Tutorial for week 10 (week of 23rd Nov)

1. Recall the definition of logical consequence:
   a formula $\mathbf{G}$ is a logical consequence of formulae $F_1, F_2 \ldots F_n$ if and only if, for all interpretations $\mathcal{I}$,

   $$\textit{if } \mathcal{I} \models F_1 \textit{ and } \ldots \textit{and } \mathcal{I} \models F_n, \\ \textit{then } \mathcal{I} \models \mathbf{G}.$$

   Standard logical consequence is *monotonic*:

   > given sets of statements S, T and a goal G,
   > if $S \models G$, then $S \cup T \models G$.

   Use the definition of logical consequence to show that logical consequence is indeed monotonic.

2. The CWA applied to a set of definite clauses gives a consistent theory. This is not true in general for first order theories.

   Show that using the CWA for the theory below results in an inconsistent theory:

   ```
   happy(jane) ∨ happy(john)
   ```

   Why is this problematic?

3. The Clark completion algorithm works exactly the same for general programs (with negation in the body of clauses) as for definite clauses.

   What is the Clark completion for the following general program?

   ```
   p(a) :- \+ q(X).
   q(a).
   ```

4. Explain how standard negation by failure works, and show that using negation by failure the query `?- \+ p(a).` succeeds for the program above.

   `not p(a)` is not a logical consequence of the completion of the program. What does this tell us about the soundness of negation by failure in general as a logic inference rule?

5. Explain how search is extended in λProlog to allow goals which are *implications*, `P => Q`, as a form of hypothetical reasoning.

   Show how a positive answer is found for the following query, where there are no program clauses:

   ```
   ?- (a => b => c) => (a => b) => a => c.
   ```

   Recall that `=>` associates to the right, and that a statement `P => Q` is equivalent to `Q :- P`.

6. Show how a positive answer is derived for the following query, given the typing and program clauses shown.

   ```
   kind i type.        % type of individuals
   type s        i -> i.
   type pos      i -> o.
   type natural  i -> o.

   pos (s(X)) :- natural X.
   natural (s(X)) :- natural X.

   ?- pi Y\ (natural Y => pos (s(s(Y)))).
   ```