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, January 30, 2003.

Abstract Commonly used Prolog predicates.

Contents

1	Insta	Illation	1
2	Pre-l	load actions	1
	2.1	Operators	1
	2.2	Flags	1
	2.3	Hooks	2
3	Pred	icates	2
	3.1	Code to demonstrate predicates	2
	3.2	List stuff	2
	3.3	Maths stuff	3
	3.4	Lookup tables	3
	3.5	Random numbers	3
	3.6	String Stuff	4
	3.7	Predicates for Pairs	5
	3.8	Input/output stuff	7
	3.9	Meta-level predicates	8
4	Start	-up commands	9
5	Bugs	3	9
А	Licer	1se	9
	A.1	nowarranty.txt	9
	A.2	warranty.txt	9
	A.3	conditions.txt	9

List of Figures

1	egwrites.spy	2
2	egdeletes.spy	3
3	eglookup.spy	3
4	egrands.spy	3
5	egbeta.spy	4
6	egnormal.spy	4
7	eggamma.spy	4
8	egformat.spy	5
9	eginc.spy	5
10	egdist.spy	5
11	egbarsNormal.spy	6
12	egbarsBeta.spy	6
13	egbarsGamma1.spy	6

14	1	~
14	egbarsGamma2.spy	6
15	egbarsGamma3.spy	7
16	egnormalize.spy	7
17	nowarranty.txt	8
18	egchars.spy	8
19	egtidy.spy	8
20	egdemand.spy	8
21	egtimes.spy	8
22	Find out more about PROD	10

1 Installation

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

2 Pre-load actions

2.1 Operators

Define an operator to handle numberic ranges Min to Max. 5 :- op(1,xfx, to).

2.2 Flags

Define an **inf**inity function.

6 :- arithmetic_function(inf/0).

Define a **rand**om function.

- 7 :- arithmetic_function(rand/0).
- Define a rand(Min,Max) function.
- 8 :- arithmetic_function(rand/2).
- Define a rand(Min,Max,Mean) function.
- 9 :- arithmetic_function(rand/3).
- Define a **normal** function.
- 10 :- arithmetic_function(normal/2).

Define a **beta** function.

11 :- arithmetic_function(beta/1).

```
Define a gamma function.
```

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

```
Add a "left-justify" command to format.
```

- 13 :- format_predicate('>',padChars(_,_)).
- Add a "right-justify" command to format.
- 14 :- format_predicate('<',charsPad(_,_)).</pre>

Add a "print squiggles" command to format. 15 :- format_predicate('S',squiggle(_,_)).

- Define a predicate for the lookup tables.
- 16 :- discontiguous lookUp1/4.

```
17 :- multifile lookUp1/4.
```

18 :- index(lookUp1(1,1,1,0)).

2.3 Hooks

A hook for lookup tables.

19	<pre>term_expansion(Table = ColsRows , Out)</pre>	:-
20	nonvar(ColsRows),	
21	ColsRows = (Cols+Rows).	

lookUpTable(Table=Cols+Rows,Out). 22

3 Predicates

3.1 Code to demonstrate predicates

```
3.1.1 Define demos
```

```
23 eglib :-
24
      forall(member(G,[
25
               egwrites, egdeletes,
                                        eqmaths,
                                                  eglookup,
               eginc,
26
               egrands,
                          egbeta,
                                        egnormal, eggamma,
27
                                        egdist,
               egformat, eginc,
28
               egbarsNormal, egbarsBeta,
29
30
               egbarsGamma1, egbarsGamma2
                                             egbarsGamma3,
31
               egnormalize, egchars,
                                        egtidy,
               egdemand, egtimes
32
33
               1),
           demos(G)).
34
```

- 3.1.2 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)53 generates a file eqXXX.spy.
- Include that file. This is accomplished using the following LATEX magic:

\SRC{egXXX.spy}{From \tion{egXXX/1}.} Note the call to $tion{eqXXX/1}$. Sections can be referenced symbolically when (e.g.) \label{sec:egXXX/1) is added on the first line after a heading definition. Once this has been done, then $\tion{eqXXX/1}$ 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.

```
egwrites.spy
% output from ':- demos(egwrites).
aa
bb
CC
44
% runtime = 0 sec(s)
```

Fig. 1 From §3.2.1.

3.1.3 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.

```
35 demos(G) :-
```

- 36 sformat(Out,'~w.spv',G), freshFile(Out),
- 37 38 tell(Out),

```
format('% output from '':- demos(~w).''\n\n',G),
39
```

```
T1 is cputime,
40
```

- ignore(forall(G,true)), 41 42
- T2 is (cputime T1),
- format('\n% runtime = ~w sec(s)\n',[T2]), 43 44

```
told,
nl,write('\n%-----
```

- 45 ----\n'), format('% output from '':- demos(~w).''\n',G), 46
- 47
- ignore(forall(G,true)),
 format('\n% runtime = ~w sec(s)',[T2]). 48

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

```
49 freshFile(X) :-
```

```
(current_stream(X,_,S) -> close(S)
50
                                                  ; true),
51
      (exists file(X)
                               -> delete file(X) ; true).
```

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

```
?- demos(eqXXX).
```

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

3.2 List stuff

```
3.2.1 writes(+List): print a list
```

52 writes([H|T]) :-

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

Demonstration code:

```
62 egwrites :- % see Figure 1
```

```
writes([aa,bb,cc,dd]).
63
```

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

```
64 deletes([],_,[]).
65 deletes([Doomed ]], Doomeds, Rest) :-
      member(Doomed, Doomeds), !,
66
      deletes(T,Doomeds,Rest).
67
68 deletes([Saved | T], Doomeds, [Saved | Rest]) :-
      deletes(T, Doomeds, Rest).
69
```

Demonstration code:

```
77 egdeletes :- % see Figure 2
      List = [a,b,r,a,c,a,d,a,b,r,a],
78
79
      Doomed=[b,c],
80
      deletes(List, Doomed, Out),
```

```
81
        format('If we take \mbox{``w from \n``w we get \n``w.\n',}
82
               [Doomed,List,Out]).
```

```
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 §3.2.2.

eglookup.spy
% output from ':- demos(eglookup).'
[age(30),weight(40)]= avg
% runtime = 0 sec(s)

Fig. 3 From §3.4.

3.3 Maths stuff

3.3.1 sum(+List,-Num): sum a list of numbers

```
83 sum([H|T],S) :-
84 sum(T,H,S).
85
86 sum([],S,S).
87 sum([H|T],In,Out) :- Temp is H + In, sum(T,Temp,Out).
```

3.3.2 average(+List,-Num): average a list of numbers

```
88 average(L,Av) :- average(L,_,Av).
89 average([H|T],N,Av) :- averagel(T,1,N,H,Sum), Av is Sum/N.
90
91 averagel([],N,N,Out,Out).
92 averagel([H|T],NO,N,In,Out) :-
93 Temp is H+In,
94 N1 is N0 + 1,
95 averagel(T,N1,N,Temp,Out).
```

Demonstration code:

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

3.4 Lookup tables

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

```
102 lookUpTable(X,Out) :-
103
       bagof(Y,X<sup>1</sup>ist2Relation1(X,Y),Out).
104
105 list2Relation1(Table=Cols+Rows, lookUp1(Table,R,C,X)):-
       nth1(Pos,Cols,C),
106
107
       member([R Cells],Rows),
       nth1(Pos,Cells,X),
108
109
       nonvar(X).
  Access the cells
110 lookUp(T,X,Y,Out) :-
       lookUp1(T,R,C,Out), gt(X,R), gt(Y,C), !.
111
```

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

```
112 gt(Value,X to Y) :- !,X =< Value, Value =< Y.
113 gt(Value,Value).</pre>
```

Demonstration code:

```
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 §3.5.1.

```
119 eqLookUpDemo =
120
      % age
                    weight
121
      % -----
                    122
                    [1 to 19, 20 to 50, 51 to inf]+
      [[ 0 to 20,
123
                         low,
                                   low,
                                              avg]
      ,[21 to 40,
                         low,
124
                                   avg,
                                             high]
                         avg,
125
       ,[41 to inf,
                                  high,
                                             high]
126
      1.
127
128 eglookup :- % see Figure 3
129
      Age=30,
130
      Weight=40,
131
      lookUp(egLookUpDemo,Age,Weight,X),
      format('[age(~w),weight(~w)]= ~w \n',
132
             [Age,Weight,X]).
133
```

3.5 Random numbers

```
3.5.1 Basic randoms Generate a number 0 \le X \le 1.
```

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

Generate a number X between some Min and Max value.

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

Demonstration code:

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

3.5.2 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).

```
151 rand(Min,Max,B,X) :-
152
      X is Min + (Max-Min)*beta(B).
153
154 beta(B,X) :- beta1(B,X),!.
155 beta(B,X) :- B1 is 1 - B, beta1(B1,Y),X is 1 - Y.
156
157 betal(0.50,X) :- X is rand.
158 betal(0.60,X) :- X is rand^0.67.
159 betal(0.67,X) :- X is rand^0.5.
160 beta1(0.75,X) :- X is rand^0.33.
161 beta1(0.80,X) :- X is rand^0.25.
162 betal(0.9,X) :- X is rand^(1/9).
163 beta1(1,1).
  Demonstration code:
170 egbeta :- % see Figure 5
171
      R1 is rand(10,20,0.2),
172
```

```
172 R2 is rand(10,20,0.2),
173 R3 is rand(10,20,0.2),
174 R4 is rand(10,20,0.2),
175 R5 is rand(10,20,0.2),
176 Nums=[R1,R2,R3,R4,R5],
177 format('~w\n are random numbers 20% between 10 and 20.\n',
178 [Nums]).
```



Fig. 5 From §3.5.2.

Fig. 6 From §3.5.3.

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.

3.5.3 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).

```
179 normal(M,S,N) :-
180
      box_muller(M,S,N).
181
182 box_muller(M,S,N) :-
183
       wloop(W0,X),
       W is sqrt((-2.0 * log(W0))/W0),
184
       Y1 is X * W,
185
      N is M + Y1*S.
186
187
188 wloop(W,X) :-
       X1 is 2.0 * rand - 1,
189
       X2 is 2.0 * rand - 1,
190
191
       W0 is X1*X1 + X2*X2,
192
       (W0 >= 1.0 -> wloop(W,X) ; W0=W, X = X1).
  Demonstration code:
198 egnormal :- % see Figure 6
```

```
R1 is normal(10,2),
199
       R2 is normal(10,2),
200
201
      R3 is normal(10,2),
       R4 is normal(10,2)
202
       R5 is normal(10,2)
203
       Nums=[R1,R2,R3,R4,R5],
204
205
       format('w \leq n are random numbers from normal(10,2).',
206
               [Nums]).
```

3.5.4 Gamma distributions Generate random numbers from zero to infinity.

```
207 gamma(Mean, Alpha, Out) :-
       Beta is Mean/Alpha,
208
       (Alpha > 20
209
           Mean is Alpha * Beta.
210
       ->
           Sd is sgrt(Alpha*Beta*Beta),
211
           Out is normal(Mean.Sd)
212
213
           gamma(Alpha,Beta,0,Out)).
214
215 gamma(0,_,X,X) :- !.
216 gamma(Alpha,Beta, In, Gamma) :-
       Temp is In + ( -1 * Beta * log(1-rand)),
217
218
       Alpha1 is Alpha - 1,
219
       gamma(Alpha1, Beta, Temp, Gamma).
```

Technically, this is *gamma* distribution. A standard random *gamma* distribution has the mean $= \frac{alpha}{beta}$. The *alpha* value

```
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 §3.5.4.

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 evenlydistributed 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:

```
225 eggamma :- % see Figure 7
       R1 is gamma(10,2),
226
227
       R2 is gamma(10,2),
228
       R3 is gamma(10,2),
229
       R4 is gamma(10,2),
230
       R5 is gamma(10,2),
       Nums=[R1,R2,R3,R4,R5],
231
232
       format('~w\n are random numbers from gamma(10,2).',
              [Nums]).
233
```

```
3.6 String Stuff
```

3.6.1 *Right-justify a string*. Right-justifies a string **A** in a space **s**:

```
234 right_justify(S,A) :-
235 writeThing(A,Thing,N),
236 Pad is S - N,
237 forall(between(1,Pad,_),put(32)),
238 write(Thing).
239
240 writeThing(X,S,L) :-
241 sformat(S,'~w',[X]),
242 string_length(S,L).
```

Map right_justify into the format predicate.

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

3.6.2 *Left-justify a string*

```
245 left_justify(S,A) :-
246 writeThing(A,Thing,N),
247 atom_length(A,N),
248 Pad is S - N,
249 write(Thing),
250 forall(between(1,Pad,_),put(32)).
251
252 charsPad(default,A) :- left_justify(5,A).
253 charsPad(S,A) :- left_justify(S,A).
```

3.6.3 *Print some squiggles* Generates \mathbf{N} squiggles in a space normalized to a screen with maximum width \mathbf{W} .

```
254 squiggles(W,N) :-
255 N1 is round(N/W),
256 forall(between(1,N1,_),put(126)).
257
258 squiggle(default,A) :- squiggles(25,A).
259 squiggle(W,N) :- squiggles(W,N).
```

```
egformat.spy
% output from ':- demos(egformat).'
[ tim]
[ tim]
[tim ]
[tim ]
[~~~]
% runtime = 0 sec(s)
```

Fig. 8 From §3.6.

```
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 §3.7.3.

Demonstration code.

```
270 egformat :- % Figure 8
       format('[~>]\n',tim),
                                  % right-justify
271
       format('[~12>]\n',tim), %
272
       format('[~<]\n',tim),</pre>
273
                                  % left-justify
       format('[~12<]\n',tim), %</pre>
274
       format('[~s]\n',100),
                                   % print some twiddles
275
       format('[~50S]\n',100), %
276
```

3.7 Predicates for Pairs

3.7.1 pairs(?Keys,?Values,?Pairs): key-value pairs

```
277 pairs([],[],[]).
278 pairs([X|XS],[Y|YS],[X=Y|T]) :- pairs(XS,YS,T).
```

3.7.2 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.

```
279 key(L0,K,V0,V,[K=V|L]) :-
280 lessl(L0,K=V0,L).
281
282 lessl([H|T],H,T).
283 lessl([H|T],Out,[H|Rest]) :-
284 lessl(T,Out,Rest).
```

3.7.3 inc(+Pairs,+Key,?Pairs): a lists of counters Maintain a list of keys. Incrementing a key add one to its value.

```
285 inc([], A, [A=1]).
286 inc([A=B|C],D,E) :-
287 compare(F,A,D),inc(F,A=B,C,D,E).
288
289 inc(<, A, B, C, [A|D]) :- inc(B, C, D).
290 inc(=, A=B, C, A, [A=D|C]) :- D is B+1.
291 inc(>, A, B, C, [C=1, A|B]).
Demonstration code:
```

```
299 eginc :- % see Figure 9
300 List = [a,b,r,a,c,a,d,a,b,r,a],
301 eginc1(List,[],Incs),
302 format('The keys in \n ~w\n occur with frequencies \n ~w.
303 [List,Incs]).
304
305 eginc1([],W,W).
306 eginc1([H|T],W0,W) :- inc(W0,H,W1), eginc1(T,W1,W).
```

```
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 §3.7.4.

3.7.4 dist(+List, -Pairs): Simple collection of histogram data

```
307 dist(L0,L) :-
      dist(L0,_,_,L).
308
309
310 dist(L0,Min,Max,L) :-
311
      msort(L0,[Min|L1]), % <..... 311</pre>
      dist([Min|L1],[],Min,Max,L).
312
313
314 dist([],X,Max,Max,X).
315 dist([H|T],[H=N0|Rest],_,Max,Out) :- !,
      N is N0 + 1,
316
      dist(T,[H=N Rest],H,Max,Out).
317
318 dist([H|T], In, Min, Max, Out) :-
      dist(T,[H=1 | In],Min,Max,Out).
319
```

Demonstration code:

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

Note that dist/2 could be implemented using inc/3. However, the call of msort at line 311 makes dist/2 faster for large lists.

3.7.5 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.

332 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 " $_{i}$ " and "S" which are special format commands defined above.

```
333 sformat(S,'~~w> ~~w> ~~wS\n',
334 [Numl,Num2,Num3]),
335 dist(List,Dist),
336 nl,
337 format(S,[item,frequency,0]),
338 forall(member(What=N,Dist),
339 format(S,[What,N,N])).
```

A useful default call.

```
340 bars(List) :-
341 bars(5, % the "item" column is 5 wide
342 5, % the "frequency" column is 5 wide
,43 3, % the "scale factor" is 3
344 List % now, go display these pairs
345 ).
```

Demonstration code:





egbarsBeta.spy
<pre>% output from ':- demos(egbarsBeta).'</pre>
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 §3.7.5.

```
520 egbarsNormal :- % see Figure 11
521
       egbarDemos(10000,normal(20,2)).
522
523 egbarsBeta :- % see Figure 12
      egbarDemos(10000, rand(10, 20, 0.2)).
524
525
526 egbarsGammal :- % see Figure 13
      egbarDemos(10000,gamma(10,15)).
527
528
529 egbarsGamma2 :- % see Figure 14
       egbarDemos(10000,gamma(10,5)).
530
531
532 egbarsGamma3 :- % see Figure 15
      egbarDemos(10000,gamma(10,2)).
533
  Support code for the demostration code:
534 egbarDemos(Repeats,F) :-
535 format('\n\n-| ~w * ~w |-----',[Repeats,F]),
```

```
535 format('\n\n---| ~w * ~w |------',[Repeats,F]),
536 Size=1,
537 findall(X,(between(1,Repeats,_),X is F),L0),
538 cutDown2Sizes(Size,L0,L),
539 bars(5,5,100,L).
540
541 cutDown2Sizes(Size) --> maplist(cutDown2Size(Size)).
542 cutDown2Sizes(Size,X,Y) :- Y is round(X/Size).
```

3.7.6 normalize(+Pairs1,-Pairs2): normalize a list of numbers Input list with values $M_1, M_2 \dots M_i$ with sum

```
_ egbarsGamma1.spy _
% output from ':- demos(egbarsGamma1).'
---| 10000 * gamma(10, 15) |-----
item frequency
  22
         3
         4
  21
        10
  20
  19
        11
  18
        34
  17
        76
  16
       144
       241 ~~
  15
       451 ~~~~~
  14
       665 ~~~~~~
  13
       942 ~~~~~~~
  12
      1321 ~~~~~~~~~~
  11
      10
      1593 -----
   9
      1378 -----
   8
       901 ~~~~~~~~
   7
       466 ~~~~~
   6
       169 ~~
   5
   4
        38
   3
         5
% runtime = 1.94279 sec(s)
```

Fig. 13 From §3.7.5.

egbarsGamma2.spy		
% outp	ut from ':- demos(egbarsGamma2).'	
10	0000 * 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	
% runt:	ime = 0.761094 sec(s)	

Fig. 14 From §3.7.5.

% outpu	ut from ':- demos(egbarsGamma3).'
10	0000 * 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
20	13 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
•	

Fig. 15 From §3.7.5.

Demonstration code: 570 egnormalize :- % see Figure 16 571 L=[a=10,b=5,c=20,d=50,e=5,c=10], normalize(L,Normals), 572 format('When w n is normalized it generates n w. n', 573 [L,Normals]). 574

3.8 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.

_egnormalize.spy .

% output from ':- demos(egnormalize).' When [a=10, b=5, c=20, d=50, e=5, c=10]

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

551 mostnormal1([X=V0|T],Sum,Y=N,[X=N1|Out],Most) :-

; mostnormal1(T,Sum,Y=N,Out,Most)).

-> mostnormal1(T,Sum,X=N1,Out,Most)

Temp is V + In, sumpairs(T,Temp,Out).

[a=0.1, b=0.05, c=0.2, d=0.5, e=0.05, c=0.1].

is normalized it generates

% runtime = 0 sec(s)

mostnormal(L,N,_).

550 mostnormal1([],_,Out,[],Out).

546 mostnormal(L,N,Most) :sumpairs(L,Sum),

N1 is V0/Sum,

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

sumpairs(T,V,S).

561 sumpairs([_=V|T],In,Out) :-

(N1 > N

560 sumpairs([],S,S).

Fig. 16 From §3.7.6.

543 normalize(L,N) :-

544

545

547

548 549

552

553 554

555

556

558

559

562

3.8.1 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.

```
575 sneak(X) :-
      load_files(X,[silent(true),if(changed)]).
576
```

3.8.2 spit(+Num1,+Num2,+Term): Print something, some-Useful for tracking a long process since it, sometimes. times, spits out a marker.

```
577 spit(N1,N2,X) :-
      (0 is N1 mod N2 -> blurt(X); true).
578
  3.8.3 blurt(+Term): print, then flush.
579 blurt(X) :-
```

```
write(user,X),flush_output(user).
580
   3.8.4 chars(+String): copy a file to the screen.
587 chars(File) :-
      see(File), get_byte(X), ignore(chars1(X)), seen.
588
589
590 chars1(-1) :- !.
591 chars1(X) :- put(X), get_byte(Y), chars1(Y).
  Demonstration code:
601 egchars :- % see Figure 18.
602
      chars('nowarranty.txt').
```

 $M_1+M_2+\ldots+M_i$ to a second list of numbers N_1, N_2, \ldots, N_i where $0 \le N_i \le 1$ and $N_1 + N_2 + ... + N_i = 1$.

```
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 taxt file
```

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 $\S3.8.4$ displays the contents of Figure 17 to the screen.

3.8.5 **barph(+Term)**: print a warning, then fail. A standard barph:

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

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

```
604 barphln(X) :-
605 here(File,Line),
606 format('%W> ~p@~p : ~p\n',[File,Line,X]),
607 fail.
608
609 here(File,Line) :-
610 source_location(Path,Line),
611 file_base_name(Path,File).
```

3.9 Meta-level predicates

614

```
3.9.1 tidy(+Rule0,-Rule1: remove stray "trues" from
a rule body.
612 tidy(A,C) :-
613 tidy1(A,B),
```

(B = (Head :- true) -> C=Head ; C=B).

```
615
616 tidy1(A,C) :- once(tidy2(A,C)).
617
618 tidy2(A,
                          A) :- var(A).
                    (A,TB)) :- var(A), tidy1(B,TB).
619 tidy2((A,B),
                    (TA,B)) :- var(B), tidy1(A,TA).
620 tidy2((A,B),
621 tidy2(((A,B),C),
                         R) :- tidy1((A,B,C), R).
                         R) :- tidy1(A,R).
R) :- tidy1(A,R).
622 tidy2((true,A),
623 tidy2((A,true),
                          R) :- tidy1(A,R).
624 tidy2((A;true),
625 tidy2((true;A),
                          R) :- tidy1(A,R).
626 tidy2((A;B), (TA;TB)) :- tidy1(A,TA), tidy1(B,TB).
627 tidy2((A->B), (TA->TB)) :- tidy1(A,TA), tidy1(B,TB).
628 tidy2(not(A), not(TA)) :- tidy1(A,TA).
629 tidy2((A :- B), R) :-
       tidy1(B,TB),(TB=true-> R=A; R=(A:-TB)).
630
631 tidy2((A,B), R) :-
       tidy1(A,TA), tidy1(B,TB),(TB=true -> R=TA; R=(TA,TB)).
632
633 tidy2(A,A).
```

Demonstration code:

```
646 egtidy :- % see Figure 19
647 Inl= (a :- b, true,c, (d->true;e)),
648 In2= (f :- true,(true;true;true),true),
649 tidy(Inl,Outl),
650 portray_clause(Outl),
651 tidy(In2,Out2),
652 portray_clause(Out2).
```

```
egtidy.spy
% output from ':- demos(egtidy).'
a :-
    b,
    C,
    (    d
    -> true
    ; e
    ).
f.
% runtime = 0 sec(s)
```

```
Fig. 19 From §3.9.1.
```

```
egdemand.spy _____

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

%W> failed(10>20)

% runtime = 0 sec(s)
```

Fig. 20 From §3.9.2.

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

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

```
% runtime = 0.100144 sec(s)
```

Fig. 21 From §3.9.4.

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

653 demand(X) :- X,!.

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

Demonstration code:

3.9.3 repeats (+Num,+Goal): run a goal N times

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

3.9.4 times(+Num,+Goal,-Time): time an execution

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

Demonstration code:

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

3.9.5 Lists/ conjuctions conversions. Convert a conjunction to a list:

681 c2l((X,Y),[X|Z]) :- !,c2l(Y,Z). 682 c2l(X,[X]).

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

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

Convert a list to a conjunction:

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

Convert disjunctions to a list.

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

3.9.6 clause1(?Head,?Body): does a goal match only one clause?

```
689 clause1(X,Y) :-
       singleton(X), clause(X,Y).
690
691
692 singleton(X) :-
693
       Sym='$singleton_',
694
       flag(Sym,_,0),
695
        + singleton1(Sym,X).
       flag(Sym,1,1).
696
697
698 singleton1(Sym,X) :-
       clause(X,_),flag(Sym,N,N+1),N > 1,!.
699
```

3.9.7 only(?Goal): can a goal only succeed once?

```
700 only(X) :-
       Sym='$only_',
701
       flag(Sym,_,0),
702
        + only1(Sym,X)
703
       flag(Sym,1,1).
704
705
706 only1(Sym,X) :-
       X, flag(Sym, N, N+1), N > 1, !.
707
708
709 solo(X) :-
       only(X), X.
710
```

4 Start-up commands

```
711 :- current_prolog_flag(max_integer,X),
712 Xl is X - 1,
713 retractall(inf(_)),
714 assert(inf(Xl)).
```

5 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

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 license 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.

 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,



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 memo #2.

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 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 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 IM-PLIED, 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 PRO-GRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORREC-TION.

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 PERMIT-TED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GEN- ERAL, 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 INACCU-RATE 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 POSSIBIL-ITY OF SUCH DAMAGES. END OF TERMS AND CONDITIONS