Exercise 1 (20 points) Let us define List(T) as ∀X.X → (T → X → X) → X. Then the two list constructors Nil and Cons can be defined as follows.

\[\begin{align*}
\text{Nil} & \equiv \lambda X.\lambda \text{nil} : X.\lambda \text{cons} : T \rightarrow X \rightarrow X.\text{nil} \\
\text{Cons}(x : T)(xs : \text{List}(T)) & \equiv \lambda X.\lambda \text{nil} : X.\lambda \text{cons} : T \rightarrow X \rightarrow X.\text{cons}(x)(xs[X](\text{nil})(\text{cons}))
\end{align*}\]

Given the above list encoding, please implement the append and reverse functions on lists in System F. Your implementation needs to be coded in ATS. Note that you may not construct recursive functions in your implementation.

Exercise 2 (20 points + 30 extra points) The following datatype \texttt{gtree} and

datatype \texttt{gtree (a:t@ype)} =
\begin{itemize}
    \item \texttt{E (a) of ()}
    \item \texttt{B (a) of (a -><cloref1> gtree a)}
\end{itemize}

The type constructor \texttt{gtree} can be used to form general tree types. For instance, \texttt{gtree(bool)} can be considered as a type for binary trees; \texttt{B (lam x => E)} represents a binary tree \(t_1\) whose left and right subtrees are empty; \texttt{B (lam x => if x then E else t1)} represents a binary tree \(t_2\) whose left subtree is empty and right subtree is \(t_1\).

- (10 points) Please construct a value of the type \texttt{gtree(int)} in ATS such that this value represents an infinite tree satisfying the following property:
  - For each \(n \geq 0\), every node at level \(n\) has \(n + 1\) nonempty children.
  For instance, the root is at level 0, and it has one nonempty child; this child is at level 1, and it has two nonempty children, which are at level 2; etc.

- (10 points) Please encode \texttt{gtree} and its associated constructors \texttt{E} and \texttt{B} in System F.

- (30 points) Please implement the following function \texttt{leftGtree} in System F. Your implementation needs to be written in ATS.

\begin{verbatim}
fun leftGtree (t: gtree bool) =
  case+ t of
  | B (ft) => let
    val ft = ft: bool -*<cloref1> gtree bool in ft (true)
  end // end of [B]
  | E () => E ()
\end{verbatim}
Note that you may not construct recursive functions in your implementation of \texttt{leftGtree}.

**Exercise 3** (20 points) We can define a type erasure function \(|\cdot|\) as follows, which translates a term in System F into an untyped \(\lambda\)-term.

\[
\begin{align*}
|x| &= x & |\lambda x : T . t| &= \lambda x . |t| & |t_1 (t_2)| &= |t_1| (|t_2|) & |\Lambda X . t| &= |t| & |t[T]| &= |t|
\end{align*}
\]

Notice that for a value \(v\) in System F, \(|v|\) may not necessarily be a value. Therefore, given a term \(t\) in System F, \(|t| \rightarrow^* v_0\) does not necessarily imply that we have \(t \rightarrow^* v\) for some value \(v\) such that \(|v| = v_0\). To have this property, we can impose a restriction on \((\mathbf{T-Tabs})\) by requiring that \(|t|\) be a value whenever the following rule is applied.

\[
\frac{\overline{X} ; X ; \Gamma \vdash t : T \quad \overline{X} \vdash \Gamma [ctx]}{\overline{X} ; \Gamma \vdash \Lambda X . t : \forall X . T} \quad (\mathbf{T-Tabs})
\]

This restriction is often called value restriction.

Assume \(\mathcal{D} :: \emptyset ; \emptyset \vdash t : T\) is derivable in System F with value restriction and \(|t| \rightarrow u\). Show that \(t \rightarrow^* t'\) holds for some term \(t'\) such that \(|t'| = u|\).

**Exercise 4** (20 points) The following is a typical implementation of the list reverse function written in ATS:

\[
\text{fun}\{a:type\} \text{ revapp } \{m,n:nat\} .<m>.
\quad (xs: list (a, m), ys: list (a, n)) :<> list (a, m+n) =
\quad \text{case+} \; xs \; \text{of}
\quad \quad | \text{list_cons} (x, xs) \Rightarrow \text{revapp} (xs, \text{list_cons} (x, ys))
\quad \quad | \text{list_nil} () \Rightarrow ys
\quad \text{end of [revapp]}
\]

\[
\text{fn}\{a:type\} \text{ reverse } \{n:nat\}
\quad (xs: list (a, n)) :<> list (a, n) = \text{revapp} (xs, \text{list_nil} ())
\quad \text{end of [reverse]}
\]

Please give a paper/pencil proof based on this implementation that the reverse of the reverse of a given list equals the list itself.

**Exercise 5** (30 points) Please construct a proof in ATS showing that the reverse of the reverse of a given list equals the list itself. See the file \texttt{list.dats} for more details.