memo6@.ai.wvu.2003. Available from `http://tim.menzies.com/pdf/03lib.pdf`

LIB: Commonly used predicates

Tim Menzies

Lane Department of Computer Science, West Virginia University, PO Box 6109, Morgantown, WV, 26506-6109, USA;
`http://tim.menzies.us`; `tim@menzies.us`

Wp ref: `~menzies/src/pl/prod/lib.pl`, February 23, 2003.

Abstract Commonly used Prolog predicates.

Contents

List of Figures

1 Change log

Addeded: **`ensure/1`** Ensure that some assertion exists.

2 Installation

```
:- load_files([lib0  % pre-load actions
              ,lib1  % predicates
              ,lib2  % start-up commands
              ],[silent(yes),if(changed)]).
```

3 Pre-load actions

3.1 Operators

Define an operator to handle numberic ranges **`Min to Max`**.

```
:- op(1,xfx, to).
```

3.2 Flags

Define an **`inf`**inity function.

```
:- arithmetic_function(inf/0).
```

Define a **`rand`**om function.

```
:- arithmetic_function(rand/0).
```

Define a **`rand(Min,Max)`** function.

```
:- arithmetic_function(rand/2).
```

Define a **`rand(Min,Max,Mean)`** function.

```
:- arithmetic_function(rand/3).
```

Define a **normal** function.

```
:- arithmetic_function(normal/2).
```

Define a **beta** function.

```
:- arithmetic_function(beta/1).
```

Define a **gamma** function.

```
:- arithmetic_function(gamma/2).
```

Add a “left-justify” command to **format**.

```
:- format_predicate('>',padChars(_,_)).
```

Add a “right-justify” command to **format**.

```
:- format_predicate('<',charsPad(_,_)).
```

Add a “print squiggles” command to **format**.

```
:- format_predicate('S',squiggle(_,_)).
```

Define a predicate for the lookup tables.

```
:- discontiguous lookUp1/4.
:- multifile lookUp1/4.
:- index(lookUp1(1,1,1,0)).
```

3.3 Hooks

A hook for lookup tables.

```
term_expansion(Table = ColsRows , Out) :-
    nonvar(ColsRows),
    ColsRows = (Cols+Rows),
    lookUpTable(Table=Cols+Rows,Out).
```

4 Predicates

4.1 Code to demonstrate predicates

4.1.1 Define demos

```
eglib :-
    forall(member(G,[
            egwrites, egdeletes,   egmaths,  eglookup,
            eginc,
            egrands,  egbeta,      egnormal, eggamma,
            egformat, eginc,       egdist,
            egbarsNormal, egbarsBeta,
            egbarsGamma1, egbarsGamma2, egbarsGamma3,
            egnormalize, egchars,  egtidy,
            egdemand, egtimes
            ]),
        demos(G)).
```

4.1.2 Processing demos To demo our code, we need to:

– Write a demo predicate that shows off our code in action. In PROD, these predicates are named `egXXX/0`. Include with this predicate, a pointer to the output; e.g.

```
egXXX :- % See \fig{egXXX.spy}
     ...
```

– Trap the output to a file. This is accomplished using the `demos/1` predicate shown below. The goal `demos(egXXX)` generates a file `egXXX.spy`.
– Include that file. This is accomplished using the following LaTeX magic:

```
\SRC{egXXX.spy}{From \tion{egXXX/1}.}
```

Note the call to `\tion{egXXX/1}`. Sections can be referenced symbolically when (e.g.) `\label{sec:egXXXX/1}` is added on the first line after a heading definition. Once this has been done, then `\tion{egXXX/1}` will be typset as a reference to the relevant section.

After all that, then:

– The output of the demo will be shown in the document,
– The demo predicate will include a pointer to the figure,
– The caption of the figure will include a pointer to the section in the text that generated it.

Most of the demonstrations in this file use this approach.

4.1.3 **demos(+Goal)** `Demos/1` runs a goal `G` and catches the output to the file `G.spy`. Also, just so you know what is going on, it runs the goal `G` a second time and sends the output to the screen.

```
demos(G) :-
    sformat(Out,'~w.spy',G),
    freshFile(Out),
    tell(Out),
    format('% output from '':- demos(~w).''\n\n',G),
    T1 is cputime,
    ignore(forall(G,true)),
    T2 is (cputime - T1),
    format('\n% runtime = ~w sec(s)\n',[T2]),
    told,
    nl,write('\n%------------------------------\n'),
    format('% output from '':- demos(~w).''\n',G),
    ignore(forall(G,true)),
    format('\n% runtime = ~w sec(s)',[T2]).
```

`Demos/1` needs a helper predicate. `FreshFile/1` makes sure that no one else has scribbled, or is currently scribbling, on our output file.

```
freshFile(X) :-
    (current_stream(X,_,S) -> close(S)       ; true),
    (exists_file(X)        -> delete_file(X) ; true).
```

4.1.4 Using `Demos/1`. Next, we need to run the demo code as follows:

```
?- demos(egXXX).
```

Once that is done, then when this document will include the output in the figure with the label `egXXX.spy`.

4.2 List stuff

4.2.1 **writes(+List)***: print a list*

```
writes([H|T]) :-
    forall(member(One,[H|T]),(print(One),nl)).
```

Demonstration code:

```
egwrites :-  % see Figure 1
    writes([aa,bb,cc,dd]).
```

4.2.2 **deletes(+List1,+List2,-List3)***: delete items from a list*

```
deletes([],_,[]).
deletes([Doomed|T],Doomeds,Rest) :-
    member(Doomed,Doomeds),!,
    deletes(T,Doomeds,Rest).
deletes([Saved|T],Doomeds,[Saved|Rest]) :-
    deletes(T,Doomeds,Rest).
```

egwrites.spy

```
% output from ':- demos(egwrites).'

aa
bb
cc
dd

% runtime = 0 sec(s)
```

Fig. 1 From §4.2.1.

egdeletes.spy

```
% output from ':- demos(egdeletes).'

If we take [b, c] from
[a, b, r, a, c, a, d, a, b, r, a] we get
[a, r, a, a, d, a, r, a].

% runtime = 0 sec(s)
```

Fig. 2 From §4.2.2.

Demonstration code:

```
egdeletes :- % see Figure 2
    List = [a,b,r,a,c,a,d,a,b,r,a],
    Doomed=[b,c],
    deletes(List,Doomed,Out),
     format('If we take ~w from\n~w we get\n~w.\n',
           [Doomed,List,Out]).
```

4.3 *Maths stuff*

4.3.1 `sum(+List,-Num)`: *sum a list of numbers*

```
sum([H|T],S) :-
    sum(T,H,S).

sum([],S,S).
sum([H|T],In,Out) :- Temp is H + In, sum(T,Temp,Out).
```

4.3.2 `average(+List,-Num)`: *average a list of numbers*

```
average(L,Av)  :- average(L,_,Av).
average([H|T],N,Av) :- average1(T,1,N,H,Sum), Av is Sum/N.

average1([],N,N,Out,Out).
average1([H|T],N0,N,In,Out) :-
    Temp is H+In,
    N1 is N0 + 1,
    average1(T,N1,N,Temp,Out).
```

Demonstration code:

```
egmaths :- % See Figure ??
    Nums = [2,3,5,2,4,6,3,4,2,4],
    average(Nums,Av),
    sum(Nums,Sum),
    format('The sum and average of\n~w\n are ~w and ~w\n.',
        [Nums,Sum,Av]).
```

4.4 *Lookup tables*

Convert a list of tabular data to one fact for each cell.

```
lookUpTable(X,Out) :-
    bagof(Y,X^list2Relation1(X,Y),Out).

list2Relation1(Table=Cols+Rows, lookUp1(Table,R,C,X)):-
    nth1(Pos,Cols,C),
    member([R|Cells],Rows),
    nth1(Pos,Cells,X),
    nonvar(X).
```

Access the cells

```
lookUp(T,X,Y,Out) :-
    lookUp1(T,R,C,Out), gt(X,R), gt(Y,C).
```

Cell access can be via an exact match or via a range query:

```
gt(Value,X to Y) :- !,X =< Value, Value =< Y.
gt(Value,Value).
```

Demonstration code:

```
egLookUpDemo =
    % age        weight
    % ----------  ------------------------------
                 [1 to 19, 20 to 50, 51 to inf]+
    [[ 0 to 20,        low,      low,       avg]
    ,[21 to 40,        low,      avg,      high]
    ,[41 to inf,       avg,     high,      high]
    ].

eglookup :- % see Figure 3
    Age=30,
    Weight=40,
    lookUp(egLookUpDemo,Age,Weight,X),
    format('[age(~w),weight(~w)]= ~w\n',
           [Age,Weight,X]).
```

eglookup.spy

```
% output from ':- demos(eglookup).'

[age(30),weight(40)]= avg

% runtime = 0 sec(s)
```

Fig. 3 From §4.4.

egrands.spy

```
% output from ':- demos(egrands).'

0.279609 is a random number between 0 and 1.
18.9953 is a random number between 10 and 20.

% runtime = 0 sec(s)
```

Fig. 4 From §4.5.1.

4.5 *Random numbers*

4.5.1 *Basic randoms*

Generate a number $0 \leq X \leq 1$.

```
rand(X) :-
    X is random(inf+1)/inf.
```

Generate a number X between some Min and Max value.

```
rand(Min,Max,X) :-
    X is Min + (Max-Min)*rand.
```

Demonstration code:

```
egrands :-  % see Figure 4
    Rand1 is rand,
    format('~w is a random number between 0 and 1.\n',
            [Rand1]),
    Rand2 is rand(10,20),
    format('~w is a random number between 10 and 20.\n',
           [Rand2]).
```

4.5.2 *Beta distributions*

Generate a number X whose mean is B% between Min and Max. Technically, this is an application of a $beta$ function. Here, I use a very simplistic method that only works for certain values of B: ($B = 0.1, 0.2, 0.3, ..., 0.9, 1$).

egbeta.spy

```
% output from ':- demos(egbeta).'

[11.2313, 14.5453, 11.2714, 13.4645, 10.2245]
 are random numbers 20% between 10 and 20.

% runtime = 0 sec(s)
```

Fig. 5 From §4.5.2.

egnormal.spy

```
% output from ':- demos(egnormal).'

[10.479, 11.2775, 7.82854, 9.13898, 5.80684]
 are random numbers from normal(10,2).
% runtime = 0 sec(s)
```

Fig. 6 From §4.5.3.

```
rand(Min,Max,B,X) :-
    X is Min + (Max-Min)*beta(B).

beta(B,X) :- beta1(B,X),!.
beta(B,X) :- B1 is 1 - B, beta1(B1,Y),X is 1 - Y.

beta1(0.50,X) :- X is rand.
beta1(0.60,X) :- X is rand^0.67.
beta1(0.67,X) :- X is rand^0.5.
beta1(0.75,X) :- X is rand^0.33.
beta1(0.80,X) :- X is rand^0.25.
beta1(0.9,X)  :- X is rand^(1/9).
beta1(1,1).
```

Demonstration code:

```
egbeta :- % see Figure 5
    R1 is rand(10,20,0.2),
    R2 is rand(10,20,0.2),
    R3 is rand(10,20,0.2),
    R4 is rand(10,20,0.2),
    R5 is rand(10,20,0.2),
    Nums=[R1,R2,R3,R4,R5],
    format('~w\n are random numbers 20% between 10 and 20.\n',
           [Nums]).
```

Note that the numbers in Figure 5 may not look like they are, on average, 20% between 10 and 20. Later, we run this code 10,000 times and the true average results can be seen.

4.5.3 Normal distributions Generate a random number from a normal distribution with mean M and standard deviation S. This number is generated using the Box-Muller method (no, I don't understand it either).

```
normal(M,S,N) :-
    box_muller(M,S,N).

box_muller(M,S,N) :-
    wloop(W0,X),
    W is sqrt((-2.0 * log(W0))/W0),
    Y1 is X * W,
    N is M + Y1*S.

wloop(W,X) :-
    X1 is 2.0 * rand - 1,
    X2 is 2.0 * rand - 1,
    W0 is X1*X1 + X2*X2,
    (W0  >= 1.0 -> wloop(W,X) ; W0=W, X = X1).
```

Demonstration code:

```
egnormal :- % see Figure 6
    R1 is normal(10,2),
    R2 is normal(10,2),
    R3 is normal(10,2),
    R4 is normal(10,2),
    R5 is normal(10,2),
    Nums=[R1,R2,R3,R4,R5],
    format('~w\n are random numbers from normal(10,2).',
           [Nums]).
```

eggamma.spy

```
% output from ':- demos(eggamma).'

[19.7148, 7.15347, 4.25717, 8.11787, 16.355]
 are random numbers from gamma(10,2).
% runtime = 0 sec(s)
```

Fig. 7 From §4.5.4.

4.5.4 Gamma distributions Generate random numbers from zero to infinity.

```
gamma(Mean,Alpha,Out) :-
    Beta is Mean/Alpha,
    (Alpha > 20
    ->  Mean is Alpha * Beta,
        Sd is sqrt(Alpha*Beta*Beta),
        Out is normal(Mean,Sd)
    ;   gamma(Alpha,Beta,0,Out)).

gamma(0,_,X,X) :- !.
gamma(Alpha,Beta, In, Gamma) :-
    Temp is In + ( -1 * Beta * log(1-rand)),
    Alpha1 is Alpha - 1,
    gamma(Alpha1,Beta,Temp,Gamma).
```

Technically, this is *gamma* distribution. A standard random *gamma* distribution has the mean $= \frac{alpha}{beta}$. The $alpha$ value is the "spread" of the distribution and controls the clustering of the distribution around the mean. As $alpha$ increases, the $gamma$ distribution flattens out to become more evenly-distributed about the mean. That is, for large $alpha$ (i.e. $alpha \geq 20$), $gamma$ can be modeled as a noraml function. The standard $alpha, beta$ terminology can be confusing to some audiences. Hence, I define a (slightly) more-intuitive $gamma$ distribution where:

$$myGamma(mean, alpha) = gamma\left(alpha, \frac{alpha}{mean}\right)$$

Demonstration code:

```
eggamma :- % see Figure 7
    R1 is gamma(10,2),
    R2 is gamma(10,2),
    R3 is gamma(10,2),
    R4 is gamma(10,2),
    R5 is gamma(10,2),
    Nums=[R1,R2,R3,R4,R5],
    format('~w\n are random numbers from gamma(10,2).',
           [Nums]).
```

4.6 String Stuff

4.6.1 Right-justify a string. Right-justifies a string **`A`** in a space **`S`**:

```
right_justify(S,A) :-
    writeThing(A,Thing,N),
    Pad is S - N,
    forall(between(1,Pad,_),put(32)),
    write(Thing).

writeThing(X,S,L) :-
    sformat(S,'~w',[X]),
    string_length(S,L).
```

Map **`right_justify`** into the **`format`** predicate.

```
padChars(default,A) :- right_justify(5,A).
padChars(S,      A) :- right_justify(S,A).
```

egformat.spy

```
% output from ':- demos(egformat).'

[  tim]
[          tim]
[tim  ]
[tim          ]
[~~~~]
[~~]

% runtime = 0 sec(s)
```

Fig. 8 From §4.6.

4.6.2 *Left-justify a string*

```
left_justify(S,A) :-
    writeThing(A,Thing,N),
    atom_length(A,N),
    Pad is S - N,
    write(Thing),
    forall(between(1,Pad,_),put(32)).

charsPad(default,A) :- left_justify(5,A).
charsPad(S,A)       :- left_justify(S,A).
```

4.6.3 *Print some squiggles*

Generates **N** squiggles in a space normalized to a screen with maximum width **W**.

```
squiggles(W,N) :-
    N1 is round(N/W),
    forall(between(1,N1,_),put(126)).

squiggle(default,A) :- squiggles(25,A).
squiggle(W,N)       :- squiggles(W,N).
```

Demonstration code.

```
egformat :- % Figure 8
    format('[~>]\n',tim),   % right-justify
    format('[~12>]\n',tim), %
    format('[~<]\n',tim),   % left-justify
    format('[~12<]\n',tim), %
    format('[~S]\n',100),    % print some twiddles
    format('[~50S]\n',100),  %
```

4.7 *Predicates for Pairs*

4.7.1 `pairs(?Keys,?Values,?Pairs)`*: key-value pairs*

```
pairs([],[],[]).
pairs([X|Xs],[Y|Ys],[X=Y|T]) :- pairs(Xs,Ys,T).
```

4.7.2 `key(+Pairs,?Key,?Value,?Pairs)`*: a key-in-front working memory*

Acccess values in a list of `key=value` pairs. As a side-effect of accessing, move the accessed pair to the front of the list.

```
key(L0,K,V0,V,[K=V|L]) :-
    less1(L0,K=V0,L).

less1([H|T],H,T).
less1([H|T],Out,[H|Rest]) :-
    less1(T,Out,Rest).
```

4.7.3 `inc(+Pairs,+Key,?Pairs)`*: a lists of counters*

Maintain a list of keys. Incrementing a key add one to its value.

```
inc([], A, [A=1]).
inc([A=B|C],D,E) :-
    compare(F,A,D),inc(F,A=B,C,D,E).

inc(<, A, B, C, [A|D]) :- inc(B, C, D).
inc(=, A=B, C, A, [A=D|C]) :- D is B+1.
inc(>, A, B, C, [C=1, A|B]).
```

eginc.spy

```
% output from ':- demos(eginc).'

The keys in
[a, b, r, a, c, a, d, a, b, r, a]
 occur  with frequencies
[a=5, b=2, c=1, d=1, r=2].
% runtime = 0 sec(s)
```

Fig. 9 From §4.7.3.

egdist.spy

```
% output from ':- demos(egdist).'

The distribution of symbols
[a, b, r, a, c, a, d, a, b, r, a] is
[r=2, d=1, c=1, b=2, a=5].

% runtime = 0 sec(s)
```

Fig. 10 From §4.7.4.

Demonstration code:

```
eginc :- % see Figure 9
    List = [a,b,r,a,c,a,d,a,b,r,a],
    eginc1(List,[],Incs),
    format('The keys in\n~w\n occur  with frequencies\n~w. ',
          [List,Incs]).

eginc1([],W,W).
eginc1([H|T],W0,W) :- inc(W0,H,W1), eginc1(T,W1,W).
```

4.7.4 `dist(+List,-Pairs)`*: Simple collection of histogram data*

```
307 dist(L0,L) :-
308     dist(L0,_,_,L,_).
309
310 dist(L0,L,Most) :-
311     dist(L0,_,_,L,Most).
312
313 dist(L0,Min,Max,L) :-
314     dist(L0,Min,Max,L,_).
315
316 dist(L0,Min,Max,L,Most) :-
317     msort(L0,[Min|L1]), % ◄................................ 317
318     dist([Min|L1],[],Min,Max,L,0,Most).
319
320 dist([],X,Max,Max,X,Most,Most).
321 dist([H|T],[H=N0|Rest],_,Max,Out,Most0,Most) :- !,
322     N is N0 + 1,
323     Most1 is max(Most0,N),
324     dist(T,[H=N|Rest],H,Max,Out,Most1,Most).
325 dist([H|T],In,Min,Max,Out,Most0,Most) :-
326     Most1 is max(Most0,1),
327     dist(T,[H=1|In],Min,Max,Out,Most1,Most).
```

Demonstration code:

```
egdist :- % see Figure 10
    List = [a,b,r,a,c,a,d,a,b,r,a],
    dist(List,Dist),
    format('The distribution of symbols\n~w is\n~w.\n',
          [List,Dist]).
```

Note that `dist/2` *could be* implemented using `inc/3`. However, the call of `msort` at line 317 makes `dist/2` faster for large lists.

4.7.5 `bars(+Num1,+Num2,+Num3,+Pairs)`*: print a bar chart*

Display the pairs as a bar chart. **Num1** is the width of the first "item" column displaying the name of each bar; **Num2** is the width of the second "frequency" column showing how many items fall into that bar; **Num3** is the width of the last column showing the population size.

```
egbarsNormal.spy
% output from ':- demos(egbarsNormal).'



---| 10000 * normal(20, 2) |-------
 item  frequency
   29      1
   27      7
   26     32
   25     92 ~
   24    249 ~~
   23    668 ~~~~~~~
   22   1195 ~~~~~~~~~~~~
   21   1760 ~~~~~~~~~~~~~~~~~~
   20   1963 ~~~~~~~~~~~~~~~~~~~~
   19   1767 ~~~~~~~~~~~~~~~~~~
   18   1271 ~~~~~~~~~~~~~
   17    603 ~~~~~~
   16    288 ~~~
   15     76 ~
   14     26
   13      2

% runtime = 0.540778 sec(s)
```

Fig. 11 From §4.7.5.

```
egbarsBeta.spy
% output from ':- demos(egbarsBeta).'



---| 10000 * rand(10, 20, 0.2) |-------
 item  frequency
   19      4
   18     30
   17    109 ~
   16    244 ~~
   15    499 ~~~~~
   14    860 ~~~~~~~~~
   13   1401 ~~~~~~~~~~~~~~
   12   2059 ~~~~~~~~~~~~~~~~~~~~~
   11   2961 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   10   1833 ~~~~~~~~~~~~~~~~~~

% runtime = 0.310446 sec(s)
```

Fig. 12 From §4.7.5.

```
bars(Num1,Num2,Num3,List) :-
```

Use **sformat** to builds a string that stores the widths and scale factor for our columns. Note the use of "¿" and "S" which are special format commands defined above.

```
    sformat(S,'~~~w>  ~~~w> ~~~wS\n',
        [Num1,Num2,Num3]),
    dist(List,Dist),
    nl,
    format(S,[item,frequency,0]),
    forall(member(What=N,Dist),
           format(S,[What,N,N])).
```

A useful default call.

```
bars(List) :-
    bars(5,     % the "item" column is 5 wide
         5,     % the "frequency" column is 5 wide
         3,     % the "scale factor" is 3
         List  % now, go display these pairs
         ).
```

Demonstration code:

```
egbarsGamma1.spy
% output from ':- demos(egbarsGamma1).'



---| 10000 * gamma(10, 15) |-------
 item  frequency
   22      3
   21      4
   20     10
   19     11
   18     34
   17     76 ~
   16    144 ~
   15    241 ~~
   14    451 ~~~~~
   13    665 ~~~~~~~
   12    942 ~~~~~~~~~
   11   1321 ~~~~~~~~~~~~~
   10   1548 ~~~~~~~~~~~~~~~
    9   1593 ~~~~~~~~~~~~~~~~
    8   1378 ~~~~~~~~~~~~~~
    7    901 ~~~~~~~~~
    6    466 ~~~~~
    5    169 ~~
    4     38
    3      5

% runtime = 1.94279 sec(s)
```

Fig. 13 From §4.7.5.

```
egbarsGamma2.spy
% output from ':- demos(egbarsGamma2).'



---| 10000 * gamma(10, 5) |-------
 item  frequency
   36      1
   34      1
   33      1
   32      3
   31      2
   30      5
   29      2
   28     10
   27     17
   26      7
   25     12
   24     26
   23     31
   22     52 ~
   21     61 ~
   20     86 ~
   19    139 ~
   18    184 ~~
   17    236 ~~
   16    293 ~~~
   15    348 ~~~
   14    433 ~~~~
   13    578 ~~~~~~
   12    662 ~~~~~~~
   11    783 ~~~~~~~~
   10    900 ~~~~~~~~~
    9    949 ~~~~~~~~~
    8    947 ~~~~~~~~~
    7    906 ~~~~~~~~~
    6    824 ~~~~~~~~
    5    709 ~~~~~~~
    4    463 ~~~~~
    3    244 ~~
    2     77 ~
    1      8

% runtime = 0.761094 sec(s)
```

Fig. 14 From §4.7.5.

egbarsGamma3.spy

```
% output from ':- demos(egbarsGamma3).'



---| 10000 * gamma(10, 2) |-------
 item  frequency
   64      1
   55      1
   53      1
   52      3
   50      1
   48      1
   46      2
   45      1
   44      1
   43      6
   42      6
   41      2
   40      9
   39      4
   38     12
   37      8
   36     10
   35     14
   34     14
   33     22
   32     27
   31     30
   30     32
   29     27
   28     43
   27     52 ~
   26     53 ~
   25     64 ~
   24     72 ~
   23     95 ~
   22     99 ~
   21    121 ~
   20    158 ~~
   19    167 ~~
   18    182 ~~
   17    227 ~~
   16    267 ~~~
   15    290 ~~~
   14    367 ~~~~
   13    389 ~~~~
   12    448 ~~~~
   11    485 ~~~~~
   10    534 ~~~~~
    9    601 ~~~~~~
    8    634 ~~~~~~
    7    699 ~~~~~~~
    6    743 ~~~~~~~
    5    719 ~~~~~~~
    4    757 ~~~~~~~~
    3    632 ~~~~~~
    2    534 ~~~~~
    1    295 ~~~
    0     38

% runtime = 0.41059 sec(s)
```

Fig. 15 From §4.7.5.

```
egbarsNormal :- % see Figure 11
    egbarDemos(10000,normal(20,2)).

egbarsBeta :- % see Figure 12
    egbarDemos(10000,rand(10,20,0.2)).

egbarsGamma1 :- % see Figure 13
    egbarDemos(10000,gamma(10,15)).

egbarsGamma2 :- % see Figure 14
    egbarDemos(10000,gamma(10,5)).

egbarsGamma3 :- % see Figure 15
    egbarDemos(10000,gamma(10,2)).
```

Support code for the demostration code:

egnormalize.spy

```
% output from ':- demos(egnormalize).'

When [a=10, b=5, c=20, d=50, e=5, c=10]
 is normalized it generates
[a=0.1, b=0.05, c=0.2, d=0.5, e=0.05, c=0.1].

% runtime = 0 sec(s)
```

Fig. 16 From §4.7.6.

```
egbarDemos(Repeats,F) :-
    format('\n\n---| ~w * ~w |-------',[Repeats,F]),
    Size=1,
    findall(X,(between(1,Repeats,_),X is F),L0),
    cutDown2Sizes(Size,L0,L),
    bars(5,5,100,L).

cutDown2Sizes(Size) --> maplist(cutDown2Size(Size)).
cutDown2Size(Size,X,Y) :- Y is round(X/Size).
```

4.7.6 **`normalize(+Pairs1,-Pairs2)`***: normalize a list of numbers* Input list with values $M_1, M_2 \ldots M_i$ with sum $M_1{+}M_2{+}\ldots{+}M_i$ to a second list of numbers $N_1, N_2, \ldots, N_i$ where $0 \leq N_i \leq 1$ and $N_1 + N_2 + \ldots + N_i = 1$.

```
normalize(L,N) :-
    mostnormal(L,N,_).

mostnormal(L,N,Most) :-
    sumpairs(L,Sum),
    mostnormal1(L,Sum,junk= -1,N,Most).

mostnormal1([],_,Out,[],Out).
mostnormal1([X=V0|T],Sum,Y=N,[X=N1|Out],Most) :-
    N1 is V0/Sum,
    (N1 > N
    -> mostnormal1(T,Sum,X=N1,Out,Most)
        ;  mostnormal1(T,Sum,Y=N,Out,Most)).

sumpairs([_H=V|T],S) :-
    sumpairs(T,V,S).

sumpairs([],S,S).
sumpairs([_=V|T],In,Out) :-
    Temp is V + In, sumpairs(T,Temp,Out).
```

Demonstration code:

```
egnormalize :- % see Figure 16
    L=[a=10,b=5,c=20,d=50,e=5,c=10],
    normalize(L,Normals),
    format('When ~w\n is normalized it generates\n~w.\n',
                [L,Normals]).
```

4.7.7 Ordered Sets Standard definition:

```
oset_add([], El, [El]).
oset_add([H|T], El, Add) :-
    compare(Order, H, El),
    addel(Order, H, T, El, Add).

addel(<, H, T,  El, [H|Add]) :- oset_add(T, El, Add).
addel(=, H, T, _El, [H|T]).
addel(>, H, T,  El, [El,H|T]).
```

With key-value pairs Bulk additions

```
koset_adds(L,Out) :- koset_adds(L,[],Out).

koset_adds([],Out,Out).
koset_adds([H|T],In,Out) :- koset_add(In,H,Temp), koset_adds(T,Temp,O

koset_add([], El, [El]).
koset_add([H=X|T], El=Y, Add) :-
    compare(Order, H, El),
    kaddel(Order, H, X,T, El, Y,Add).

kaddel(<, H, X,T,  El, Y,[H=X     |Add]) :- koset_add(T, El=Y, Add).
kaddel(=, H, _,T, _El, Y,[H=Y     |T]).
kaddel(>, H, X,T,  El, Y,[El=Y,H=X|T]).
```

nowarranty.txt

```
comes with ABSOLUTELY NO WARRANTY:
for more details type 'warranty'.

This is free software, and you are welcome to
redistribute it under certain conditions: for
more details, type 'conditions'.
```

Fig. 17 A text file.

egchars.spy

```
% output from ':- demos(egchars).'

comes with ABSOLUTELY NO WARRANTY:
for more details type 'warranty'.

This is free software, and you are welcome to
redistribute it under certain conditions: for
more details, type 'conditions'.
% runtime = 0 sec(s)
```

Fig. 18 The code in §4.8.4 displays the contents of Figure 17 to the screen.

4.8 *Input/output stuff*

Demonstrations are offered for only some of the predicates in this section. I/O code can makes explicit calls to input/output streams which mucks up our demonstration system.

4.8.1 `sneak(+List)`*: load files.*

Don't bother loading the files if they haven't changed. But if you do load them, don't print anything to the screen.

```
sneak(X) :-
    load_files(X,[silent(true),if(changed)]).
```

4.8.2 `spit(+Num1,+Num2,+Term)`*: Print something, sometimes.*

Useful for tracking a long process since it, sometimes, spits out a marker.

```
spit(N1,N2,X) :-
    (0 is N1 mod N2 -> blurt(X) ; true).
```

4.8.3 `blurt(+Term)`*: print, then flush.*

```
blurt(X) :-
    write(user,X),flush_output(user).
```

4.8.4 `chars(+String)`*: copy a file to the screen.*

```
chars(File) :-
    see(File), get_byte(X), ignore(chars1(X)), seen.

chars1(-1) :- !.
chars1(X)  :- put(X), get_byte(Y), chars1(Y).
```

Demonstration code:

```
egchars :-  % see Figure 18.
    chars('nowarranty.txt').
```

4.8.5 `barph(+Term)`*: print a warning, then fail.*

A standard barph:

```
barph(X) :- format('%W> ~p\n',X),fail.
```

A barph that also prints line numbers showing the origin of the barph.

```
barphln(X) :-
    here(File,Line),
    format('%W> ~p@~p : ~p\n',[File,Line,X]),
        fail.

here(File,Line) :-
        source_location(Path,Line),
        file_base_name(Path,File).
```

egtidy.spy

```
% output from ':- demos(egtidy).'

a :-
    b,
    c,
    (   d
    ->  true
    ;   e
    ).
f.

% runtime = 0 sec(s)
```

Fig. 19 From §4.9.1.

egdemand.spy

```
% output from ':- demos(egdemand).'

%W> failed(10>20)

% runtime = 0 sec(s)
```

Fig. 20 From §4.9.3.

4.9 *Meta-level predicates*

4.9.1 `tidy(+Rule0,-Rule1)`*: remove stray "trues" from a rule body.*

```
tidy(A,C) :-
    tidy1(A,B),
    (B = (Head :- true) -> C=Head ;  C=B).

tidy1(A,C) :- once(tidy2(A,C)).

tidy2(A,             A) :- var(A).
tidy2((A,B),    (A,TB)) :- var(A), tidy1(B,TB).
tidy2((A,B),    (TA,B)) :- var(B), tidy1(A,TA).
tidy2(((A,B),C),     R) :- tidy1((A,B,C), R).
tidy2((true,A),      R) :- tidy1(A,R).
tidy2((A,true),      R) :- tidy1(A,R).
tidy2((A;true),      R) :- tidy1(A,R).
tidy2((true;A),      R) :- tidy1(A,R).
tidy2((A;B),   (TA;TB)) :- tidy1(A,TA), tidy1(B,TB).
tidy2((A->B), (TA->TB)) :- tidy1(A,TA), tidy1(B,TB).
tidy2(not(A),  not(TA)) :- tidy1(A,TA).
tidy2((A :- B), R) :-
    tidy1(B,TB),(TB=true-> R=A; R=(A:-TB)).
tidy2((A,B),  R) :-
    tidy1(A,TA), tidy1(B,TB),(TB=true -> R=TA; R=(TA,TB)).
tidy2(A,A).
```

Demonstration code:

```
egtidy :- % see Figure 19
    In1= (a :- b, true,c, (d->true;e)),
    In2= (f :- true,(true;true;true),true),
    tidy(In1,Out1),
    portray_clause(Out1),
    tidy(In2,Out2),
    portray_clause(Out2).
```

4.9.2 `ensure(+Term)`*: some assertion exists*

```
ensure(X) :- X,!.
ensure(X) :- assert(X).
```

4.9.3 `demand(+Goal)`*: warn if a goal fails.*

```
demand(X) :- X,!.
demand(X) :- numbervars(X,0,_),barph(failed(X)).
```

Demonstration code:

```
egdemand :- % see Figure 20
    demand(3 > 2),
    demand(10 > 20).
```

egtimes.spy

```
% output from ':- demos(egtimes).'

In 10000 repeats, each run took 8.01152e-006 seconds.

% runtime = 0.100144 sec(s)
```

Fig. 21 From §4.9.5.

4.9.4 **`repeats(+Num,+Goal)`***: run a goal N times*

```
repeats(N0,G) :-
    N is N0,
    forall(between(1,N,_),G).
```

4.9.5 **`times(+Num,+Goal,-Time)`***: time an execution*

```
times(N,G,Out) :-
    T1 is cputime, repeats(N,true),
    T2 is cputime, repeats(N,G),
    T3 is cputime, Out is (T3-T2-(T2-T1))/N.
```

Demonstration code:

```
egtimes :- % see Figure 21
    N=10000,
    List = [a,b,r,a,c,a,d,a,b,r,a,s],
    times(N,member(s,List),T),
    format('In ~w repeats, each run took ~w seconds.\n',
           [N,T]).
```

4.9.6 **`!Repeats*!Goal1/!Goal2`***: compare runtimes*

```
N*X/Y :- !,
        times(N,X,T1),
    times(N,Y,T2),
    Inc=0.000001,
        Ratio is (T1+Inc)/(T2+Inc),
    write(goal1=X),nl,
    write(time(goal1)=T1),nl,
    write(goal2=Y),nl,
    write(time(goal2)=T2),nl,
    write(time(goal1)/time(goal2)=Ratio),nl.


N*X :- time(times(N,X,_)).
```

4.9.7 Lists/ conjuctions conversions.

Convert a conjunction to a list:

```
c2l((X,Y),[X|Z]) :- !,c2l(Y,Z).
c2l(X,[X]).
```

Convert everything but the last item of a conjunction to a list:

```
mostC2l((X,Y),[X|Z]) :- !,mostC2l(Y,Z).
mostC2l(_,[]).
```

Convert a list to a conjunction:

```
l2c([W,X|Y],(W,Z)) :- l2c([X|Y],Z).
l2c([X],X).
```

Convert disjunctions to a list.

```
d2l((X;Y),[X|Z]) :- !,d2l(Y,Z).
d2l(X,[X]).
```

4.9.8 **`clause1(?Head,?Body)`***: does a goal match only one clause?*

```
clause1(X,Y) :-
    singleton(X), clause(X,Y).

singleton(X) :-
    Sym='$singleton_',
    flag(Sym,_,0),
    \+ singleton1(Sym,X),
    flag(Sym,1,1).

singleton1(Sym,X) :-
    clause(X,_),flag(Sym,N,N+1),N > 1,!.
```

4.9.9 **`only(?Goal)`***: can a goal only succeed once?*

```
only(X) :-
    Sym='$only_',
    flag(Sym,_,0),
    \+ only1(Sym,X),
    flag(Sym,1,1).

only1(Sym,X) :-
    X, flag(Sym,N,N+1),N > 1,!.

solo(X) :-
    only(X), X.
```

5 Start-up commands

```
:- current_prolog_flag(max_integer,X),
    X1 is X - 1,
    retractall(inf(_)),
    assert(inf(X1)).
```

6 Bugs

None known but many suspected.

Acknowledgements This research was conducted at West Virginia University under NASA contract NCC2-0979. The work was sponsored by the NASA Office of Safety and Mission Assurance under the Software Assurance Research Program led by the NASA IV&V Facility. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government.

References

1. T. Menzies. PROD: A PROlog Documentation, and Delivery tool, 2003. Available from `http://tim.menzies.com/pdf/03prod2.pdf`.
2. T. Menzies. PROD: the soruce code, 2003. Available from `http://tim.menzies.com/prodfiles.zip`.
3. T. Menzies. An example of the PROD prolog delivery and documentation system, 2003. Available from `http://tim.menzies.com/pdf/03prod1.pdf`.
4. T. Menzies. A family database: documentation of a very simple prolog family database using PROD., 2003. Available from `http://tim.menzies.com/pdf/03family.pdf`.
5. T. Menzies. Including gpl-2 in PROD, 2003. Available from `http://tim.menzies.com/pdf/03gpl.pdf`.
6. T. Menzies. Monte carlo simulations in prolog: a PROD tool, 2003. Available from `http://tim.menzies.com/pdf/03lurch.pdf`.
7. T. Menzies. Motivations: the why and who of PROD, 2003. Available from `http://tim.menzies.com/pdf/03prodabout.pdf`.
8. T. Menzies. PROD's commonly used predicates, 2003. Available from `http://tim.menzies.com/pdf/03lib.pdf`.
9. T. Menzies. PROD's handler for config files and command line options, 2003. Available from `http://tim.menzies.com/pdf/03cfg.pdf`.

This paper was prepared using the PROD Prolog documenet and delivery preperation system [1, 2]:

– PROD was created to help me train my graduate students. It also is a nice demonstration of the utility of Prolog. For more on this point, see [7] (warning: includes sermonizing).
– PROD comes with a simple examples of documenting Prolog source code (e.g. a predicate to sum a list of numbers [3] and the standard Prolog family database [4] as well as a “bare-bones” example [11].
– PROD was designed as an open source tool. The GNU public license (version 2) can be shown from any PROD application and is added to every PROD document [5]).
– PROD comes with some handy little utilities and some applications:
  – A handler for config files, user preferences, and command line options [9].
  – A library of commonly used predicates [8].
  – A software cost estimation tool [10].
  – A monte carlo simulation package for Prolog [6].

Fig. 22 This document is part of the PROD delivery and documentation tool for Prolog applications. To find out more about PROD, the best place to start is [1].

10. T. Menzies. Software cost estimation: a PROD tool, 2003. Available from `http://tim.menzies.com/pdf/03omo.pdf`.
11. T. Menzies. Title: a bare-bones minimal example of PROD., 2003. Available from `http://tim.menzies.com/pdf/03prod3.pdf`.

A License

This software is distributed under the GNU General Public License.

A.1 nowarranty.txt

LIB comes with ABSOLUTELY NO WARRANTY: for more details type ’warranty’.

This is free software, and you are welcome to redistribute it under certain conditions: for more details, type ’conditions’.

A.2 warranty.txt

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; version 2 (see http://www.gnu.org/copyleft/gpl.html or type ’conditions’).

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, US.

A.3 conditions.txt

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The ’Program’, below, refers to any such program or work, and a ’work based on the Program’ means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term ’modification’.) Each licensee is addressed as ’you’.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program’s source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source code along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and 'any later version', you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM 'AS IS' WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS