Question 1

- William frequents every bar that serves at least one of the beers he likes.
  \[ \forall b \left[ \text{William frequents } b \leftarrow \exists d \left( \text{William likes } d \land b \text{ serves } d \right) \right] \]
  or equivalently
  \[ \forall b \forall d \left[ \text{William frequents } b \leftarrow \left( \text{William likes } d \land b \text{ serves } d \right) \right] \]

Why equivalently? Because \( \forall x \left( p(x) \leftarrow \exists y q(x, y) \right) \) is logically equivalent to \( \forall x \forall y \left( p(x) \leftarrow q(x, y) \right) \).

- Harry frequents any bar that does not serve a beer he does not like.
  \[ \forall b \left[ \text{Harry frequents } b \leftarrow \neg \exists d \left( b \text{ serves } d \land \neg \text{Harry likes } d \right) \right] \]
  which is logically equivalent to
  \[ \forall b \left[ \text{Harry frequents } b \leftarrow \forall d \left( b \text{ serves } d \rightarrow \text{Harry likes } d \right) \right] \]

- Charles frequents every bar that serves all of the beers he likes.
  \[ \forall b \left[ \text{Charles frequents } b \leftarrow \forall d \left( \text{Charles likes } d \rightarrow b \text{ serves } d \right) \right] \]
  which is logically equivalent to
  \[ \forall b \left[ \text{Charles frequents } b \leftarrow \neg \exists d \left( \text{Charles likes } d \land \neg b \text{ serves } d \right) \right] \]

- Camilla frequents every bar that Charles frequents, and also any bar that serves Young's Special Bitter.
  \[ \forall b \left[ \text{Camilla frequents } b \leftarrow \text{Charles frequents } b \land \forall b \left( \text{Camilla frequents } b \leftarrow b \text{ serves } \text{Young's} \right) \right] \]
  which is logically equivalent to
  \[ \forall b \left( \text{Camilla frequents } b \leftarrow \left( \text{Charles frequents } b \lor b \text{ serves } \text{Young's} \right) \right) \]

Question 2

You can declare the infix operators in Prolog like this:

\[ :- \text{op(550, xfx, [likes, serves, frequents])}. \]

Then:

- \[ \forall b \forall d \left[ \text{William frequents } b \leftarrow \left( \text{William likes } d \land b \text{ serves } d \right) \right] \]
  \[ '\text{William}' \text{ frequents } B :- \]
  \[ '\text{William}' \text{ likes } D, B \text{ serves } D. \]

- \[ \forall b \left[ \text{Harry frequents } b \leftarrow \neg \exists d \left( b \text{ serves } d \land \neg \text{Harry likes } d \right) \right] \]
  \[ '\text{Harry}' \text{ frequents } B :- \]
  \[ \+ (B \text{ serves } D, \+ '\text{Harry}' \text{ likes } D). \]

But:

1. In what sense is Prolog’s negation-by-failure \( \+ \) a correct representation of truth-functional negation \( \neg \)? We are coming to that.
2. Prolog can’t be used to generate bindings from negation-by-failure calls. So we need something like:
   \[ '\text{Harry}' \text{ frequents } B :- \]
   \[ B \text{ serves } _, \% \text{ generates the name of a bar } B \]
   \[ \+ (B \text{ serves } D, \+ '\text{Harry}' \text{ likes } D). \]

3. Some Prologs (and other logic programming/deductive database implementations) don’t allow nested negation-as-failure. So then we have to write something like this:
   \[ '\text{Harry}' \text{ frequents } B :- \]
   \[ B \text{ serves } _, \% \text{ generates the name of a bar } B \]
   \[ \+ \text{bad_for_Harry}(B). \]
   \[ \text{bad_for_Harry}(B) :- \]
   \[ B \text{ serves } D, \+ '\text{Harry}' \text{ likes } D. \]

- \[ \forall b \left[ \text{Charles frequents } b \leftarrow \forall d \left( \text{Charles likes } d \rightarrow b \text{ serves } d \right) \right] \]
  \[ '\text{Charles}' \text{ frequents } B :- \]
  \[ '\text{Charles}' \text{ likes } D, B \text{ serves } D. \]

- \[ \forall b \left( \text{Camilla frequents } b \leftarrow \left( \text{Charles frequents } b \lor b \text{ serves } \text{Young's} \right) \right) \]
  \[ '\text{Camilla}' \text{ frequents } B :- '\text{Charles}' \text{ frequents } B. \]
  \[ '\text{Camilla}' \text{ frequents } B :- B \text{ serves } '\text{Youngs}'. \]
  \% alternatively (equivalently)
  \% 'Camilla' frequents B :-
  \% \% ('Charles' frequents B ; B serves 'Youngs').