2013 February 18

Prolog Basic
Example Exam Questions

1.

Write a Prolog predicate **split(TheList, Evens, Odds)** that asserts the following.

- **Odds** contains all the items in the odd positions of **TheList** that are not the empty list
- **Evens** contains all the items in the even positions of **TheList**, including the empty list

  Use = and \= to distinguish cases.

2.

Define the predicate **odd_list(a_list)** where a_list is a list of atoms. The predicate asserts the list contains an odd number of elements. Do NOT USE the length predicate or numbers.

```
?- odd_list([]).
No
?- odd_list([one]).
Yes.
?- odd_list([one, two]).
No.
?- odd_list([one, two, three]).
Yes.
?- odd_list(one).
No.
```

3.

Consider the following predicate, `mystery(List1,List2,Result).`

```
mystery([],L2,L2).
mystery(L1,[],L1) :- L1 = [_|_].
mystery([H1|T1],[H2|T2],[H1|T3]) :- mystery(T1,[H2|T2],T3).
mystery([H1|T1],[H2|T2],[H2|T3]) :- mystery([H1|T1],T2,T3).
```

A What does the following query produce, if semi-colon, ;, is used to find more than one answer.

```
mystery([1,2],[a,b],R).
```

B Give a brief English description of the predicate `mystery`.

C Replace the last two rules of the predicate `mystery` with a single rule.

4.

What is the difference between the following two rules?

```
blah :- a(X) , b(X).
blah :- a(_) , b(_).
```

5.

Write a definition of the predicate **removeNil (List, Rlist)** that asserts **Rlist** is the same as **List** except that all instances of the item **[]** (the empty list) have been removed at all levels of **List.**

6.

Define a Prolog predicate remove2nd(List, NewList) that asserts that NewList is the same a List except that the second top-level item is removed.

```
remove2nd([a, b, c], [a, c]) → yes
remove2nd([a, b], X) → X = [a]
remove2nd ([a], X) → X = [a]
```

7.

Write a Prolog predicate facti (N, F) that asserts F is the factorial of N. Ensure that it works for the query facti (N, F).

8.

Define a Prolog predicate **flatten (List, FlattenedList)** that asserts List is any nested list of atoms and FlattenedList is the same list with the nesting removed. The atom [ ] should also be removed. Your predicate should only produce one answer. You may use the built-in predicates \+ (not), ! and append. Do not use a helper predicate.

```
?- flatten([a, [[b, c], d], [[e]], [f]], X).
X = [a, b, c, d, e, f];
no

?- flatten([a, [[]], [[c, d], e]], X).
X = [a, c, d, e] ;
no
```

9.

Write a prolog predicate **insert_nth(item, n, into_list, result)** that asserts that **result** is the list **into_list** with **item** inserted as the **`n`**'th element into every list at all levels. Counting begins at 1.

Precondition: n ≥1 and n ≤ 1+length(shortest list at any level in `list`)

10.

Write a Prolog predicate, `remove-nth(Before, After)` that asserts the `After` list is the `Before` list with the removal of every `n`'th item from every `list` at all levels. Counting begins at 1.

Precondition: N ≥1, `Before` and `After` are lists.

11.

Write a predicate `nth(N,Alist,Elem)` such that Elem is the N'th item in the list Alist. `nth(1,Alist,Elem)` is true for the first item in the list.

12.

Write a predicate `index(Array,[I1,I2,...,In],Elem)` such that Array[I1,I2,...,In] = Elem. There is no fixed size for the number of dimensions. You may use the predicate `nth` from part A if you wish but you do not have to. Assume index value 1 is the first item in the corresponding dimension.

13.

Assume the prolog predicate `gt(A, B)` is true when `A` is greater than `B`. Use this predicate to define the predicate `addLeaf(Tree, X, NewTree)` which is true if `NewTree` is the `Tree` produced by adding the item `X` in a leaf node. Tree and NewTree are binary search trees. The empty tree is represented by the atom `nil`.

14.

Write a Prolog predicate to remove the N'th item from a list.

15.

The predicate **maximum(X, Y, M)** is true if and only if **M** is the maximum integer of **X** and **Y**. The following is a variation of the definition that was discussed in class.

```
maximum(X, Y, M) :- X >= Y , M = X  ;  Y >= X , M = Y.
```

Is the predicate correct? Are there any circumstances when it may fail to give the expected answer? If it does fail, correct the definition in the simplest possible way.

16.

Define a Prolog predicate sort(X, Y) that asserts that given X is a list of integers then Y is the same list but sorted in ascending order. Your algorithm **MUST** be the following

Repeatedly choose the smallest remaining element from X and add it to Y.

Hint: use a helper predicate called smallest.