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) $(\lambda z.\, \pi_1(z))(\langle \mathsf{num}[0], \mathsf{str}[\text{‘hi’}]\rangle)$

Solution:

(i)

$$\begin{aligned}\mathsf{case}(\mathsf{inr}(\langle \mathsf{str}[\text{‘hi’}], \mathsf{num}[0]\rangle); y.\, y; z.\, \pi_1(z)) &\longmapsto \pi_1(\langle \mathsf{str}[\text{‘hi’}], \mathsf{num}[0]\rangle) \\ &\longmapsto \mathsf{str}[\text{‘hi’}]\end{aligned}$$

(ii)

$$\begin{aligned}(\lambda x : \mathsf{Str} + \mathsf{Num}.\, \mathsf{case}(x; y.\, \mathsf{inr}(y); z.\, \mathsf{inl}(z)))(\mathsf{inl}(\mathsf{num}[0])) &\longmapsto \mathsf{case}(\mathsf{inl}(\mathsf{num}[0]); y.\, \mathsf{inr}(y); z.\, \mathsf{inl}(z)) \\ &\longmapsto \mathsf{inr}(\mathsf{num}[0])\end{aligned}$$

(iii) $(\lambda z.\, \pi_1(z))(\langle \mathsf{num}[0], \mathsf{str}[\text{‘hi’}]\rangle) \longmapsto \pi_1(\langle \mathsf{num}[0], \mathsf{str}[\text{‘hi’}]\rangle) \longmapsto \mathsf{num}[0]$

3. This question is about modelling the following Haskell data type in the simply-typed λ-calculus.

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

Intuitively, we expect this data type MaybeStr to have the following typing rules.

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

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

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.

Solution:

(i) $\mathbf{1} + \mathsf{Str}$

(ii) Let

$$\begin{aligned}\mathsf{Nothing} &\stackrel{\text{def}}{=} \mathsf{inl}(\langle\rangle)\\ \mathsf{Just}(x) &\stackrel{\text{def}}{=} \mathsf{inr}(x)\\ \mathsf{match}(e; e_n; x.\, e_s) &\stackrel{\text{def}}{=} \mathsf{case}(e; \mathsf{inl}(y).\, e_n; \mathsf{inr}(x).\, e_s)\end{aligned}$$

Here is the proof that the typing rules given above are derivable.

$$\dfrac{\dfrac{}{\Gamma \vdash \langle\rangle : \mathbf{1}}\;\textsc{Unit}}{\Gamma \vdash \mathsf{Nothing} \stackrel{\text{def}}{=} \mathsf{inl}(\langle\rangle) : \mathbf{1} + \mathsf{Str}}\;\textsc{Inl} \qquad \dfrac{\dfrac{\vdots}{\Gamma \vdash x : \mathsf{Str}}\;\textsc{Unit}}{\Gamma \vdash \mathsf{Just}(x) \stackrel{\text{def}}{=} \mathsf{inl}(x) : \mathbf{1} + \mathsf{Str}}\;\textsc{Inr}$$

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash e : \mathbf{1} + \mathsf{Str}} \qquad \dfrac{\dfrac{\vdots}{\Gamma \vdash e_n : \tau}}{\Gamma, y : \mathbf{1} \vdash e_n : \tau}\;\textsc{Wk} \qquad \dfrac{\vdots}{\Gamma, x : \mathsf{Str} \vdash e_s : \tau}}{\Gamma \vdash \mathsf{match}(e; e_n; x.\, e_s) \stackrel{\text{def}}{=} \mathsf{case}(e; y.\, e_n; x.\, e_s) : \tau}\;\textsc{Case}$$

The rule Wk corresponds to the assumption that weakening is a primitive rule of the system. Of course, weakening is not a rule of the STLC, but it is admissible: given a derivation of $\Gamma \vdash e_n : \tau$ we can construct a derivation of $\Gamma, y : \mathbf{1} \vdash e_n : \tau$ (this requires inductive proof).

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.]

Solution:

Claim 1 (Inversion). Suppose $\Gamma \vdash e : \tau$.

- If $e = \lambda x : \sigma.\, e'$ then $\tau = \sigma \to \tau'$ for some type τ', and $\Gamma, x : \sigma \vdash e_2 : \tau'$.
- If $e = e_1(e_2)$ then there exists a type τ_2 such that $\Gamma \vdash e_1 : \tau_2 \to \tau$ and $\Gamma \vdash e_2 : \tau_2$.

The proof is by inspection.

Claim 2 (Substitution). If $\Gamma \vdash e : \tau$ and $\Gamma, x : \tau \vdash u : \sigma$ then $\Gamma \vdash u[e/x] : \sigma$.

Proof. By induction on the derivation of $\Gamma, x : \tau \vdash u : \sigma$.

Case(Lam). Suppose the derivation is of the form

$$\dfrac{\dfrac{\vdots}{\Gamma, x : \tau, y : \sigma_1 \vdash u : \sigma_2}}{\Gamma, x : \tau \vdash \lambda y : \sigma_1 .\, u : \sigma_1 \to \sigma_2}\ \textsc{Lam}$$

By **weakening** we have $\Gamma, y : \sigma_1 \vdash e : \tau$. Then, applying the **IH** to that and the smaller derivation $\Gamma, x : \tau, y : \sigma_1 \vdash u : \sigma_2$ we obtain a derivation of $\Gamma, y : \sigma_1 \vdash u[e/x] : \sigma_2$. We can then construct

$$\dfrac{\dfrac{\vdots}{\Gamma, y : \sigma_1 \vdash u[e/x] : \sigma_2}}{\Gamma \vdash \lambda y : \sigma_1 .\, u[e/x] : \sigma_1 \to \sigma_2}\ \textsc{Lam}$$

But we have that $(\lambda y : \sigma_1 .\, u)[e/x] \stackrel{\text{def}}{=} \lambda y : \sigma_1 .\, u[e/x]$, so we have the result.

Case(App). Suppose the derivation is of the form

$$\dfrac{\dfrac{\vdots}{\Gamma, x : \tau \vdash u_1 : \tau_2 \to \sigma} \qquad \dfrac{\vdots}{\Gamma, x : \tau \vdash u_2 : \tau_2}}{\Gamma, x : \tau \vdash u_1(u_2) : \sigma}\ \textsc{App}$$

Applying the **IH** to the smaller derivation $\Gamma, x : \tau \vdash u_1 : \tau_2 \to \sigma$ we obtain $\Gamma \vdash u_1[e/x] : \tau_2 \to \sigma$.

Applying the **IH** to the smaller derivation $\Gamma, x : \tau \vdash u_2 : \tau_2$ we obtain $\Gamma \vdash u_2[e/x] : \tau_2$.

We combine these two derivations using the rule App:

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash u_2[e/x] : \tau_2} \quad \dfrac{\vdots}{\Gamma \vdash u_1[e/x] : \tau_2 \to \sigma}}{\Gamma \vdash u_1[e/x](u_2[e/x]) : \sigma}\ \textsc{App}$$

But the subject of this last derivation is by definition $(u_1(u_2))[e/x]$.

□

Claim 3 (Preservation). If $\vdash e : \tau$ and $e \longmapsto e'$ then $\vdash e' : \tau$.

Proof. By induction on $e \longmapsto e'$.

Case(D-Beta). Suppose the reduction is of the form $(\lambda x : \sigma .\, e_1)(e_2) \longmapsto e_1[e_2/x]$.

We know from the assumptions that $\vdash (\lambda x : \sigma .\, e_1)(e_2) : \tau$. By **inversion** this means that there exists τ_2 such that $\vdash \lambda x : \sigma .\, e_1 : \tau_2 \to \tau$ and $\vdash e_2 : \tau_2$.

Again by **inversion** on the judgement $\vdash \lambda x : \sigma .\, e_1 : \tau_2 \to \tau$ we see that $\tau_2 = \sigma$, and it must be that $x : \sigma \vdash e_1 : \tau$.

By **substitution** on the judgements $x : \sigma \vdash e_1 : \tau$ and $\vdash e_2 : \tau_2$ (recall that $\sigma = \tau_2$) we get $\vdash e_1[e_2/x] : \tau$, which is what we wanted to prove.

Case(D-App-1). Suppose that the reduction is the form $e_1(e_2) \longmapsto e_1'(e_2)$ with premise $e_1 \longmapsto e_1'$.

We know from the assumptions that $\vdash e_1(e_2) : \tau$. By **inversion** it must be that there exists τ_2 such that $\vdash e_1 : \tau_2 \to \tau$ and $\vdash e_2 : \tau_2$.

By the **IH** applied to $\vdash e_1 : \tau_2 \to \tau$ and $e_1 \longmapsto e_1'$ we get that $\vdash e_1' : \tau_2 \to \tau$ as well. Thus, we can construct a typing derivation

$$\dfrac{\dfrac{\vdots}{\vdash e_1' : \tau_2 \to \tau} \qquad \dfrac{\vdots}{\vdash e_2 : \tau_2}}{\vdash e_1'(e_2) : \tau}\ \textsc{App}$$

$\square$

Claim 4 (Canonical Forms). If $\vdash e : \sigma \to \tau$ and e val, then $e = \lambda x : \sigma.\, u$ with $x : \sigma \vdash u : \tau$.

The proof of the canonical forms lemma is by inspection. It is a necessary lemma for proving

Claim 5 (Progress). If $\vdash e : \tau$ then either e val or $e \longmapsto e'$ for some e'.

Proof. By induction on $\vdash e : \tau$.

Case(Lam). If the typing derivation ends with $\vdash \lambda x : \sigma.\, e : \tau$ then by the rule Val-Lam we immediately know that $\lambda x : \sigma.\, e$ val, so we have the result.

Case(App). Suppose the typing derivation is of the form

$$\dfrac{\dfrac{\vdots}{\vdash e_1 : \tau_2 \to \tau} \qquad \dfrac{\vdots}{\vdash e_2 : \tau_2}}{\vdash e_1(e_2) : \tau}\ \textsc{App}$$

for some τ_2. Then we apply the **IH** to $\vdash e_1 : \tau_2 \to \tau$, which gives two cases.

- If e_1 val then we know that $\vdash e_1 : \tau_2 \to \tau$ so by the **canonical forms** lemma we see that $e_1 = \lambda x : \tau_2.\, u$ for some u. Hence we can perform the reduction $(\lambda x : \tau_2.\, u)(e_2) \longmapsto u[e_2/x]$ by the rule D-Beta.
- If there exists e_1' with $e_1 \longmapsto e_1'$ then by the rule D-App-1 we can perform the reduction $e_1(e_2) \longmapsto e_1'(e_2)$.

In either case there is an available reduction we can perform on the term $e_1(e_2)$.

$\square$