CS 430 Programming Languages
Spring 2016
Module 17 Lab

JAMES MADISON UNIVERSITY

1. Open two windows and set up one window with an editor open on a new file called lab17.pl, and the other running gprolog.

2. Lists are almost as important in Prolog as they are in Haskell, and there is a similar syntax for defining them. The empty list is designated []. The list whose first element is A and remaining elements are the list B is designated [A|B]. Note that B is a list. So the notation [A|B] in Haskell terms is A:B (except that in Haskell these variables would have to be lowercase). A list can also be made by just putting its elements between brackets separated by commas. Type [1,2,3,4]=[A|B] . followed by return in gprolog.

3. Prolog has lots of built-in list predicates. Type member(X,[1,2,3,4]). Hit semicolon until the goal fails, or type a to see all the ways this goal can be satisfied. Type member(1,[A|B]). and hit semicolon several times before just hitting a return. What is happening?

4. One difference between Haskell and Prolog with regard to list syntax is that the concatenation operator can be used multiple times in Haskell, so a:b:c:[] is the list [a,b,c]. In Prolog, this would be [a,b,c|[]] instead (or just [a,b,c], of course). But pattern matching with the concatenation operator in Prolog is just as powerful as it is in Haskell; for example [X1,X2,X3,X4|T] matches a list with at least four elements; X1 is instantiated to the first, X2 to second, and so forth, and T is instantiated to the rest of the list after the fourth element. (You will need this sort of pattern matching in the PA for this module.)

5. Lets make a predicate size(L,N) that counts how many elements are in a list L. We know that the size of an empty list is 0: size([],0). We also know that the size of a non-empty list is 1 more than the size of that list with its first element removed: size([_|T],N) :- size(T,M), N is M+1. Note that the special infix predicate is must be used to get Prolog to evaluate an arithmetic expression. Type these rules into lab17.pl, **save the file**, and consult it. Experiment to see how these rules work. Write tests for the size predicate.

6. Note that the order in which things occur here is very important: if the second rule has its conjuncts reversed, there will be an instantiation error because M will not have a value when M+1 is evaluated. Furthermore, one must keep in mind the order in which the inference engine tries to satisfy its goals. Notice too that we are using recursion in this predicate; we will use recursion to process lists frequently. Finally, Prolog has a built-in predicate length that works like size.

7. Write rules in lab17.pl to implement a predicate end(L,X) that places the last element of list L in X. It fails for the empty list. There is a built-in predicate called fail (no arguments) that you can use in a rule to obtain this behavior. Also write tests for this predicate. Note: Prolog has a built-in last predicate that works like end.

8. Write rules to implement the predicate max(L,M) whose first argument is a list and whose last argument is the maximum number in the list. The predicate should fail if the list is

empty. There are also built-in numeric in-fix comparison operators < and =< for less than and less than or equal to. You may assume that the list contains only numbers. Write tests for this predicate. Prolog has the predicates min_list, max_list, and sum_list built-in.

8. Write rules to implement the predicate concatenate(A,B,C) where C is the result of appending A and B. Write tests for it. Prolog has a built-in append predicate, of course.

9. Prolog is more relaxed about types than Haskell. In Haskell, a list must contain elements that are all the same type. In Prolog, a list can contain elements of any type, including other lists. Hence the list [1,a,[b,[]],4] is perfectly fine in Prolog. It helps to be able to distinguish the types of list elements. The built-in predicate list(X) does this, succeeding if X is a list and failing otherwise.

10. Lets use the list predicate to make a predicate that mimics the built-in predicate flatten(X,Y). A flattened list is a list that contains only non-list elements. When given a non-list element X, flatten(X,Y) succeeds if Y is the list with X in it. When given a list X, which may contain lists (which may themselves contains lists, etc.), it succeeds when Y is a list containing all the non-list elements of X in the order in which they occur in X. For example, flatten(a,[a]) succeeds, as does flatten([1,a,[b,[],[3]],4],[1,a,b,3,4]). Write your own version of this method called flattn(X,Y). Write test cases for your predicate.

11. We can also write predicates that do things to elements of a list, similar to map in Haskell. For example, write a predicate prefix_all(X,Y,Z) that succeeds when X is a value, Y is a list of lists, and Z is Y with X attached as the first element of every element of Y. For example, the following succeeds.
    prefix_all(0, [[],[1],[1,2],[1,2,3]], [[0],[0,1],[0,1,2],[0,1,2,3]])
    (In Haskell, we could write this as the function prefixAll x z = map (x:) z.) Write tests for prefix_all.

12. Finally, lets write a predicate that uses prefix_all. The power set of a set *S* is the set of all subsets of *S*. For example, the power set of {1,2,3} is the set {{}, {1}, {2}, {3}, {1,2}, {1,3}, {2,3}, {1,2,3}}. If we represent sets as lists in Prolog, then we can generate the power set of a set as a list of lists. The predicate power_set(S,P) succeeds if P is the power set of S (in some order). We can use it as a power set generator, that is, in expressions like power_set([1,2,3],X), where X is instantiated to a list representing the power set of {1,2,3}. Use prefix_all to write power_set.

13. To test power_set, we have to get the result in some regular order. The easiest thing to do is to sort the result. Then we can write a test like the following.
    power_set([1,2,3],P), sort(P,Q), Q==[[],[1],[1,2],[1,2,3],[1,3],[2],[2,3],[3]].
    This goal will succeed if power_set generates a list with the right elements no matter which order it generates them in.

14. At this point, you have all the tools you need to do the PA for this module.