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 \([ [h, c] , [h, f, g] , [f, g] , [c, f, h, a] , [g] ]\). The binary predicate

\[ \text{prove}(\text{Node}, \text{KB}) \]

below is an approach to calculating if all the clauses in \( \text{Node} \) are logical consequences of \( \text{KB} \).

\[
\begin{align*}
\text{prove}([], \text{KB}). \\
\text{prove}(\text{Node}, \text{KB}) & : - \text{arc}(\text{Node}, \text{Next}, \text{KB}), \text{prove}(\text{Next}, \text{KB}). \\
\text{arc}([H|T], N, \text{KB}) & : - \text{member}([H|B], \text{KB}), \text{append}(B, T, N). \\
\end{align*}
\]

1. As claimed at lecture, the predicate \( \text{prove} \) above is not complete. In fact, given an atom \( g \), there is a set \( \text{KB} \) of propositional definite clauses such that \( \text{KB} \models g \) but not \( \text{prove}([g], \text{KB}) \). Give the simplest example of such a \( \text{KB} \).

2. Define a predicate \( \text{lc}(+\text{KB}, ?\text{C}) \) in Prolog that collects in \( \text{C} \) 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

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

For example,

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

3. 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 \texttt{List} is a logical consequence of \texttt{KB}.

**Some runs to cover**

\begin{verbatim}
| ?- queryRule([a,b],[[a],[b,c]]). yes
| ?- queryRule([b,a],[[a],[b,c]]). no
| ?- queryRule([a,b],[[a,b,c],[c]]). yes
| ?- 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
\end{verbatim}