Tutorial for week 4 (15–19 Oct)

Lists and recursion

1. Reversing lists

(a) Define a predicate `reverse/2` such that `reverse(L,M)` holds when `M` is the reverse of `L`.

For this part, your definition may use the built-in `append/3` predicate but no other built-in or helper functions.

Trace the behaviour of query `?- reverse([1,2,3,4],X)`.

What happens if you pose query `?- reverse(X,[1,2,3,4])`?

What is the complexity of this implementation of `reverse/2` (assuming `append/3` is linear in the size of its first argument)?

(b) Now we will implement reverse more efficiently, using a technique called accumulators. Write a helper predicate `reverse3` such that `reverse3(L,M,N)` holds if `N` is the result of concatenating `M` with the reverse of `L`.

Define a reverse predicate `qrev/2` in terms of `reverse3(L,M,N)`, and retry the tracing experiments above using `qrev/2`. This implementation should have no calls to `append`.

2. Shuffling

Given two lists, a shuffle is a list consisting of alternating elements from the two lists, starting with the first. If one of the lists is empty, then shuffling just returns the other list.

For example:

```
shuffle([],[1,2,3,4],[1,2,3,4]).
shuffle([1,2],[3],[1,3,2]).
shuffle([1,2],[3,4],[1,3,2,4]).
```

Here is a simple definition of `shuffle/3`:

```
simple shuffle([],L,L).
simple shuffle(L,[] ,L).
simple shuffle([X|L],[Y|M],[X,Y|N]) :- simple shuffle(L,M,N).
```
What happens if you ask the query:

?- simple shuffle([1,2],[3,4],X).

What about ?- simple shuffle(X,Y,[1,2,3,4])?

Define an improved shuffle/3 such that if L1 and L2 are ground then shuffle(L1,L2,L3) returns exactly one answer.

3. (*) Bridge dealing

In a four-player game of bridge, each player gets 13 cards, dealt in order. Write a predicate deal(Cards, H1, H2, H3, H4) that takes a first argument, and succeeds by binding H1 to the 13 cards received by player 1 in the deal, etc.

Hint: One strategy is to write four helper predicates deal1 that deals to player 1, deal2 that deals to player 2, etc.

4. (**) Cutting the deck

Write a predicate cut/3 such that if L is a list with even length, then cut(L, M, N) succeeds by binding M to the first half of L and N to the second half.

Hint: One can get M and N by generating possible splits of L using append/3, and defining a predicate same_length/2 that holds of two lists whenever they have the same length (ignoring their element values). Another way to do this is to use the built-in length/2 predicate.