These solution notes are as submitted in January 2003

Artificial Intelligence I 2003 Paper 5 Question 7 (SBH)

Covers elements from most of the course.

The following Prolog relation appends a list `A` to a list `B` to give a list `C`.

```
append([],Y,Y).
append([H|T],Y,[H|Z]) :- append(T,Y,Z).
```

(*a*) Using the `append` relation, write a Prolog predicate `insert(X,Y,Z)` that is true if `X` can be inserted into a list `Y` to give a list `Z`. Your relation should be capable of using backtracking to generate all lists obtained from `Y` by inserting `X` at some point, using a query such as:

```
insert(c,[a,b],Z).
```

to obtain `Z=[c,a,b]`, `Z=[a,c,b]`, and `Z=[a,b,c]` and it should generate each possibility exactly once. [5 marks]

Answer:

```
insert(X,[],[X]) :- !.
insert(X,Y,Z) :- append(A,B,Y), append(A,[X],A2), append(A2,B,Z).
```

The cut is essential, otherwise backtracking causes everything to be generated twice.

(*b*) Using the `insert` relation, write a Prolog predicate `perm(X,Y)` that is true if a list `Y` is a permutation of a list `X`. Again, your predicate should respond to a query such as

```
perm([a,b,c],Y)
```

by using backtracking to generate all permutations of the given list. [5 marks]

Answer:

```
perm([],[]).
perm([H|T],Y) :- perm(T,Y2), insert(H,Y2,Y).
```

(*c*) We have a list of events `[e1,e2,...,en]`. A partial order can be expressed in Prolog by stating

```
before(e3,e4).
before(e1,e5).
```

and so on, where `before(a,b)` says that event `a` must happen before event `b` (although not necessarily immediately before). No ordering constraints are imposed other than those stated using `before`.

Given a list of events, a *linearision* of the list is any ordering of its events for which none of the `before` constraints are broken. Given the example above and the list `[e1,e2,e3,e4,e5]`, one valid linearision would be `[e3,e1,e2,e5,e4]`. However `[e4,e2,e1,e5,e3]` is not a valid linearision because the first `before` constraint does not hold.

Using the `perm` predicate or otherwise, and assuming that your Prolog program contains `before` constraints in the format suggested above, write a Prolog predicate `po(X,Y)` that is true if `Y` is a valid linearision of the events in the list `X`. Your relation should be capable of using backtracking to generate all valid linearision as a result of a query of the form

```
po([e1,e2,e3,e4,e5],Y).
```

[8 marks]

Answer:

```
valid([]).
valid([X]).
valid([A,B|T]) :- valid([B|T]), \+before(B,A), valid([A|T]).

po(X,Y) :- perm(X,Y), valid(Y).
```