Problem Sheet 5

Alex Kavvos

The following questions are about PCF.

1. Draw derivations that evidence the following typing judgements.

   (i) $\vdash \mathsf{fix}(n : \mathsf{Nat}.\, \mathsf{succ}(n)) : \mathsf{Nat}$

   (ii) $\vdash \mathsf{plus} : \mathsf{Nat} \rightharpoonup \mathsf{Nat} \rightharpoonup \mathsf{Nat}$ (as in Lecture 10)

Solution: Here is (i):

$$\dfrac{\dfrac{\dfrac{}{n : \mathsf{Nat} \vdash n : \mathsf{Nat}}}{n : \mathsf{Nat} \vdash \mathsf{succ}(n) : \mathsf{Nat}}\ \textsc{Succ}}{\vdash \mathsf{fix}(n : \mathsf{Nat}.\, \mathsf{succ}(n)) : \mathsf{Nat}}\ \textsc{Fix}$$

(ii) is similar.

2. Will the following terms eventually reduce to values? If so, write down transition sequences.

   You may find it useful to introduce some shorthands so you do not end up writing the same thing over and over. E.g. you can define $u \stackrel{\text{def}}{=} \mathsf{fix}(x : \tau.\, e)$, and then abbreviate the conclusion of D-Fix to the much more economical reduction $u \longmapsto e[u/x]$.

   (i) $\mathsf{plus}(\mathsf{succ}(\mathsf{zero}))(\mathsf{succ}(\mathsf{zero}))$ (as in Lecture 10)

   (ii) $\mathsf{fix}(x : \mathsf{Nat}.\, x)$

Solution:

(i) Write $\mathsf{one} \stackrel{\text{def}}{=} \mathsf{succ}(\mathsf{zero})$, and skip types in lambdas.

$$\begin{aligned}
\mathsf{plus}(\mathsf{one})(\mathsf{one}) &\longmapsto (\lambda n.\, \lambda m.\, \mathsf{ifz}(m; n; x.\, \mathsf{succ}(\mathsf{plus}(n)(x))))(\mathsf{one})(\mathsf{one}) && \text{by D-Fix} \\
&\longmapsto (\lambda m.\, \mathsf{ifz}(m; \mathsf{one}; x.\, \mathsf{succ}(\mathsf{plus}(\mathsf{one})(x))))(\mathsf{one}) && \text{by D-Beta} \\
&\longmapsto \mathsf{ifz}(\mathsf{one}; \mathsf{one}; x.\, \mathsf{succ}(\mathsf{plus}(\mathsf{one})(x))) && \text{by D-Beta} \\
&\longmapsto \mathsf{succ}(\mathsf{plus}(\mathsf{one})(\mathsf{zero})) && \text{by D-Ifz-Succ} \\
&\longmapsto \mathsf{succ}((\lambda n.\, \lambda m.\, \mathsf{ifz}(m; n; x.\, \mathsf{succ}(\mathsf{plus}(n)(x))))(\mathsf{one})(\mathsf{zero})) && \text{by D-Fix and D-Succ-1} \\
&\longmapsto \mathsf{succ}((\lambda m.\, \mathsf{ifz}(m; \mathsf{one}; x.\, \mathsf{plus}(\mathsf{one})(x)))(\mathsf{zero})) && \text{by D-Beta and D-Succ-1} \\
&\longmapsto \mathsf{succ}(\mathsf{ifz}(\mathsf{zero}; \mathsf{one}; x.\, \mathsf{plus}(\mathsf{one})(x))) && \text{by D-Beta and D-Succ-1} \\
&\longmapsto \mathsf{succ}(\mathsf{one}) && \text{by D-Ifz-Zero and D-Succ-1} \\
&\stackrel{\text{def}}{=} \mathsf{succ}(\mathsf{succ}(\mathsf{zero}))
\end{aligned}$$

(ii) Omitting type annotations we have

$$\mathsf{fix}(x.\, x) \longmapsto x[\mathsf{fix}(x.\, x)/x] \stackrel{\text{def}}{=} \mathsf{fix}(x.\, x) \longmapsto \mathsf{fix}(x.\, x) \longmapsto \dots$$

It is evident that this term will not evaluate to a value.

3. (i) Does the term $\mathsf{fix}(x : \mathsf{Nat}.\, \mathsf{succ}(x))$ reduce to a value?

   (ii) Suppose we change the rule VAL-SUCC to read

$$\frac{}{\mathsf{succ}(e)\ \mathsf{val}}\ \textsc{Val-Succ}$$

   Does it reduce to a value in that case? How is the evaluation of the term in 2(i) affected?

   (iii) Which of the two behaviours more closely approximates that of Haskell and the data type `Nat`?

Solution:

(i) Omitting type annotations we have:

$$\mathsf{fix}(x.\, \mathsf{succ}(x)) \longmapsto \mathsf{succ}(\mathsf{fix}(x.\, \mathsf{succ}(x))) \longmapsto \mathsf{succ}(\mathsf{succ}(\mathsf{fix}(x.\, \mathsf{succ}(x)))) \longmapsto \ldots$$

It is thus evident that this term will continue reproducing itself.

(ii) In this case the term reduces to a value in one step:

$$\mathsf{fix}(x.\, \mathsf{succ}(x)) \longmapsto \mathsf{succ}(\mathsf{fix}(x.\, \mathsf{succ}(x)))$$

(iii) Recall that the data type in question is

```
data Nat = Zero | Succ Nat
```

We are indeed able to define

```
infnat :: Nat
infnat = Succ infnat
```

But if we try to evaluate `infnat` we get the following output.

```
ghci> infnat

<interactive>:3:1: error:
    • No instance for (Show Nat) arising from a use of ‘print’
    • In a stmt of an interactive GHCi command: print it
```

This is of course the error message that `ghci` uses to let us know it does not know how to print the result of the evaluation. Thus, a fair answer to this question would be to say that Haskell behaves like the revised rule: when the interpreter reaches a `Succ` constructor, it does not know what to do with it, and hence halts. This is corroborated by the observation that if there is no top-level `Succ` constructor, i.e. if our definition is e.g.

```
inf :: Nat
inf = inf
```

then Haskell produces no observable behaviour upon evaluation of `inf`.

This behaviour can be overriden by defining an instance of the `Show` class:

```
instance Show Nat where
  show Zero = "Zero"
  show (Succ n) = "Succ(" ++ show n ++ ")
```

The resultant behaviour is similar to the original rule.

Instead, observable printing similar to the revised rule is given by

```
instance Show Nat where
  show Zero = "Zero"
  show (Succ n) = "Succ <something>"
```

whereupon evaluation of `infnat` produces the output `"Succ <something>"`

4. Complete the definition of `times :: Nat -> Nat -> Nat` from Lecture 10. Then translate it to PCF.

   [Hint: use `plus` and plus respectively.]

Solution:

```
times :: Nat -> Nat -> Nat
times n Zero = Zero
times n (Succ Zero) = n
times n (Succ x) = plus (times n x) n
```

This corresponds to the PCF term (omitting type annotations):

$$\vdash \mathsf{times} \stackrel{\text{def}}{=} \mathsf{fix}(f.\, \lambda n.\, \lambda m.\, \mathsf{ifz}(m; \mathsf{zero}; x.\, \mathsf{ifz}(x; n; y.\, \mathsf{plus}(f(n)(x))(n)))) : \mathsf{Nat} \rightharpoonup \mathsf{Nat} \rightharpoonup \mathsf{Nat}$$

(Of course it is equally valid to pattern match on `n`.)

5. This question is about proving progress and preservation for PCF.

   (i) State the substitution lemma for PCF. Prove the case of the fixed point rule.

   (ii) State the inversion lemma for the pre-term $\mathsf{fix}(x : \tau.\, e)$.

   (iii) State the progress and preservation theorems for PCF. Prove the cases for the fixed point rules.

Solution:

(i) The substitution lemma is the same as before: if $\Gamma \vdash e : \sigma$ and $\Gamma, x : \tau \vdash u : \tau$ then $\Gamma \vdash u[e/x] : \tau$.

(ii) If $\Gamma \vdash e : \tau$ and $e = \mathsf{fix}(x : \tau'.\, e)$ then $\tau = \tau'$ and $\Gamma, x : \tau \vdash e : \tau$.

(iii) The statements remain the same as before.

Claim 1 (Preservation)**.** If $\vdash e : \tau$ and $e \longmapsto e'$ then $\vdash e' : \tau$.

Proof. By induction on $e \longmapsto e'$. We only show the case for fixed points.

Case(D-Fix). Then the reduction is $\mathsf{fix}(x : \tau'.\, e) \longmapsto e[\mathsf{fix}(x : \tau'.\, e)/x]$ for some τ' and e. We know by assumption that $\vdash \mathsf{fix}(x : \tau'.\, e) : \tau$. Then by **inversion** we have that $\tau = \tau'$ and $x : \tau \vdash e : \tau$. By **substitution**, if we substitute $\vdash \mathsf{fix}(x : \tau.\, e) : \tau$ into $x : \tau \vdash e : \tau$ we obtain $\vdash e[\mathsf{fix}(x : \tau.\, e)/x] : \tau$, which is what we wanted to prove.

□

Claim 2 (Progress)**.** If $\vdash e : \tau$ then either e val or $e \longmapsto e'$ for some e'.

Proof. By induction on $\vdash e : \tau$. We only show the case for fixed points.

Case(Fix). Suppose the derivation ends with

$$\dfrac{\dfrac{\vdots}{x : \tau \vdash e : \tau}}{\vdash \mathsf{fix}(x : \tau.\, e) : \tau}\ \textsc{Fix}$$

By D-Fix we have $\mathsf{fix}(x : \tau.\, e) \longmapsto e[\mathsf{fix}(x : \tau.\, e)/x]$ so there is always a transition the term can take.

□

6. (∗) This question is about the uniqueness of typing in PCF.
   (i) State the uniqueness of typing for PCF (cf. problem sheet 2).
   (ii) Prove the inductive case for the fixed point rule.
   (iii) Would uniqueness of typing hold if we replaced the fixed point rule with the following one?

$$\dfrac{\Gamma, x : \tau \vdash e : \tau}{\Gamma \vdash \mathsf{fix}(x.\, e) : \tau}\ \textsc{Fix}$$

   If yes, give a proof. If not, give a counterexample.

Solution:

(i) Given a context Γ and a pre-term e there is at most one τ such that $\Gamma \vdash e : \tau$.

(ii) As usual, the proof is by induction on $\Gamma \vdash e : \tau$.

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

$$\dfrac{\dfrac{\vdots}{\Gamma, x : \tau \vdash e : \tau}}{\Gamma \vdash \mathsf{fix}(x : \tau.\, e) : \tau}\ \textsc{Fix}$$

Suppose we also have a derivation of $\Gamma \vdash \mathsf{fix}(x : \tau.\, e) : \tau'$ for some τ'. By **inversion** we must have $\tau = \tau'$, which proves the result.

(iii) No: $\mathsf{fix}(x.\, x)$ can have any type. For example $\vdash \mathsf{fix}(x.\, x) : \mathsf{Nat}$ and $\vdash \mathsf{fix}(x.\, x) : \mathsf{Nat} \rightharpoonup \mathsf{Nat}$.