



Solution 4

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

$$\begin{aligned}
 Nil &= \Lambda X.\lambda nil : X.\lambda cons : T \rightarrow X \rightarrow X.nil \\
 Cons(x : T)(xs : List(T)) &= \Lambda X.\lambda nil : X.\lambda cons : T \rightarrow X \rightarrow X.cons(x)(xs[X])(nil)(cons)
 \end{aligned}$$

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.

Please see the file `exercise1.dats`.

Exercise 2 (20 points + 30 extra points) The following datatype `gtree` and

```
datatype gtree (a:t@type) =
  | E (a) of () | B (a) of (a -<cloref1> gtree a)
```

The type constructor `gtree` can be used to form general tree types. For instance, `gtree(bool)` can be considered as a type for binary trees; `B (lam x => E)` represents a binary tree t_1 whose left and right subtrees are empty; `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 `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 `gtree` and its associated constructors `E` and `B` in System F.
- (30 points) Please implement the following function `leftGtree` in System F. Your implementation needs to be written in ATS.

```
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 ()
```

Note that you may not construct recursive functions in your implementation of `leftGtree`.

Please see the file `gtree.dats`.

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

$$|x| = x \quad |\lambda x : T.t| = \lambda x.|t| \quad |t_1(t_2)| = |t_1|(|t_2|) \quad |\Lambda X.t| = |t| \quad |t[T]| = |t|$$

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 **(T-Tabs)** by requiring that $|t|$ be a value whenever the following rule is applied.

$$\frac{\vec{X}, X; \Gamma \vdash t : T \quad \vec{X} \vdash \Gamma [ctx]}{\vec{X}; \Gamma \vdash \Lambda X.t : \forall X.T} \text{ (T-Tabs)}$$

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 $|t'| = u$.

Proof (sketch) We first list as follows some key facts that are to be used in the following proof:

1. Note that $[[x \mapsto t]t_0]$ equals $[x \mapsto |t|]|t_0|$ for any t, x and t_0 . This can be proven by structural induction on t .
2. If t is well-typed and $|t|$ is a value, then $t \rightarrow^* t'$ holds for some value t' such that $|t| = |t'|$. This can be proven by structural induction on t .

We now proceed by structural induction on \mathcal{D} . The most interesting case is the one where $t = t_1(t_2)$ for some terms t_1 and t_2 . Let \mathcal{D}_1 and \mathcal{D}_2 be the two subderivations in \mathcal{D} for assigning types to t_1 and t_2 , respectively. Clearly, we have $|t| = |t_1|(|t_2|)$. We have the following cases:

- $|t_1|$ is not a value. Then we have $u = u_1(|t_2|)$ for some term u_1 such that $|t_1| \rightarrow u_1$ holds. By induction hypothesis on \mathcal{D}_1 , we have $t_1 \rightarrow^* t'_1$ holds for some t'_1 such that $|t'_1| = u_1$. So we have $t \rightarrow^* t'$ for $t' = t'_1(t_2)$. Note that $|t'| = u$, and we are done.
- $|t_1|$ is a value but $|t_2|$ is not a value. By (2), we have $t_1 \rightarrow^* t'_1$ for some value t'_1 such that $|t_1| = |t'_1|$. Note that $u = |t_1|(u_2)$ for some u_2 such that $|t_2| \rightarrow u_2$ holds. By induction hypothesis on \mathcal{D}_2 , we have $t_2 \rightarrow^* t'_2$ holds for some t'_2 such that $|t'_2| = u_2$. So we have $t \rightarrow^* t'_1(t_2) \rightarrow^* t'_1(t'_2)$. Note that $|t'_1(t'_2)| = |t_1|(u_2) = |u|$, and we are done.
- $|t_1|$ and $|t_2|$ are values. By (2), we have a value t'_1 such that $|t_1| = |t'_1|$ and $t_1 \rightarrow^* t'_1$. Also, we have a value t'_2 such that $|t_2| = |t'_2|$ and $t_2 \rightarrow^* t'_2$. By the Lemma of Canonical Forms, we know that t'_1 is of the form $\lambda x.t'_{10}$. So we have $t \rightarrow^* t'_1(t_2) \rightarrow^* t'_1(t'_2) \rightarrow [x \mapsto t'_2]t'_{10}$. Also, we have $|t| = |t_1|(|t_2|) = (\lambda x.|t'_{10}|)(|t'_2|) \rightarrow [x \mapsto t'_2]|t'_{10}|$. By (1), we have $[[x \mapsto t'_2]t'_{10}] = [x \mapsto t'_2]|t'_{10}|$, and we are done.

All of the other cases are less complicated and thus omitted.

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

```
fun{a:t@ype} revapp {m,n:nat} .<m>.
  (xs: list (a, m), ys: list (a, n)):<> list (a, m+n) =
  case+ xs of
  | list_cons (x, xs) => revapp (xs, list_cons (x, ys))
  | list_nil () => ys
// end of [revapp]

fn{a:t@ype} reverse {n:nat}
  (xs: list (a, n)):<> list (a, n) = revapp (xs, list_nil ())
// 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 `list.dats` for more details.*

Please see the file `list.dats`.