Homework 15: λProlog

Due:

This is a "warm-up" homework for λProlog.

Unless we specifically request, your λProlog code does not have to run "backwards". (But you may want to see which ones can do that.) In the examples you will see "no more solutions" or "yes" printed; your program does not have to have exactly this output.

The first few problems deal with lists in λProlog. Try typing `describe '::'.` to the interpreter for a description of the list building operation. You'll often have to parenthesize lists, for example `(1::nil)`, so it's safer to always parenthesize them. See also the file `/usr/unsup/ess/elp/lib/rel/lists.mod` for some examples of operations on lists. The operations in that file are not built-in to λProlog, but you can use them by putting `import lists.` in your module.

Specify the following relations in λProlog.

1. The relation `find_last` such that

   ```
   type find_last  (list T) -> T -> o.
   ```

   and `(find_last L X)` holds iff `X` is the last item in the list `L`. For example.

   ```
   ?- use "hw15".
   Solving ...
   Loading module hw15 from file "./hw15.mod".
   Inferring types on module hw15.
   ?- find_last nil 3.
   Solving ...
   no
   ?- find_last (1::2::3::nil) 3.
   Solving ...
   Query succeeded
   no more solutions
   ?- find_last (1::2::3::nil) Y.
   Solving ...
   Y == 3
   no more solutions
   ?- find_last L 3.
   Solving ...
   L == 3 :: nil;
   L == Y: int :: 3 :: nil
   yes
   ```

2. The relation `consecutive` such that

   ```
   type consecutive  (list T) -> T -> T -> o.
   ```

   where `consecutive L X Y` holds iff in the list `L`, the items `X` and `Y` appear in that order. For example,

   ```
   ?- consecutive (1::2::nil) 1 2.
   Solving ...
   Query succeeded
   ```

```
no more solutions
?- consecutive (1::1::2::nil) 1 2.
Solving ...
Query succeeded
no more solutions
?- consecutive (1::nil) 1 2.
Solving ...
no
?- consecutive (4::1::3::2::nil) 1 2.
Solving ...
no
```

3. The relation `delete_first` such that

   ```
   type delete_first (list T) -> T -> (list T) -> o.
   ```

   and `(delete_first L1 X L2)` holds if `L2` is like `L1` except that the first occurrence of `X` has been deleted. For example,

   ```
   ?-  delete_first (1::1::3::4::1::nil) 1 L.
   Solving ...
   L == 1 :: 3 :: 4 :: 1 :: nil
   no more solutions
   ?- delete_first (3::nil) 1 L.
   Solving ...
   L == 3 :: nil
   no more solutions
   ```

   Hint, you may want to use `not` to get exactly the behavior given in these examples.

4. The relation `delete` such that

   ```
   type delete (list T) -> T -> (list T) -> o.
   ```

   and `(delete L1 X L2)` holds if `L2` is like `L1` except that every occurrence of `X` has been deleted. For example,

   ```
   ?-  delete (1::1::3::4::1::nil) 1 L.
   Solving ...
   L == 3 :: 4 :: nil
   no more solutions
   ?- delete (3::nil) 1 L.
   Solving ...
   L == 3 :: nil
   no more solutions
   ```

5. The relation `delete_last` such that

   ```
   type delete_last (list T) -> T -> (list T) -> o.
   ```

   and `(delete_last L1 X L2)` holds if `L2` is like `L1` except that the last occurrence of `X` has been deleted. For example,

```
?-  delete_last (1::1::3::4::1::nil) 1 L.
Solving ...
L == 1 :: 1 :: 3 :: 4 :: nil
yes
?- delete_last (3::nil) 1 L.
Solving ...
L == 3 :: nil
no more solutions
```

In the following problems, you are to build on the code in `~leavens/src/lambda-prolog/unary.mod`. (You should copy that and import it into your answer module.)

6. Write a relation `expt` such that

   ```
   type expt nat -> nat -> nat -> o.
   ```

   and such that `(expt X Y Z)` holds if `Z` is `X` raised to the `Y` power. For example,

   ```
   ?- to_nat 3 THREE, to_nat 2 TWO, expt THREE TWO Z, to_int Z N.
   Solving ...
   THREE == s (s (s z))
   TWO == s (s z)
   Z == s (s (s (s (s (s (s (s (s z))))))))
   N == 9
   yes
   ```

7. Write a relation `iterate` such that

   ```
   type iterate nat -> (T -> T) -> T -> T -> o.
   ```

   and such that `(iterate N Fun X Z)` holds if `Z` is the result of iterating the function `Fun N` times on `X`. The following examples use lambda abstraction.

   ```
   ?- iterate z (x \ x) z Ans.
   Solving ...
   Ans == z
   no more solutions
   ?- iterate (s (s z)) (x : nat \ (s x)) z Ans.
   Solving ...
   Ans == s (s z)
   no more solutions
   ?-  iterate (s (s z)) (x : nat \ (s (s (s x)))) z Ans.
   Solving ...
   Ans == s (s (s (s (s z)))))
   no more solutions
   ```

1 Extra Credit

The following explore the conceptual modeling (or database) aspects of λProlog.

8. Encode your weekly schedule as a set of λProlog facts and rules. Show how it would be used by writing some queries.

9. Encode your family tree as a set of λProlog facts and rules. Can you make the program answer questions like "is Sue my second cousin once removed"? Can you make the program answer questions like what is the relationship between Sue and Fred"?