APPENDIX ONE
Constructing Problem Sets

The function TEST runs all problems in the list **problems**. The contents of **problems** can be set by loading one of the problem-sets provided, or the user can construct new problem-sets. A problem-set is a list of problems, where a problem is a list of the following:

1. problem-number
2. a list of premises, which are triples (formula, supposition, degree-of-justification)
3. a list of desired conclusions, which are pairs (formula, degree-of-interest)
4. a list of forwards prima facie reasons, which are quintuples (name,premises,conclusion,variables,strength)
5. a list of forwards conclusive reasons, which are quadruples (name,premises,conclusion,variables)
6. a list of backwards prima facie reasons, which are sextuples (name,forwards-premises,backwards-premises,conclusion,variables,strength)
7. a list of backwards conclusive reasons, which are quintuples (name,forwards-premises,backwards-premises,conclusion,variables)
8. an optional string describing the problem.

All formulas can be entered as pretty formulas instead.

For example, the following is a problem:

```
(2
 (("P" nil 1) ("A" nil 1))
 (("R" 1))
 (("pf-reason 1" (("P" is-inference)) "Q" nil 1)
  ("pf-reason 2" (("Q" is-inference)) "R" nil 1)
  ("pf-reason 3" (("A" is-inference)) "B" nil 1))
 (("con-reason 1" (("G" is-inference)) "J" nil 1)
  ("con-reason 2" (("E" is-inference)) "H" nil 1)
  ("con-reason 3" (("H" is-inference)) "K" nil 1)
  ("con-reason 4" (("F" is-inference)) "I" nil 1)
  ("con-reason 5" (("F" is-inference)) "(B @ E)" nil 1)
  ("con-reason 6" (("H" is-inference)) "(D @ G)" nil 1))
 (("pf-reason 4" nil (("C" nil)) (("~" "R") nil) nil 1)
  ("pf-reason 5" nil (("B" nil)) ("C" nil) nil 1))
 (("con-reason 7" nil (("F" nil)) (("~" "S") nil) nil 1)
  ("con-reason 8" nil (("G" nil)) ("V" nil) nil 1))
 "This is a hard problem.")
```

This list of problems in **problems** can be displayed in a more perspicuous form by running (DISPLAY-PROBLEMS). For example, if **problems** is the list consisting of just the above problem, this produces the following display:

```
Problem #2
This is a hard problem.
Given premises:
     P    justification = 1
     A    justification = 1
Ultimate epistemic interests:
     R    intere
    FORWARDS PRIMA FACIE REASONS
      pf-reason 1:   {P} ||=> Q   strength = 1
      pf-reason 2:   {Q} ||=> R   strength = 1
      pf-reason 3:   {A} ||=> B   strength = 1

    FORWARDS CONCLUSIVE REASONS
      con-reason 1:   {G} ||=> J   strength = 1
      con-reason 2:   {E} ||=> H   strength = 1
      con-reason 3:   {H} ||=> K   strength = 1
      con-reason 4:   {F} ||=> I   strength = 1
      con-reason 5:   {F} ||=> (B @ E)   strength = 1
      con-reason 6:   {H} ||=> (D @ G)   strength = 1

    BACKWARDS PRIMA FACIE REASONS
      pf-reason 4:   {} {C} ||=> ~R   strength = 1
      pf-reason 5:   {} {B} ||=> C   strength = 1

    BACKWARDS CONCLUSIVE REASONS
      con-reason 7:   {} {F} ||=> ~S   strength = 1
      con-reason 8:   {} {G} ||=> V   strength = 1
```

Problems can also be entered in this more perspicuous form, using the function MAKE-PROBLEM-LIST. For example, executing

```
(setf *problems* (make-problem-list
"Problem #1
This is a case of collective rebutting defeat
Given premises:
     P    justification = 1
     A    justification = 1
Ultimate epistemic interests:
     R    interest = 1

    FORWARDS PRIMA FACIE REASONS
      pf-reason 1:   {P} ||=> Q   strength = 1
      pf-reason 2:   {Q} ||=> R   strength = 1
      pf-reason 3:   {C} ||=> ~R   strength = 1
      pf-reason 4:   {B} ||=> C   strength = 1
      pf-reason 5:   {A} ||=> B   strength = 1

Problem #2
This is the same as #1 except that some reasons are backwards.
Given premises:
     P    justification = 1
     A    justification = 1
Ultimate epistemic interests:
     R    interest = 1

    FORWARDS PRIMA FACIE REASONS
      pf-reason 1:   {P} ||=> Q   strength = 1
      pf-reason 2:   {Q} ||=> R   strength = 1
      pf-reason 3:   {A} ||=> B   strength = 1

    BACKWARDS PRIMA FACIE REASONS
      pf-reason 4:   {} {C} ||=> ~R   strength = 1
      pf-reason 5:   {} {B} ||=> C   strength = 1
“))
```

yields the following set of *_problems_*:

```
((1 (("P" nil 1) ("A" nil 1)) (("R" 1))
  (("pf-reason 1" (("P" #<Compiled-function is-inference #x278A006>)) "Q" nil 1)
   ("pf-reason 2" (("Q" #<Compiled-function is-inference #x278A006>)) "R" nil 1)
   ("pf-reason 3" (("C" #<Compiled-function is-inference #x278A006>)) "~R" nil 1)
   ("pf-reason 4" (("B" #<Compiled-function is-inference #x278A006>)) "C" nil 1)
   ("pf-reason 5" (("A" #<Compiled-function is-inference #x278A006>)) "B" nil 1))
  nil nil nil "This is a case of collective rebutting defeat")
 (2 (("P" nil 1) ("A" nil 1)) (("R" 1))
  (("pf-reason 1" (("P" #<Compiled-function is-inference #x278A006>)) "Q" nil 1)
   ("pf-reason 2" (("Q" #<Compiled-function is-inference #x278A006>)) "R" nil 1)
   ("pf-reason 3" (("A" #<Compiled-function is-inference #x278A006>)) "B" nil 1))
  nil
  (("pf-reason 4" nil (("C" nil)) (("~" "R") nil) nil 1)
   ("pf-reason 5" nil (("B" nil)) ("C" nil) nil 1))
```

```
    nil "This is the same as #1 except that some reasons are backwards."))
```

When problems are entered in this form, the premises for forwards-reasons must be either pretty-formulas, or have the form *<pretty-formula , condition>* where *condition* is either *inference*, *percept*, or *desire*. For example, we might construct a problem as follows:

```
(setf *problems* (make-problem-list
"Problem #1
This is a case of collective rebutting defeat
Given premises:
     P    justification = 1
     A    justification = 1
Ultimate epistemic interests:
     R    interest = 1

    FORWARDS PRIMA FACIE REASONS
      pf-reason 1:   {P , <Q , desire> , <R , percept>} ||=> S   strength = 1
      pf-reason 4:   {B} ||=> C   strength = 1
      pf-reason 5:   {A} ||=> B   strength = 1
"))
```

with the resulting problem

```
(1 (("P" nil 1) ("A" nil 1)) (("R" 1))
  (("pf-reason 1"
    (("P" #<Compiled-function is-inference #x278A006>)
     ("Q" #<Compiled-function is-desire #x278A0F6>)
     ("R" #<Compiled-function is-percept #x278A1E6>))
    "S" nil 1)
   ("pf-reason 4" (("B" #<Compiled-function is-inference #x278A006>)) "C" nil 1)
   ("pf-reason 5" (("A" #<Compiled-function is-inference #x278A006>)) "B" nil 1))
  nil nil nil "This is a case of collective rebutting defeat")
```

Reasons can also contain variables, for use in pattern matching. For instance, here is a formulation of the paradox of the preface using variables:

```
Problem #15
Figure 18 -- the paradox of the preface.
Given premises:
     (P1 a)    justification = 1
     (P2 a)    justification = 1
     (P3 a)    justification = 1
     (S a)    justification = 1
     (T a)    justification = 1

Ultimate epistemic interests:
     ((Q1 a) & ((Q2 a) & (Q3 a)))    interest = 1

    FORWARDS PRIMA FACIE REASONS
      pf-reason 1:   {(P1 x)} ||=> (Q1 x)  variables = {x}   strength = 1
      pf-reason 2:   {(P2 x)} ||=> (Q2 x)  variables = {x}   strength = 1
      pf-reason 3:   {(P3 x)} ||=> (Q3 x)  variables = {x}   strength = 1
      pf-reason 4:   {(S x)} ||=> (R x)  variables = {x}   strength = 1
      pf-reason 5:   {(T x)} ||=> ~((Q1 x) & ((Q2 x) & (Q3 x)))  variables = {x}   strength = 1
      pf-reason 6:   {(S1 x)} ||=> ((T x) @ ~((Q1 x) & ((Q2 x) & (Q3 x))))  variables = {x}   strength = 1
      pf-reason 7:   {(S2 x)} ||=> ((T x) @ ~((Q1 x) & ((Q2 x) & (Q3 x))))  variables = {x}   strength = 1
      pf-reason 8:   {(S3 x)} ||=> ((T x) @ ~((Q1 x) & ((Q2 x) & (Q3 x))))  variables = {x}   strength = 1

    FORWARDS CONCLUSIVE REASONS
      con-reason 1:   {(Q1 x) , (Q2 x)} ||=> ((Q1 x) & (Q2 x))  variables = {x}   strength = 1
      con-reason 2:   {(Q2 x) , (Q3 x)} ||=> ((Q2 x) & (Q3 x))  variables = {x}   strength = 1
      con-reason 3:   {(Q1 x) , (Q3 x)} ||=> ((Q1 x) & (Q3 x))  variables = {x}   strength = 1
      con-reason 4:   {(R x) , ((Q1 x) & (Q3 x))} ||=> (S2 x)  variables = {x}   strength = 1
      con-reason 5:   {(R x) , ((Q2 x) & (Q3 x))} ||=> (S1 x)  variables = {x}   strength = 1
      con-reason 6:   {(R x) , ((Q1 x) & (Q2 x))} ||=> (S3 x)  variables = {x}   strength = 1
      con-reason 7:   {((Q1 x) & (Q2 x)) , ~((Q1 x) & ((Q2 x) & (Q3 x)))} ||=> ~(Q3 x)  variables = {x}
            strength = 1
      con-reason 8:   {((Q2 x) & (Q3 x)) , ~((Q1 x) & ((Q2 x) & (Q3 x)))} ||=> ~(Q1 x)  variables = {x}
            strength = 1
      con-reason 9:   {((Q1 x) & (Q3 x)) , ~((Q1 x) & ((Q2 x) & (Q3 x)))} ||=> ~(Q2 x)  variables = {x}
            strength = 1

    BACKWARDS CONCLUSIVE REASONS
      con-reason 11:   {} {(Q1 x) , (Q2 x) , (Q3 x)} ||=> ((Q1 x) & ((Q2 x) & (Q3 x)))  variables = {x}
            strength = 1
```

To expedite constructing problems in this latter form, the user may find it useful to cut and paste the following template for a single problem:

```
Problem #1
description of problem
Given premises:
    P    justification = 1
    P    justification = 1
    P    justification = 1
    P    justification = 1
    P    justification = 1

Ultimate epistemic interests:
    R    interest = 1
    R    interest = 1
    R    interest = 1

   FORWARDS PRIMA FACIE REASONS
     pf-reason 1:   {P , P , P} ||=> Q     variables = {x , y , z}    strength = 1
     pf-reason 1:   {P , P , P} ||=> Q     variables = {x , y , z}    strength = 1
     pf-reason 1:   {P , P , P} ||=> Q     variables = {x , y , z}    strength = 1
     con-reason 1:   {<P , condition> , <P , condition>} ||=> Q     variables = {x , y , z}    strength = 1
     con-reason 1:   {<P , condition> , <P , condition>} ||=> Q     variables = {x , y , z}    strength = 1

   FORWARDS CONCLUSIVE REASONS
     con-reason 1:   {P , P , P} ||=> Q     variables = {x , y , z}    strength = 1
     con-reason 1:   {P , P , P} ||=> Q     variables = {x , y , z}    strength = 1
     con-reason 1:   {P , P , P} ||=> Q     variables = {x , y , z}    strength = 1
     con-reason 1:   {<P , condition> , <P , condition>} ||=> Q     variables = {x , y , z}    strength = 1
     con-reason 1:   {<P , condition> , <P , condition>} ||=> Q     variables = {x , y , z}    strength = 1

   BACKWARDS PRIMA FACIE REASONS
     pf-reason 2:   {P , P , P} {Q , Q , Q} ||=> R     variables = {x , y , z}    strength = 1
     pf-reason 2:   {P , P , P} {Q , Q , Q} ||=> R     variables = {x , y , z}    strength = 1
     pf-reason 2:   {P , P , P} {Q , Q , Q} ||=> R     variables = {x , y , z}    strength = 1
     pf-reason 2:   {P , P , P} {Q , Q , Q} ||=> R     variables = {x , y , z}    strength = 1
     pf-reason 2:   {P , P , P} {Q , Q , Q} ||=> R     variables = {x , y , z}    strength = 1

   BACKWARDS CONCLUSIVE REASONS
     con-reason 2:   {P , P , P} {Q , Q , Q} ||=> R     variables = {x , y , z}    strength = 1
     con-reason 2:   {P , P , P} {Q , Q , Q} ||=> R     variables = {x , y , z}    strength = 1
     con-reason 2:   {P , P , P} {Q , Q , Q} ||=> R     variables = {x , y , z}    strength = 1
     con-reason 2:   {P , P , P} {Q , Q , Q} ||=> R     variables = {x , y , z}    strength = 1
     con-reason 2:   {P , P , P} {Q , Q , Q} ||=> R     variables = {x , y , z}    strength = 1
```

This template is contained in the file “Template”.

A precompiled version of a problem-set can be produced by first printing the contents of **problems*, producing a display like the following:

```
((1 (("P" nil 1) ("A" nil 1)) (("R" 1))
  (("pf-reason 1" (("P" #<Compiled-function is-inference #x278A006>)) "Q" nil 1)
   ("pf-reason 2" (("Q" #<Compiled-function is-inference #x278A006>)) "R" nil 1)
   ("pf-reason 3" (("C" #<Compiled-function is-inference #x278A006>)) "~R" nil 1)
   ("pf-reason 4" (("B" #<Compiled-function is-inference #x278A006>)) "C" nil 1)
   ("pf-reason 5" (("A" #<Compiled-function is-inference #x278A006>)) "B" nil 1))
  nil nil nil "This is a case of collective rebutting defeat")
 (2 (("P" nil 1) ("A" nil 1)) (("R" 1))
  (("pf-reason 1" (("P" #<Compiled-function is-inference #x278A006>)) "Q" nil 1)
   ("pf-reason 2" (("Q" #<Compiled-function is-inference #x278A006>)) "R" nil 1)
   ("pf-reason 3" (("A" #<Compiled-function is-inference #x278A006>)) "B" nil 1))
  nil
  (("pf-reason 4" nil (("C" nil)) (("~" "R") nil) nil 1)
   ("pf-reason 5" nil (("B" nil)) ("C" nil) nil 1))
  nil "This is the same as #1 except that some reasons are backwards."))
```

The next step is to replace the terms for the compiled functions by the corresponding expressions “desire”, “percept”, and “inference”, thus producing:

```
((1 (("P" nil 1) ("A" nil 1)) (("R" 1))
  (("pf-reason 1" (("P" inference)) "Q" nil 1)
   ("pf-reason 2" (("Q" inference)) "R" nil 1)
   ("pf-reason 3" (("C" inference)) "~R" nil 1)
   ("pf-reason 4" (("B" inference)) "C" nil 1)
   ("pf-reason 5" (("A" inference)) "B" nil 1))
  nil nil nil "This is a case of collective rebutting defeat")
 (2 (("P" nil 1) ("A" nil 1)) (("R" 1))
  (("pf-reason 1" (("P" inference)) "Q" nil 1)
   ("pf-reason 2" (("Q" inference)) "R" nil 1)
   ("pf-reason 3" (("A" inference)) "B" nil 1))
  nil
  (("pf-reason 4" nil (("C" nil)) (("~" "R") nil) nil 1)
   ("pf-reason 5" nil (("B" nil)) ("C" nil) nil 1))
  nil "This is the same as #1 except that some reasons are backwards."))
```

Then enclose the result in the following expression:

```
(setf *problems (quote    ...    ))
```

The files *Problems-sl.lsp* and *Problems-Q.lsp* were produced in this way. The advantage of using pre-compiled problem-sets is that they load much more quickly.