CSE 341 — Additional CLP($\mathcal{R}$) Discussion Questions – Answer Key

These questions use the following CLP($\mathcal{R}$) rules:

```
member(X,[X|Xs]).
member(X,[Y|Ys]) :- member(X,Ys).

member_cut(X,[X|Xs]) :- !.
member_cut(X,[Y|Ys]) :- member(X,Ys).

length([],0).
length([_|Xs],N) :- N>0, length(Xs,N-1).
```

1. What are all the answers that CLP($\mathcal{R}$) returns for the following goals? (If there are an infinite number give the first several; if there are none, say so.)

   ```
   member(squid,[clam,squid,tuna])

   member(squid,X)

   member_cut(squid,[clam,squid,tuna])

   member_cut(squid,X)

   member(X,[a,b,c,d]), member(X,[c,d,e,f])

   length([A,B,C],N)

   length([A,B,C|Cs],N)
   ```

   Answers: try them and see . . .

2. Write a CLP($\mathcal{R}$) rule to find the average of a list of numbers. Fail if the list is empty.

   ```
   sum([],0).
   sum([X|Xs],X+S) :- sum(Xs,S).

   average(Xs,S/L) :- length(Xs,L), L>0, sum(Xs,S).
   ```

3. Write a CLP($\mathcal{R}$) rule `range(Lo,Hi,List)` that succeeds if `List` consists of all the numbers between `Lo` and `Hi` inclusive. (Assume that `Lo` and `Hi` are integers.) If `Lo` is greater than `Hi`, `List` should be empty.

   ```
   range(Lo,Hi,[]) :- Lo > Hi.
   range(Lo,Hi,[Lo|Xs]) :- Lo<=Hi, range(Lo+1,Hi,Xs).
   ```

4. Given your `range` rule from Question 3, what are all the results for the following goals?

   ```
   range(2,5,A)
   range(5,2,A)
   range(A,B,[2,3,4,5,6])
   range(A,B,[2,4,6])
   ```

Answers: try them and see ...

5. Write another version of `range` that also takes a `Step` parameter: `range(Lo,Hi,Step,List)`. `Lo`, `Hi`, and `Step` don't have to be integers for this version. For example, `range(2.2, 3.0, 0.2, L)` should succeed with `L = [2.2,2.4,2.8,3.0]`.

```
range(Lo,Hi,Step,[]) :- Lo > Hi.
range(Lo,Hi,Step,[Lo|Xs]) :- Lo<=Hi, range(Lo+Step,Hi,Step,Xs).
```

6. Is CLP($\mathcal{R}$) statically typed? Is it type safe?

   CLP($\mathcal{R}$) is not statically typed, but it is type safe. For example, you can consult the following file without CLP($\mathcal{R}$) complaining. However, if you try `go1`, you get an answer of "no" because of a type problem. (It would probably be better to give an explicit error message, but at least the system isn't treating the string as a number.)

```
go1 :- X = 3, Y="squid", X+Y=Z.
```

7. Compare the way parameters are passed in Haskell and CLP($\mathcal{R}$).

   Parameters in Haskell are passed using lazy evaluation (which semantically is the same as call-by-name). The actual parameter isn't evaluated until the value of the formal parameter is needed in the function that was called; then it is evaluated and the value cached in case it is needed again. There is a limited form of pattern matching available for specifying formal parameters. Information only flows from the actual parameter to the formal.

   Parameters in CLP($\mathcal{R}$) are passed by setting up equality constraints between the actual and formal parameters. (Thus either the actual or formal parameter, or both, can be expressions as well as variables or constants.) These constraints are solved, along with the others in the body of the goal. The result is that information can flow both in and out via the parameters — or even in both directions at once (in effect) as the constraints are solved.