Here are some things that caught my eye when looking through last week’s submissions. This list is not necessarily complete; most likely it is not. But hopefully it will help somewhat.

- In question 1, notice that both arguments to function `append` are lists. (Don’t be confused by my poor choice of the letter `m` for one of the list variables.)

- For question 2: `→` is an “arbitrary” relation in the sense that there are inference rules for it, but you don’t know what they are. For the proof it is not necessary to know the rules for `→`, all you need is the two rules for `→*` and the knowledge that those are the only rules for `→*`.

- When doing a proof by induction on the derivation of some judgment `j`, be careful that you get your steps in the right order: First you do a case-split by considering, in turn, every rule that could have been the last rule used in the derivation for `j`. In each case you then determine what special properties must hold for `j`. Also, by inverting the particular rule in question, you find what premises needed to be derived in order to derive `j`. If such a premise is an instance of `j`, then you are ready to use the induction hypothesis at that point.

In any case, my point is: You do the case-split first (determining which rule you consider), then you invert that rule. I have seen at least one attempted solution that somehow first inverted a rule (without justifying why that rule and not some other rule), and which then proceeded to do a case-split (i.e., one which at that point was no longer needed).

- When giving inference rules for the syntactic structure of a language, in order to say that `e` is an expression we usually do not want to derive `e` itself but rather some judgment of the form `e exp`.

- When giving an inductive definition of a relation `R`, don’t forget to write the phrase “`R` is the smallest set such that…”

- When giving an evaluation semantics for a language, it is common that most—if not all—rules have a conclusion of the form `e ⊩ n` where `e` itself is a template for one of the possible syntactic forms. Example: Let the BNF-style rules for some small language be:

\[
\begin{align*}
  n & \in \mathbb{Z} \\
  e & ::= \quad n \mid \text{plus}(e,e) \mid \text{times}(e,e)
\end{align*}
\]
Then there ought to be 3 rules in the evaluation semantics: one for $n$, one for $\text{plus}(e_1, e_2)$, and one for $\text{times}(e_1, e_2)$. For example, it could look like this:

\[
\begin{align*}
\text{CONSTANT} & \quad n \downarrow n \\
\text{PLUS} & \quad e_1 \downarrow n_1 \quad e_2 \downarrow n_2 \quad n = n_1 + n_2 \\
\text{TIMES} & \quad e_1 \downarrow n_1 \quad e_2 \downarrow n_2 \quad n = n_1 \times n_2
\end{align*}
\]