Artificial Intelligence I

The following Prolog relation appends a list \( A \) to a list \( B \) to give a list \( C \).

\[
\text{append}([], Y, Y).
\]
\[
\text{append}([H|T], Y, [H|Z]) :- \text{append}(T, Y, Z).
\]

\((a)\) Using the \texttt{append} relation, write a Prolog predicate \texttt{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:

\[
\text{insert}(c, [a, b], Z).
\]
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

\((b)\) Using the \texttt{insert} relation, write a Prolog predicate \texttt{perm}(X, Y) that is true if a list \( Y \) is a permutation of a list \( X \). Your predicate should respond to a query such as

\[
\text{perm}([a, b, c], Y)
\]
by using backtracking to generate all permutations of the given list. \[6\] marks

\((c)\) We have a list of events \([\text{e}1, \text{e}2, \ldots, \text{e}n]\). A partial order can be expressed in Prolog by stating

\[
\text{before}(\text{e}3, \text{e}4).
\]
\[
\text{before}(\text{e}1, \text{e}5).
\]

and so on, where \texttt{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 \texttt{before}.

Given a list of events, a \textit{linearisation} of the list is any ordering of its events for which none of the \texttt{before} constraints are broken. Given the example above and the list \([\text{e}1, \text{e}2, \text{e}3, \text{e}4, \text{e}5]\), one valid linearisation would be \([\text{e}3, \text{e}1, \text{e}2, \text{e}5, \text{e}4]\). However, \([\text{e}4, \text{e}2, \text{e}1, \text{e}5, \text{e}3]\) is not a valid linearisation because the first \texttt{before} constraint does not hold.

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

\[
\text{po}([\text{e}1, \text{e}2, \text{e}3, \text{e}4, \text{e}5], Y).
\]
\[9\] marks