \frac{2}{3} \) is a solution

**Answer is**:

\( (X1 \# X2) \# (X3 \# X4) \)

**where**:

**X1** = \( x = 2 \)

**X2** = \( x = \frac{1}{2} \)

**X3** = \( x = \frac{3}{2} \)

**X4** = \( x = \frac{2}{3} \)

quatemean 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:portr compiled: 580 words, 0.85 sec.
press:solve consulted 921 words 0.34 sec.
press:isola consulted 300 words 0.13 sec.
press:isola.ax consulted 1674 words 0.67 sec.
press:ineais.ax consulted 756 words 0.34 sec.
press:weaknF consulted 375 words 0.15 sec.
press:collec consulted 186 words 0.06 sec.
press:na1 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:homose.top consulted 1278 words 0.51 sec.
press:homose.rew consulted 2219 words 0.98 sec.
press:homose.trs consulted 3057 words 1.28 sec.
press:homose.msc consulted 2302 words 1.03 sec.
press:nasty consulted 195 words 0.09 sec.
press:semea consulted 426 words 0.20 sec.
press:ineea 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.
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, [J, Soln, X]),

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

ddivides([E1, .., Rest], Scanned, Const*Ans, X) :- lookup(E1, Arg, Ans),
                                                 append(Rest, Scanned, Restofterm),
                                                 check(Arg, Restofterm, Const, X),

ddivides([E1, .., Rest], Scanned, Ans, X) :- ddivides(Rest, [E1, .., 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([C], [C]),
inverse([FL1, ..., RLList], [FLList[(-1), ..., RNes]],
  :- inverse(RList, RNes).

lookup(cos(Ars), Ars, sin(Ars)),
lookup(sin(Ars), Ars, -cos(Ars)),
lookup(tan(Ars), Ars, los(sec(Ars))),
lookup(cot(Ars), Ars, los(sin(Ars))),
lookup(sec(Ars), Ars, los(los(Ars) + tan(Ars))),
lookup(arctan(Ars), Ars, Ars*arsin(Ars) + 1 - Ars^2)/2!((-1)),
lookup(arcsin(Ars), Ars, Ars*arsin(Ars) + 1 - Ars^2)/2!((-1)),
lookup(los(Ars), Ars, Ars*los(Ars) - Ars),
lookup(Ars, Ars, Ars, los(Ars)),
lookup(Ars, Ars, Ars, los(Ars)),
lookup(Ars, Ars, D, Ars, (D+1)^(-1)*Ars^D+1)^((-1)),
  freeof(D, x),
  D\==(-1),
lookup(D^Ars, Ars, (los(D)^((-1)) * D^Ars)^(-freeof(D, x)),
  D\==(-1),

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

recomp1(E, X, L),
recomp1(E, X, L),
recomp1(E, X, L),
recomp1(E, X, L),
recomp1(E, X, L),
recomp1(E, X, L),
recomp1(E, X, L),
recomp1(E, X, L),
recomp1(E, X, L),
recomp1(E, X, L).

diffwrt1(C*X, C*X*los(C)^(-1), X)^(-freeof(C, X)),
  diffwrt1(los(X), X^(-1), X)^(-freeof(C, X)),
  diffwrt1(tan(X), sec(X)^2*X)^(-freeof(C, X)),
  diffwrt1(cot(X), -cosec(X)^2*X)^(-freeof(C, X)),
  diffwrt1(los(X), sec(X)^2*X*tan(X))^(-freeof(C, X)),
  diffwrt1(arsin(X), (1-X^2)^(-2((-1)))*X)^(-freeof(C, X)),
  diffwrt1(cosec(X), -cos(X)*cosec(X)^2*X)^(-freeof(C, X)),
  diffwrt1(artan(X), (1+X^2)^(-1), X)^(-freeof(C, X)).
tsimpax(-U,(1)*U),
tsimpax(los(e),1),
sensym(var,val1),
/* Evaluation routines */

\[ \text{eval}\left(\text{Exp}_1,\text{Exp}_2\right) \rightsquigarrow \text{atomic}\left(\text{Exp}_1\right), \]

\[ \text{eval}\left(\text{Exp}_1,\text{Ans}\right) \]
\[ \left\{ \begin{array}{ll}
\text{nonvar}\left(\text{Exp}_1\right), & \text{Exp}_2 = +, [\text{P}\left[\text{Arss}\right]], \\
\maplist\left(\text{eval},\text{Arss},\text{Narss}\right), & \text{NexP} = +, [\text{P}\left[\text{Narss}\right]], \\
\text{checklist}\left(\text{Integer},\text{Narss}\right) \rightarrow \text{eval}\left(\text{NexP},\text{Ans}\right) \end{array} \right. \]

\[ \text{eval1}\left(\text{A}\leftarrow\text{B},\text{true}\right) \rightarrow \text{A}=\text{B}, \]

\[ \text{eval1}\left(\text{A}\leftarrow\text{B},\text{false}\right) \rightarrow \text{A}\neq\text{B}, \]

\[ \text{eval1}\left(\text{A}=\text{B},\text{Ans}\right) \rightarrow \text{eval1}\left(\text{A}=\text{B}\right), \]

\[ \text{eval1}\left(\text{A}=\equiv\text{B},\text{true}\right) \rightarrow \text{A}=\equiv\text{B}, \]

\[ \text{eval1}\left(\text{A}=\equiv\text{B},\text{false}\right) \rightarrow \text{A}=\neq\text{B}, \]

\[ \text{eval1}\left(\text{A}>\text{B},\text{true}\right) \rightarrow \text{A} > \text{B}, \]

\[ \text{eval1}\left(\text{A}>\text{B},\text{false}\right) \rightarrow \text{A} < \text{B}, \]

\[ \text{eval1}\left(\text{A}<\text{B},\text{Ans}\right) \rightarrow \text{eval1}\left(\text{B}>\text{A},\text{Ans}\right), \]

\[ \text{eval1}\left(\text{A} > \equiv\text{B},\text{Ans}\right) \rightarrow \text{eval1}\left(\text{B}>\text{A},\text{Ans}\right), \]

\[ \text{eval1}\left(\text{A}=\text{B},\text{Ans}\right) \rightarrow \text{eval1}\left(\text{B}=\text{A},\text{Ans}\right), \]

\[ \text{eval1}\left(\text{A}+\text{B},\text{Ans}\right) \rightarrow \text{Ans is A}+\text{B}, \]

\[ \text{eval1}\left(\text{A}-\text{B},\text{Ans}\right) \rightarrow \text{Ans is A}-\text{B}, \]

\[ \text{eval1}\left(\text{A}*\text{B},\text{Ans}\right) \rightarrow \text{Ans is A}*\text{B}, \]

\[ \text{eval1}\left(\text{A}/\text{B},\text{Ans}\right) \rightarrow 0\ is\ A\ mod\ B, \]

\[ \text{eval1}\left(\text{A}/\text{B},\text{Ans}^{-}\left(1\right)\right) \rightarrow 0\ is\ B\ mod\ A, \]

\[ \text{eval1}\left(\text{A}/\text{B},\text{A}_1/\text{B}_1\right) \rightarrow \text{scd}\left(\text{A},\text{B},\text{D}\right), \]

\[ \text{eval1}\left(-\text{A},\text{Ans}\right) \rightarrow \text{Ans is } -\text{A}, \]

\[ \text{eval1}\left(\text{A}+\text{B},\text{Ans}\right) \rightarrow \text{Res is A}+\text{B}, \]

\[ \text{eval1}\left(\text{A}--\text{B},\text{Ans}\right) \rightarrow \text{Res is A}-\text{B}, \]

\[ \text{eval1}\left(\text{A}^{\equiv}\text{B},\text{Ans}\right) \rightarrow \text{intexp}\left(\text{A},\text{B},\text{Ans}\right), \]

\[ \text{eval1}\left(\text{sart}(\text{X}),\text{Ans}\right) \rightarrow \text{sarteval}(\text{X},\text{Ans}), \]
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 */
sgcd(A,0,A) :- !.
sgcd(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)
    :- F is M-1,
       intexp(L,F,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)).

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

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).
\begin{verbatim}
\texttt{\textbf{:- nesative(S),}}
\texttt{eval(-1*S,S1),}
\texttt{sineval1(X,S1),}

\texttt{arccoseval(C,X)}
\texttt{:- non_nes(C),}
\texttt{!,}
\texttt{sineval1(Y,C),}
\texttt{X is 90-Y.}

\texttt{arccoseval(C,X)}
\texttt{:- nesative(C),}
\texttt{eval(-1*C,C1),}
\texttt{sineval1(Y,C1),}
\texttt{X is 90-Y.}

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

\texttt{natnum(X) \texttt{-} intese(X), X>0.}

\texttt{odd(N) \texttt{-} eval(N,N1), natnum(N1), 1 is N1 mod 2.}
\texttt{even(N) \texttt{-} eval(N,N1), natnum(N1), 0 is N1 mod 2.}

/*====================================================================*/

\texttt{\textbf{wordsin(\[J,\[J) \texttt{-} !,}}

\texttt{\textbf{wordsin(\[HD:TL],List)}}
\texttt{:- wordsin(HD,L1),}
\texttt{wordsin(TL,L2),}
\texttt{union(L1,L2,List),}
\texttt{!,}

\texttt{\textbf{wordsin(E,List)}}
\texttt{:- atomic(E) \texttt{-} fail}
\texttt{\textit{\(\texttt{\textbf{fail}}\) \texttt{-} E = \texttt{\textbf{[P:Aras]}},}}
\texttt{wordsin(Aras,List),}
\texttt{!,}

\texttt{\textbf{wordsin(E,[E])}}
\texttt{:- intese(E),}
\texttt{!,}

\texttt{\textbf{wordsin(E,[E]) \texttt{-} atom(E).}}
\end{verbatim}
/*Finds types of variables in PRESS*/
/*Alan Bundy 19.12.79*/

in(X,[L,B,T,R]) :-
  interval(X,[Lx,Bx,Tx,Rx]), !,
  less_than([Bx,Lx]), less_than([Tx,Rx],[T,R]),
  opposite(Lx,L), opposite(Rx,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_zero(X) :- interval(X,[L,B,T,R]), !, less_than([0,open],[B,L]),
  non_zero(X^N) :- negative(N), !. /*temporary patch*/
  
  less_than([O,open],[B,L]), obtuse(X) :- interval(X,[L,B,T,R]), !,
  less_than([90,open],[B,L]), less_than([T,R],[90,open]).

non_reflex<X> :- interval(X,[L,B,T,R]),
  less_than([O,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], [Ly,By,Ty,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,[Rx,0,inf,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,[Rx,0,1,Lx]), 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]) :-
    inteser(P), inteser(Q), !,
    whole_part(P,Q,N), N1 is N+1.

interval(X,[closed,X,X,closed]) ; inteser(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 s as measure(s,32,ft/sec^2) */
interval(s,[open,1,inf,open]) ; !.

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

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

/* Find lower bounding inteser 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) ; inteser(X), inteser(Y), !, Z is X+Y.

plus(X,Y,Z) ; either_infinite(X,Y,Z), !.

/* Minus a boundary */
minus(X,Y) ; inteser(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) ; inteser(X), inteser(Y), !, Z is X*Y.

mult(X,Y,Z) ; either_infinite(X,Y,\n)
sign(X,Sx), sign(Y,Sy), signed_inf(Sx,Sy,Z), !.

/* Orderings of boundaries (assumes ranges are consecutive)*/
less_than([X,Mx],[Y,My]) :-
    comb(Mx,My,M), less_than(X,Y,M), !.
less_than(X,Y,M) :-
    integer(X), integer(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*/
    ther_infinite(inf,Y,inf) :- !,
    ther_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).

/*sign of number*/
    sign(-inf, '−'),
    sign(inf, '+'),
    sign(Int, '+') :- integer(Int), Int>0,
    sign(Int, '−') :- integer(Int), Int<0.

/*Produce inf of right sign*/
signed_inf(S,S,inf) :- !,
signed_inf(S1,S2,-inf).

/* Pick upper and lower bounds from list*/
pick_upper([Hd|Tl],Hd) :- !.
pick_upper([Hd|Tl],Ans) :-
    pick_upper(Tl,Up),
    (less_than(Up,Hd) -> Hd=Ans; Up=Ans), !.
pick_lower([Hd|Tl],Hd) :- !.
pick_lower([Hd|Tl],Ans) :-
pick_lower(Tl,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) :- !.
/*recursive step*/
unionlist([Range1 I Rest] , Range) :-
   unionlist(Rest,Range2) , !,
   combine(Range1,Range2,Range) ; combine(Range2,Range1,Range)).
/*Combine two intervals, if it's 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 |
| COLAX 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|
| JALS   | 45   | 20-Feb-80|
| JAR 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 |
| IORDOUT 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 |
| UTIL PL | 10   | 8-Mar-80 |
| JPS    | 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-80|
End of save set on
System Unknown monitor APRIO
Unknown BPI 9 track OM-Kt-Jan--41 00:00:00 BACKUP tape format 1
Tape number 1

Start of save set PAPERS on
System Unknown monitor APRIO
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 APRIO
Unknown BPI 9 track OM-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('base.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('chansel.pl'),
   consult('tidy.pl'),
   consult('isola.pl'),
   consult('sort.pl'),

/* patches to PRESS */
\*- reconsult('collax.pat'),
   reconsult('decmpat.pat'),
   reconsult('interypat.pat'),
   reconsult('least.pat'),
   reconsult('poly.pat'),
/* POWERFUL ALGEBRAIC MATCHER */

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

/* COLLECTION ROUTINE THAT USES POWERFUL MATCHER */

try_hard_to_collect(X,Old,New1) :-
/* try to use powerful matcher to collect */
solve(Eqn,Unknown,Ans),
flas(try_hard,Old,true),
solve(Eqn,Unknown,Ans),
flas(try_hard,_,Old),
flas(try_hard,_,false).

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

/* powers 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,FRule),
match(Rule,Rule_X,FRule),
match(NewRule,FRule),
/* 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,FRule),
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('trvins change of unknown of the form %P

', [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 chan•e of unknown is %c
', CSubstl&Subst2J,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_matchinS(Expr,Expr_X),
try_new_substitution(Term=Expr_X , New_Subst , AddedPatternVars), ID),
trace('n

findins advice to try new substitution %P
' , [New_Subst],4),
trace(' in Place of old substitution %P
', [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('trwins to aPPl~ rule %P -> XP
 to %P

', [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) :-
\texttt{expr(Expr,E), expr(Pattern,E),}\\ \texttt{null_transform(Transform),}\\ \texttt{trace('trivially matching \%P and \%P\n',[E,E],4),}\\ \texttt{!}.\\

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

\texttt{/* HARD MATCH \- USE MEMO */}\\ \texttt{match(Expr,Pattern,Transform) :-}\\ \texttt{\hspace{1em}memo(hard_match(Expr,Pattern,Transform)),}\\ \texttt{\hspace{1em}!}.\"
PROCEDURES FOR NON-TRIVIAL MATCHES /

There are two ways of accomplishing a hard match:

by matching subexpressions ("match1") -- this may involve
bas 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
    returning %P
    ', [E,P,Transform],4).

hard_match(Expr,Pattern,Transform) :-
expr(Expr,E), expr(Pattern,P),
    trace('match failed on %P and %P
    ', [E,P],4),
    match2(Expr,Pattern,Transform),
    trace('solving for a variable succeeded in matching expression %P
    ', [E],4),
    trace(' and pattern %P
    returning %P
    ', [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('tries to match expression %P and pattern %P
    ', [E,P],4),
    /* Expr and Pattern must have the same functor */
    functor(Expr,F,N); functor(Pattern,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, El),
  subPart(Pattern, J, P1),
  match(El, P1, T1),
  apply_transform(T1, Pattern, P2),
  J 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, [J]),
  !,
  fail.

tch2(Expr, Pattern, Transform) :-
  pattern_var(Pattern, V),
  expr(Expr, E),
  expr(Pattern, P),
  contains(V, P),
  trace('trying to solve for a variable
n'),
  trace('calling equation solver to solve for %P in %P
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, UEExpr), unknown(Pattern, UPattern),
  (UPattern=UEExpr -> freeof(UEExpr, Soln), Transform=T1 ;
  V=UPattern -> contains(UEExpr, Soln), change_unknown(UEExpr, T1, Transform) ;
  freeof(UEExpr, 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 bass for expression %p

    '([Name,E],4),
  trace('    and pattern %p

    '([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 bass and trying to match them

    '([E,P],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 bass

    '([RR],4),
  trace('    yielding %p

    '([P],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,Unknown), V\=Unknown,
  /* 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 basic identity element %P

/* 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

/* 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

trace('by applying a function to each side of the rule
(expr(T,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),
sensym(q,Q), EE=bas(Op,-),
(Op= + -> New_Subst = (Old_Term=X+ -1*Q) ;
  Op= * -> New_Subst = (Old_Term=X*Q^ -1)),
recordz( when_matchins(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.
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

The expression description is a tuple:

```
expr_description(Expr, Top, Unknown, PatternVars, AddedPatternVars, Owners)
```

- **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**
  ```prolog
  expr( expr_description(Expr, Top, Unknown, PatternVars, AddedPatternVars, Owners) ; Expr ) :- !.
  ```

- **Top**
  ```prolog
  top( expr_description(Expr, Top, Unknown, PatternVars, AddedPatternVars, Owners) ; Top ) :- !.
  ```

- **Unknown**
  ```prolog
  unknown( expr_description(Expr, Top, Unknown, PatternVars, AddedPatternVars, Owners) , Unknown ) :- !.
  ```

- **Symbols**
  ```prolog
  symbols( expr_description(Expr, Top, Unknown, PatternVars, AddedPatternVars, Owners) ; Symbols ) ;
  wordsin(Top, Symbols), !.
  ```

- **Pattern vars**
  ```prolog
  pattern_vars( expr_description(Expr, Top, Unknown, PatternVars, AddedPatternVars, Owners) ; PatternVars ) :- !.
  ```

- **Owners**
  ```prolog
  owners( expr_description(Expr, Top, Unknown, PatternVars, AddedPatternVars, Owners) ; Owners ) :- !.
  ```

## Replacing Parts

- **New_expr**
  ```prolog
  new_expr( expr_description(Expr, Top, Unknown, PatternVars, AddedPatternVars, Owners) ; New_Expr ) ;
  expr_description(New_Expr, Top, Unknown, PatternVars, AddedPatternVars, Owners) ) ;
  ```

- **New_owners**
  ```prolog
  new_owners( expr_description(Expr, Top, Unknown, PatternVars, AddedPatternVars, Owners) ;
  New_Owners ) ;
  expr_description(Expr, Top, Unknown, PatternVars, AddedPatternVars, New_Owners) ) ;
  ```

- **New_unknown**
  ```prolog
  new_unknown( expr_description(Expr, Top, Unknown, PatternVars, AddedPatternVars, Owners) ;
  New_Unknown ) ;
  expr_description(Expr, Top, New_Unknown, PatternVars, AddedPatternVars, Owners) ) ;
  ```

- **New_pattern_vars**
  ```prolog
  new_pattern_vars( expr_description(Expr, Top, Unknown, PatternVars, AddedPatternVars, Owners) ;
  New_PatternVars ) ;
  expr_description(Expr, Top, Unknown, New_PatternVars, AddedPatternVars, Owners) ;
  ```

- **Intersect**
  ```prolog
  /* remove non-existent vars from AddedPatternVars */
  intersect(AddedPatternVars, New_PatternVars, 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 */
pty( expr_description(bas(_,[[]]),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) ) !- 
args(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 = bas(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( bas(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,+), pair(other,+), or pair(first,sin). "first" or "other" indicates whether the term being considered is the first element of a bas being matched, */
add_owner([pair(_,Op)|Rest], bas(Op,_), [pair(other,Op)|Rest]) :- !.
add_owner( Owners, bas(Op,_), [pair(first,Op)|Owners] ) :- !.

/* procedures for other expressions */
adddo_owners( 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 being evaluated */

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

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) being
     generated and recorded. In this case, the previously recorded results
     will be re-generated 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(Expr,EE), expr(P,PP),
  power(Expr,EE,Unknown,N1),
  power(PP,P,Unknown,N2),
  (Unknown=Unknown, 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(Expr,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),
  pickl(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).

pickl( bas(OP,[Term]) , Unknown , Term , bas(OP,[]) ) !- !,
~ickl( bas(OP,[T1,T2|Others]) , Unknown , Term , bas(OP,[TBad|Rest]) ) !- 
  pick_from_pair(T1,T2,Unknown,TGood,TBad),
  pickl( bas(OP,[TGood|Others]) , Unknown , Term , bas(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) !-
power(X,Unknown,N) :-
    atomic(X),
    !,
    N=1.

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

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

power([HT],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=.,Expr([Arss]),
    power(Arss,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(Expr1,Expr2),
    freeof(New_Unknown,Expr1),
    !,
    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 inteser 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 are data structures that represent functions, substitutions, and possibly a change of unknown to be applied to an expression.

Format:
transform(FunctionList,SubstitutionList,New_Uknown)
New_Uknown is "false" if the unknown isn't being changed

CREATING TRANSFORMS

null_transform( transform([],[],false) ).
make_substitution_transform( S, transform([],[],false) ).
make_function_transform( OP, Expr,
                     transform( [function(OP,Expr)], [], false ) ).
make_inv_function_transform( t, Expr,
                            transform( [function(t,InvExpr)], [], false ) ;
                            tids(Expr~ -1,InvExpr) ).
change_unknown( New_Uknown , transform(FList,SList,false) ,
                transform(FList,SList,New_Uknown) ).
apply_transform( transform(FList,SList,New_Uknown) , Descr1 , Descr6 ) ;
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_Uunknown=false -> Descr6=Descr5 ;
      new_unknown(Descr5,New_Uunknown,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(+,Ars) or function(*,Ars), */

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

/* These clauses handle bacs. If the function has the same operator as the
bas, just add the new argument to the bas. If the function's operator
distributes over the bas, apply the function to each element.
Otherwise fail. */
apply_function( function(Op,Func_Ars), bas(Op,Arsss),
    bas(Op,[Func_Ars|Arsss]) ) !- !,
apply_function( function(Func.Op,Func_Ars), bas(Bas.Op,[J]),
    bas(Bas.Op,[J]):-
    distributes(Func.Op,Bas.Op), !.
apply_function( F, bas(Bas.Op,[Ars1|Arsss]),
    bas(Bas.Op,[New_Ars1|New_Arsss]) ) !-
    F=function(Func.Op,-), !, /* fail completely if Func.Op doesn't distribute */
    distributes(Func.Op,Bas.Op),
    apply_function(F,Ars1,New_Ars1),
    apply_function(F,bas(Bas.Op,Arsss),bas(Bas.Op,New_Arsss)).

/* now clauses to handle expressions not in bas form */
apply_function( function(+,Ars), Expr, Expr+Ars),
apply_function( function(*,Ars), Expr, Expr*Ars),

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 */

porray( bas(0p,[A]) ) :-
    write(A), !.

porray( bas(0p,[A|Rest]) ) :-
    write(A),
    write(0p),
    porray( bas(0p,Rest) ), !.

porray( bas(0p,[]) ) :-
    write('<empty bag>'), !.

porray( transform(FL1st,SList,New_Uknown) ) :-
    writef('transform:
'),
    porray_functions(FL1st),
    porray_subst(SList),
    (New_Uknown=false -> true ;
     writef('
 change unknown to %t\n',New_Uknown) ), !.

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

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

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

porray(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,Arss),
new_expr(Descr1,bas(Op,Arss),Descr2),
!

to_bas(Op,Descr1,Descr2) :-
expr(Descr1,ExPr),
neg_expr(Descr1,bas(Op,[ExPr3],Descr2),
!

rom_bas(Descr1,Descr2) :-
expr(Descr1,bas(Op,A1)),
rev(A1,Arss),
recomp(ExPr,[Op|Arss]),
neg_expr(Descr1,ExPr,Descr2),
!

or_to_list(H|T,[H|TT]) !- !
or_to_list(T,TT),
or_to_list(ExPr,[ExPr]) !- !,
/* PROCEDURES FOR EXTRACTING FEATUIES FROM EXPRESSIONS

   callins 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|Arses],
   features(Arses,U,FArses),
   Features=[], [Op|FArses], !.
/* Procedure to instantiate all the variables in a rule, 
call: instantiate(Rule, BadNames, PatternVars)
Uses names u,v,w if possible; otherwise use sensum'd names */

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

bind_list([], BadNames) !:- !.

bind_list([SI], BadNames) !:-
    preferred_symbol(S),
    not member(S, BadNames), !,
    append(BadNames, [S], Badi),
    bind_list(T, Badi),

preferred_symbol(u),
preferred_symbol(v),
preferred_symbol(w),
preferred_symbol(X) !:-
sensum(a, X).
/* tests for powerful matcher */

\l~ tlim(5),

quadratic \l~ try_hard_to_solve( a*x^2+b*x+c=0, x, Ans),
cubic \l~ try_hard_to_solve( a*x^3+b*x^2+c*x+d=0, x, Ans),
specialized_cubic \l~ try_hard_to_solve( z^3+h*z+s=0, z, Ans),
distrib1 \l~ try_hard_to_solve( a*x+x = x^2, x, Ans),
distrib2 \l~ try_hard_to_solve( a*x-x = x^2, x, Ans),
distrib3 \l~ try_hard_to_solve( x+x = x^2, x, Ans),
trig1 \l~ try_hard_to_solve( a*sin(x)+b*cos(x)=c, x, Ans),
chance1 \l~ try_hard_to_solve( 5^2(2*y) + 5^y + 5 = 0, y, Ans),
chance2 \l~ try_hard_to_solve( 5^2(2*y) - 5^(y+1) + 6 = 0, y, Ans),
chance3 \l~ try_hard_to_solve( 3^2(2*y) - 2*3^(y+2) + 81 = 0, y, Ans),

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

chance4 \l~ try_hard_to_solve( (3*y^2-2*y-2)^2 = 21*y^2 - 14*y - 20, y, Ans),
chance5 \l~ 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,Ean,Conditions) :-
    trace('trying to learn to solve %P for %P

',Ean,Unknown,1),
/* solve the equation using powerful matcher */
    try_hard_to_solve(Ean,Unknown,Al),
/* convert Ean to the normal form */
    C=.[Form,Unknown,Ean,Norm_Ean],
    call(C),
/* change symbols in equation etc. to variables */
    wordsin(A1,Symbols),
    variablize([Ean,Norm_Ean,Unknown,A1,Symbols,Conditions],
                [EanVar,NormVar,UnknownVar,AnsVar,SymbVars,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

',Ean,Unknown,1),
    asserta(
        solve1(E1,UnknownVar,Ans) !-
            Convert,
            match(E2,NormVar),
            trace('using learned method for solving equations of the form %P

',Ean,1),
            CondVars,
            tidy(AnsVar,Ans),
            !).

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

variablize(A,[H|T],A) !-
/* 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],[H!T]) !-
  var_subst(Var=Const,H,H),
  var_subst(Var=Const,T,T), !.

var_subst(Var=Const,Old,New) !-
  Old=.,[Op!ArsS],
  var_subst(Var=Const,ArsS,NArsS),
  New=.,[Op!NArSs], !.

polynomial(X, L=R, poly_ean(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 */
  'general(X,Exp,Exp) !- !.

/* add a clause for matching stuff in polynomial normal form */
  asserta(
    match(poly_ean(X,L1),poly_ean(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)1R2]) #
/* N1< N2 */ !, C2=0, poly_match([pair(N1,C1)1R1],R2) },
!.
learn_quad :- learn_to_solve( polynomial, x, a*x^2+b*x+c=0, non_zero(a) ),
atest1 :- solve( x^2=9 , x, Ans ),
atest2 :- solve( x^2-x-6=0, x, Ans ),
atest3 :- solve( (x+3)*(x+2)= 6, x, Ans ),

learn_trig :- learn_to_solve( general, x, a*sin(x)+b*cos(x)=c, non_zero(a) ),
test1 :- solve( 1*sin(x)+0*cos(x)=1 , x, Ans ),
test2 :- solve( 1*sin(x)+1*cos(x)=2^ (2^ -1) , x, Ans ).
/* 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
',[]),
/* 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] ; [] , 0 ),
match(EDescr,G,Transform),
/* substitute for a,b,c,x in solution to general quadratic */
Sort = (b_zzz^2+ -4*a_zzz*c_zzz)(2^-1),
Sol1 = (x_zzz=(- -l*b_zzz+Sort)*(2*a_zzz)^ -1),
Sol2 = (x_zzz=(- -l*b_zzz+(- l*Sort))*(2*a_zzz)^ -1),
made_description( Sol1 , x_zzz ; [] , [] , Sol1Descr ),
made_description( Sol2 , x_zzz ; [] , [] , Sol2Descr ),
apply_transform(Transform,Sol1Descr,New1Descr),
apply_transform(Transform,Sol2Descr,New2Descr),
expr(New1Descr,New1); expr(New2Descr,New2),
trace('
applying transform to solution to quadratic equation yielding %e
', [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 base flushing procedure to combine like items to powers - put in before other procedures */
\ [+ asserts(\n    f12([X^L],New) :- twofrom(L,X1^A,X2^B,R), match(X1,X2), tidy(A+B,C),
       !, f12([X^C[R]],New)\n  )),
/* 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. */

:- asserts((
islax( 1 , sin(U)=V , U=arcsin(V) , particular_solution(U) )
)),

:- asserts((
islax( 1 , cos(U)=V , U=arccos(V) , particular_solution(U) )
)),

:- asserts((
islax( 1 , tan(U)=V , U=arctan(V) , particular_solution(U) )
)),

:- asserts((
islax( 1 , cosec(U)=V , U=arccosec(V) , particular_solution(U) )
)),

  asserts((
islax( 1 , sec(U)=V , U=arcsec(V) , particular_solution(U) )
)),

:- asserts((
islax( 1 , cot(U)=V , U=arccot(V) , particular_solution(U) )
)),
/* SQUARE ROOT EVALUATION */

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

move_powers(X,Power,J,1,X) :-
  integer(J,Power,A),
  A>X,
  remove_powers(X,Power,J,IPart,Residue) \- 
    integer(J,Power,A),
    0 is X mod A,
    X1 is X/A,
    remove_powers(X1,Power,J,IPart,Residue),
    IPart is IPart*J,
  !.

remove_powers(X,Power,J,1,IPart,Residue) :-
  J1 is J+1,
  remove_powers(X,Power,J1,IPart,Residue),
  !.
collax( w, u*w+v*w, (w+v)*w )

/* collax( w, w+v*w, (w+1)*w ), */
/* collax( w, v*w+w, (w+1)*w ), */
/* collax( w, (w+v)*w, (w+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^u*w^v, w^((u+v)) ),

collax( w, w^u*w^v, 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,[+|C],A) :- !.

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

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

decomP(A+(B+C),L) :- !, decomP(A+B,C),L.
decomP(A+(B+C),[+|B+B],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.
on_reflex(X) :- !, fail.
on_nes(X) :- !, fail.
/* Exp is a least dominating expression of X (i.e., 2 arss contain X) */
/* This patches the old "least_dom" by converting to bas representation */

least_dom(X,Exp) !-
    decomp(Exp,[F!Arss]),
    sublist(contains(X),Arss,XArss),
    length(XArss,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 (as 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 */

:- assert(
    eval1(los(Base,X),Ans) :-
    loseval(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), !
).

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

seval(Base,X,IPart,Fraction) :-
  0 is X mod Base,
  X1 is X/Base,
  loseval(Base,X1,II,Fraction),
  IPart is II+1,
!.

loseval(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).
### Predicate Cross Reference Listing

Neil Davey's Identity Assimilator

<table>
<thead>
<tr>
<th>Predicate</th>
<th>File</th>
<th>Called By</th>
</tr>
</thead>
<tbody>
<tr>
<td>append(3)</td>
<td>utility</td>
<td>foosin(2)</td>
</tr>
<tr>
<td>arsn(3)</td>
<td>scope</td>
<td>position(3)</td>
</tr>
<tr>
<td>common_ancestor(3)</td>
<td>scope</td>
<td>distance(3)</td>
</tr>
<tr>
<td>distance(3)</td>
<td>scope</td>
<td>use_for_attraction(3)</td>
</tr>
<tr>
<td>foosin(2)</td>
<td>scope</td>
<td>use_for_normalize1(2)</td>
</tr>
<tr>
<td>footerm(3)</td>
<td>scope</td>
<td>use_for_normalize2(2)</td>
</tr>
<tr>
<td>forall(2)</td>
<td>utility</td>
<td>use_for_normalize2(2)</td>
</tr>
<tr>
<td>height(2)</td>
<td>scope</td>
<td>height(2)</td>
</tr>
<tr>
<td>inner(1)</td>
<td>undefined</td>
<td>use_for_normalize2(2)</td>
</tr>
<tr>
<td>member(2)</td>
<td>utility</td>
<td>use_for_isolation(2)</td>
</tr>
<tr>
<td>nmember(3)</td>
<td>utility</td>
<td>arsn(3)</td>
</tr>
<tr>
<td>occ(3)</td>
<td>utility</td>
<td>use_for_collection(2)</td>
</tr>
<tr>
<td>pair(3)</td>
<td>scope</td>
<td>use_for_attraction(3)</td>
</tr>
<tr>
<td>partition(5)</td>
<td>scope</td>
<td>asort(4)</td>
</tr>
</tbody>
</table>

---

*PROLOG CROSS REFERENCE LISTING*
Neil Davey's Identity Assimilator

position(3) scope use_for_attraction(3) position(3)
assoc(3) scope assoc(3) assoc(4)
assoc(4) scope assoc(3) assoc(4)
soln(2) scope use_for_isolation(2) soln(2)
sugarlist(1) undefined sugarlist(2)
sugarist(2) scope use_for_normalize1(2)
union(3) utility wordsin(2)
use(1) scope use(1)
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) use_for_isolation(2)
use_for_tidy2(2) use_for_normalize1(2)
use_for_tidy2(2) use_for_normalize2(2)
Identities of Algebra for use as tests for Davey's SCOPE program

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=sqrt(v) & u=-sqrt(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
11 30.

\(u+v=w \Rightarrow u=w-v\) has been used as an isolation axiom on the variable \(u\).

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

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

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

\(u^2=v \Rightarrow u=\sqrt{v}*u=-\sqrt{v}\) has been used as an isolation axiom on the variable \(u\).

\(u^2=v \Rightarrow u=\sqrt{v}*u=-\sqrt{v}\) has now been used in as many cases as possible.

\(\sin(u)\Rightarrow u=n*180+\arcsin(v)\) has been used as an isolation axiom on the variable \(u\).

\(\sin(u)\Rightarrow u=n*180+\arcsin(v)\) has now been used in as many cases as possible.

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

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

\(u^n*(v+w)=(u^n*v)+u^n*w\) 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

: bal 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('It 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),
use_for_isolation(L->R,U) :-
  wordsin(L,S),
  member(U,S),
  soln(R,U),
  assert(isolax(L->R,U)),
  writef('%s has been used as an isolation axiom on the variable %s.

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('%s has been used as a collection axiom from left to right

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('%s has been used as an attraction axiom from left to right


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

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,GR),
  (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(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));
L=U,_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 E AS A NORMALIZE AXIOM

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

/* TRY THE FIRST TYPE OF NORMALIZE AXIOM */

use_for_normalize1(E,Foo) :-
  axiom(E,L,R),
  suszyist(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(normalize(R,L,Foo)),
writef('Xt has been used as a normalize axiom from right to left
on the function Xt.

TRY THE SECOND TYPE OF NORMALIZE AXIOM */

use_for_normalize2(E,Foo) :-
  axiom(E,L,R);
  inner(Foo);
  (foosin(L,SL),
   occ(Foo,SL)).
foosin(T,Foo,R),
foral(foosin(S,Foo,R), check1(S,T)).
assert(normalize(L,R,Foo)),
writef('Xt has been used for a normalize axiom from
left to right on the function Xt.

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:Arss].
Position(X,E,L) :-
arsn(N,E,T),
L=[N:L],
position(X,T,L).

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

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

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

sugarist(E,foo) :-
fooisin(E,S),
susarlist(D),
asort(S,L,D),
L=[foo:Rest].

fooisin([],[]) :- !.
fooisin([F:R],S) :- !,
fooisin(F,S1),
fooisin(R,S2),
append(S1,S2,S).

fooisin(X,[]) :-
atomic(X), !.
fooisin(Term,S) :-
Term=..[Functor:Arss],
fooisin(Arse,S1),
S=[Functor:S1].

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

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

partition([X:],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,[[X:L2]],D) :-
nmember(X,D,NX),
nmember(Y,D,NY),
NX>NY,
Partition(L,Y,L1,L2,D),
Partition([X],[Y],[D],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=..[_:Arss],
    member(X,Arss),
    footerm(T,Foo,X).

heisht(T,H) :-
    atomic(T),
    H is 0.

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

check1(X,Y) :- heisht(X,HX),
            heisht(Y,HY),
            HX =< HY,