Software verification - Typing

Assignments for week 4

1. Give a type derivation for the following expression:

   $$\textbf{let}\ p = \lambda x.\lambda y.\textbf{if}\ (x == 0)\ 1\ (y * (p\ (x-1)\ y))\ \textbf{in}\ p\ 7$$

   Can you see what this expression stands for?

2. In this exercise you will extend *ML* with lists. Lists are built up from the constructors **Nil** (the empty list) and **Cons** (constructing a list from a head element and a list). For example a list containing the numbers $1, 2, 3$ is defined as $\textbf{Cons}\ 1\ (\textbf{Cons}\ 2\ (\textbf{Cons}\ 3\ \textbf{Nil}))$. To manipulate lists, we need a way to check whether a list is empty or not. In the latter case, we also should be able to obtain the head and the tail of a list.

   (a) Extend the syntax of *ML* with these (five) new list constructs.

   (b) Define the *natural operational semantics* of the new constructs. Use the format that was explained on slide 12 of the lecture slides.

   (c) Define a function *append* for concatenating two lists.

   (d) The type of a list is denoted by $\textbf{List}(\sigma)$, where σ is the type of the list elements. Give the typing rules for the list constructs.

   (e) Give a type derivation for the expression you defined in 2c.

   This is a mandatory assignment that should be handed in in the post box at floor 1 of Mercator 1, behind the couches.

3. Consider the following expression.

   $$\lambda x.\textbf{let}\ f = x\ \textbf{in}\ (f, f)$$

   Suppose that we would omit the additional check $\alpha_1, \ldots, \alpha_n \notin \mathrm{FV}(\Gamma)$ in the polymorphic let rule (see lecture slide 26). Show that in that case the following type (which is obviously wrong) is derivable for the given expression.

   $$\alpha \to (\textbf{Int}, \textbf{Bool})$$