Spring, 1992 Name: _____

CS 342 — Principles of Programming Languages

Prolog Test (Problem Solving, Version 1)

Before you begin, please take a moment to look over the entire test so that you can budget your time.
The usual conventions from the Course Specification for problem solving tests apply.
A special rule is that *your programs only have to be able to answer queries of the form given in the examples for each problem.* If there are no queries that tell you that you have to be able to run the program "backwards," then you don't have to be able to run it backwards.
In your solutions you may use the following facts and rules from the textbook without further mention. Any other facts or rules that you wish to use must be written in your solution. If you wish to use facts or rules from your solutions in other parts of the exam, you must clearly state where we can find those. Remember that you can define "helping" predicates if you wish. You may also use simplify to convert from difference lists and back again unless the problem specifically prohibits that.

```
(infer (member X (cons X L)))
(infer (member X (cons Y M)) from (member X M))
(infer (addtoend nil X (cons X nil)))
(infer (addtoend (cons Y L) X (cons Y M)) from (addtoend L X M))
(infer (reverse nil nil))
(infer (reverse (cons X L) M) from (reverse L N) (addtoend N X M))
(infer (append nil L L))
(infer (append (cons X L) M (cons X N)) from (append L M N))
(infer (member X L) from (append L1 (cons X L2) L)).
(infer (lookup K A L) from (member (pair K A) L))
(infer (mult 0 Y 0))
(infer (mult X Y Z) from (minus X 1 V) (mult V Y W) (plus W Y Z))
(infer (fac 0 1))
(infer (fac N R) from (minus N 1 N1) (fac N1 R1) (mult R1 N R))
(infer (naive-sort L M) from (permutation L M) (ordered M))
(infer (<= X X))
(infer (<= X Y) from (less X Y))
(infer (ordered nil))
(infer (ordered (cons A nil)))
(infer (ordered (cons A (cons B L))) from (<= A B) (ordered (cons B L)))
(infer (permutation nil nil))
(infer (permutation L (cons H T))
      from (append V (cons H U) L) (append V U W) (permutation W T))
(infer (partition H (cons A X) (cons A Y) Z) from (<= A H) (partition H X Y Z))
(infer (partition H (cons A X) Y (cons A Z)) from (less H A) (partition H X Y Z))
(infer (partition H nil nil nil))
(infer (quicksort nil nil))
(infer (quicksort (cons H T) S)
      from
      (partition H T A B)
             (quicksort A A1)
             (quicksort B B1)
             (append A1 (cons H B1) S))
(infer (simplify (diff X X) nil))
(infer (simplify (diff (cons X Y) Z) (cons X W)) from (simplify (diff Y Z) W))
(infer (diffappend (diff L X) (diff X Y) (diff L Y))).
```

1. (6 points) Write a predicate `zero-or-more` such that `(zero-or-more` a l`)` holds if either l is an empty list or l is a list whose elements are all a. For example,

```
-> (infer? (zero-or-more foo nil))
Satisfied

-> (infer? (zero-or-more (lambda x x) (cons (lambda x x) (cons (lambda x x) nil))))
Satisfied

-> (infer? (zero-or-more foo (cons foo (cons bar nil))))
Not Satisfied
```

2. (15 points) You might want to do the second part of this problem first; or solve them simultaneously. All we ask is that you write your answers in the appropriate places.

   (a) Decide how to represent sets in Prolog, and define a predicate `is-set` such that `(is-set` s`)` holds if and only if s has the form you decided that sets should take.

   (b) Using your representation for sets, write a predicate `set-equality`, such that `(set-equality` s_1 s_2`)` holds if s_1 and s_2 represent sets with the same elements. Your `set-equality` only has to be able to work with two set arguments, it doesn't have to be able to find a set equal to one of the arguments. Remember that order does not matter in sets.

3. (15 points) For this problem, an *element* is a colored integer, represented by a prolog term of the form `(red` n`)`, `(white` n`)`, or `(blue` n`)`, where n is a number; for example, `(red 3)`, `(white 4)`, and `(blue 5)`. A Dutch computer scientist, Edsgar Dijkstra, invented the following problem. Write a predicate `dutch` that takes a list of elements colored red, white, and blue, reorder the list so that all the red elements are first, then all the white, and finally all the blue. This reordering should preserve the original relative order of elements of the same color. (This makes it look like the Dutch national flag.) For example,

```
-> (infer? (dutch nil Flag) (print Flag))
nil
Satisfied

-> (infer? (dutch (cons (red 1) (cons (white 2) (cons (blue 3)
                        (cons (red 4) (cons (white 5) nil)))))
                  Flag)
           (print Flag)
(cons (red 1) (cons (red 4) (cons (white 2) (cons (white 5) (cons (blue 3) nil)))))
Satisfied

-> (infer? (dutch (cons (red 4) (cons (white 5) (cons (blue 3)
                        (cons (red 1) (cons (white 2) nil)))))
                  Flag)
           (print Flag)
(cons (red 4) (cons (red 1) (cons (white 5) (cons (white 2) (cons (blue 3) nil)))))
```