Recall that a *definite clause* is an atom (or fact) or a rule of the form

\[
h := b_1, b_2, \ldots, b_m
\]

where \( h \) and all \( b_i \)'s are atoms. This homework focuses on definite clauses that are *propositional* in that all predicates have arity 0. (That is, there are no terms.) Let us agree to encode propositional clauses as lists, with an atom \( f \) encoded as \([f]\) and a rule \( h := b_1, \ldots, b_m \) as \([h, b_1, \ldots, b_m]\). A finite list of propositional clauses can then be encoded as a list of lists — e.g.

\[
\begin{align*}
&h := c. \\
&h := f, g. \\
&f := g. \\
&c := f, h, a. \\
&g.
\end{align*}
\]

as \([h, c], [h, f, g], [f, g], [c, f, h, a], [g]\].

1. Define a predicate \( \text{lc}(+\text{KB}, ?C) \) in Prolog that collects in \( C \) all atoms that are logical consequences of \( \text{KB} \), allowing us to check if an atom \( X \) is a logical consequence of \( \text{KB} \) through the predicate

\[
\text{query}(X, \text{KB}) :- \text{lc}(\text{KB}, C), \text{member}(X, C).
\]

For example,

\[
?- \text{lc}([h, c], [h, f, g], [f, g], [c, f, h, a], [g]), C). \\
C = [h, f, g] ? ; \\
no
\]

2. Extend \( \text{query}(X, \text{KB}) \) to definite clauses, defining a predicate \( \text{queryRule}(\text{List}, \text{KB}) \) that is true precisely when the rule encoded by \( \text{List} \) is a logical consequence of \( \text{KB} \).

Some runs to cover

\[
\begin{align*}
&?- \text{queryRule}([a, b], [[a], [b], [c]]). \\
&\text{yes} \\
&?- \text{queryRule}([b, a], [[a], [b], [c]]). \\
&\text{no} \\
&?- \text{queryRule}([a, b], [[a, b], [c]]). \\
&\text{yes}
\end{align*}
\]

\(^1\)For any extensions, email your demonstrator/marker, David Woods (dwoods@tcd.ie).
?- queryRule([a,d],[[a,b,c],[c]]).
   no

?- queryRule([a,b,c],[[a,d],[d,b,c]]).
   yes

?- queryRule([a,a],[[b],[c,b]]).
   yes