1. [10 points] Consider the rules:

\[
\begin{align*}
\text{zero nat} & \quad (P_{\text{nat}}^{\text{zero}}) \\
n \text{nat} & \quad (P_{\text{nat}}^{\text{succ}}) \\
n \text{nat} & \quad (P_{\text{tree}}^{\text{leaf}}) \\
l \text{tree} & \quad (P_{\text{tree}}^{\text{node}}) \\
\end{align*}
\]

These rules inductively define a set of terms nat representing natural numbers and a set of terms tree representing binary trees with natural numbers at the leaves.

We can inductively (i.e. recursively) define the following reflect function on trees:

\[
\begin{align*}
\text{reflect}(\text{leaf}(n)) & = \text{leaf}(n) \\
\text{reflect}(\text{node}(l,r)) & = \text{node}(\text{reflect}(r), \text{reflect}(l))
\end{align*}
\]

Give rules for an inductive definition of \text{reflect} as a binary relation, and then prove that this relation is single valued (i.e. that the binary relation is a function).

2. [10 points] Consider the following rules defining a binary relation \text{less} on nats:

\[
\begin{align*}
n \text{nat} & \quad (P_{\text{less}}^{\text{zero}}) \\
n \text{nat} \text{ succ}(n) & \quad (P_{\text{less}}^{\text{succ}}) \\
n \text{nat} \text{ succ}(n) & \quad (P_{\text{less}}^{\text{incr}})
\end{align*}
\]

Now consider the third rule

\[
\begin{align*}
n \text{nat} & \quad (P_{\text{less}}^{\text{incr}})
\end{align*}
\]

Is this third rule (a) a derived rule, (b) an admissible rule, or (c) neither. If it is a derived rule, give a derivation. If it is an admissible rule, give an informal proof of that fact. If it is neither, explain why.

Are the original two rules (5) and (6) sufficient to define the usual “less-than” ordering on natural numbers? If so, give a derivation for \text{succ}^3(\text{zero}) \text{ succ}^5(\text{zero}) \text{ less}, (i.e. 3 < 5). If not, add a rule (or rules) that will make \text{less} agree with the usual ordering.

3. [10 points] Do exercise 1 of Section 2.2 (page 14).
4. [20 points] The lambda calculus is a small language defined as follows:

- There is a countable set of variable symbols \( \{x, y, z, \ldots\} \).
- Each variable \( v \) is a lambda term.
- If \( M_1 \) and \( M_2 \) are lambda terms, then \( M_1 M_2 \) is a lambda term (representing the application of \( M_1 \) to \( M_2 \)).
- If \( v \) is a variable symbol, and \( M \) is a lambda term, then \( \lambda v.M \) is a lambda term (representing a function, or lambda abstraction with formal argument \( v \) and body \( M \)).
- Nothing else is a lambda term.

Some examples of lambda terms are: \( x \); \( y (\lambda z.x) \); \( \lambda y.(xy) \); and \( (\lambda y.x)y \). Note that application is denoted by the juxtaposition of the function subterm and the argument subterm. Application associates to the left, so \( xyz \) is interpreted as \( (xy)z \), and the scope of a lambda abstraction extends as far the right as possible, so \( \lambda x.xy \) is interpreted as \( \lambda x.(xy) \), not as \( (\lambda x.x)y \).

(a) Give a context-free grammar for lambda terms (extra credit if it is unambiguous).
(b) Give a first-order abstract syntax for lambda terms.
(c) Give a rule set that inductively defines the set of lambda terms.
(d) Give an inductive definition of a function \( \text{nlambdas} \) on the abstract syntax of lambda terms that counts the number of lambda abstractions. For example,

\[
\begin{align*}
\text{nlambdas}((\lambda x.x)(\lambda y.z)) &= 2 \\
\text{nlambdas}((\lambda x.\lambda z.x)(\lambda y.z)) &= 3 \\
\text{nlambdas}(\lambda x.\lambda z.x(\lambda y.z)) &= 3
\end{align*}
\]