Problem Sheet 4

Alex Kavvos

The following questions are about the simply-typed λ-calculus (STLC).

1. Draw derivations that evidence the following typing judgements.
   (i) $x : \mathsf{Str} + (\mathsf{Str} \times \mathsf{Num}) \vdash \mathsf{case}(x; y.\, y; z.\, \pi_1(z)) : \mathsf{Str}$
   (ii) $\vdash \lambda x : \mathsf{Str} + \mathsf{Num}.\, \mathsf{case}(x; y.\, \mathsf{inr}(y); z.\, \mathsf{inl}(z)) : \mathsf{Str} + \mathsf{Num} \to \mathsf{Num} + \mathsf{Str}$
   (iii) $f : \mathsf{Num} \times \mathsf{Str} \to \mathsf{Num}, x : \mathsf{Str} \vdash f(\langle \mathsf{num}[0], x\rangle) : \mathsf{Num}$
2. Write down transition sequences that reduce the following terms to values.
   (i) $\mathsf{case}(\mathsf{inr}(\langle \mathsf{str}[\text{‘hi’}], \mathsf{num}[0]\rangle); y.\, y; z.\, \pi_1(z))$
   (ii) $(\lambda x : \mathsf{Str} + \mathsf{Num}.\, \mathsf{case}(x; y.\, \mathsf{inr}(y); z.\, \mathsf{inl}(z)))(\mathsf{inl}(\mathsf{num}[0]))$
   (iii) $(\lambda z.\, \pi_1(z))(\langle \mathsf{num}[0], \mathsf{str}[\text{‘hi’}]\rangle)$
3. This question is about modelling the following Haskell data type in the simply-typed λ-calculus.

   ```
   data MaybeString = Nothing | Just String
   ```

   Intuitively, we expect this data type $\mathsf{MaybeStr}$ to have the following typing rules.

   $$\textsc{Nothing}\quad \frac{}{\Gamma \vdash \mathsf{Nothing} : \mathsf{MaybeStr}} \qquad\qquad \textsc{Just}\quad \frac{\Gamma \vdash e : \mathsf{Str}}{\Gamma \vdash \mathsf{Just}(e) : \mathsf{MaybeStr}}$$

   $$\textsc{Match}\quad \frac{\Gamma \vdash e : \mathsf{MaybeStr} \qquad \Gamma \vdash e_n : \tau \qquad \Gamma, x : \mathsf{Str} \vdash e_j : \tau}{\Gamma \vdash \mathsf{match}(e; e_n; x.\, e_j) : \tau}$$

   The first term represents `Nothing`, and the second term that represents `Just e`, where `e :: String`.

   The third term performs **pattern matching**. It first examines e: if that is a $\mathsf{Nothing}$ it returns e_n; if it is a $\mathsf{Just}(e)$ with $e : \mathsf{Str}$, it substitutes e for x in e_j. Thus $\mathsf{match}(-; e_n; x.\, e_j)$ corresponds to the definition

   ```
   f Nothing = e_n
   f (Just x) = e_j -- this clause can use the variable x :: String
   ```

   (i) Write down a representation of this type in the STLC. [Hint: use **1**.]
   (ii) Show that the three rules Nothing, Just and Match above are **definable**. That is, show the terms $\mathsf{Nothing}$, $\mathsf{Just}(e)$ and $\mathsf{match}(e; e_n; x.\, e_j)$ can be expanded into some term of the STLC, which is such that the typing rules are **derivable** if we assume that weakening is a typing rule of the system.
4. ($*$) Prove progress and preservation for the **constants-and-functions fragment** of the STLC.

   [Hint: The constants-and-function fragment of the STLC is an extension to the language of numbers and strings: we reached it by *adding* the rules for function types. Thus, to establish these theorems **you only need to show them for the new rules**, as last week's proofs cover the rest!

   Do this in steps. First extend the inversion, substitution, and canonical form lemmas to function types. You will need weakening; you may assume it, but you can also prove it if you feel like it. Then, prove preservation and progress. Pretend there are no products or sums throughout.]