15-814 Homework 2 Solutions

October 10, 2017

Task 1 *Prove closure under head expansion.*

Solution: We proceed by structural induction on the type τ.

Case $\tau = \texttt{nat}$: We need to show the 3 conditions for $\mathsf{Red}_{\texttt{nat}}(e)$ to hold.

1. $\cdot \vdash e : \texttt{nat}$ holds by assumption.
2. We know $\mathsf{Red}_{\texttt{nat}}(e')$, i.e. there exists v val such that $e' \mapsto^* v$. Therefore, since we assumed $e \mapsto e'$, we also have $e \mapsto^* v$ by transitivity and definition of $\mapsto^*$.
3. This follows directly from the $\mathsf{Red}_{\texttt{nat}}(e')$ assumption, since we are considering the same v.

Case $\tau = \tau_1 \to \tau_2$: We need to show the 3 conditions for $\mathsf{Red}_{\tau_1 \to \tau_2}(e)$ to hold.

1. $\cdot \vdash e : \tau_1 \to \tau_2$ holds by assumption.
2. We know $\mathsf{Red}_{\tau_1 \to \tau_2}(e')$, i.e. there exists v val such that $e' \mapsto^* v$. Therefore, since we assumed $e \mapsto e'$, we also have $e \mapsto^* v$ by definition of $\mapsto^*$.
3. Consider e'' such that $\mathsf{Red}_{\tau_1}(e'')$. We need to show $\mathsf{Red}_{\tau_2}(e\ e'')$. Since τ_2 is structurally smaller than $\tau_1 \to \tau_2$, we do this by showing the 3 premises of the inductive hypothesis.
   (a) By the third condition of $\mathsf{Red}_{\tau_1 \to \tau_2}(e')$, we have $\mathsf{Red}_{\tau_2}(e'\ e'')$.
   (b) We have $\cdot \vdash e : \tau_1 \to \tau_2$ and $\cdot \vdash e'' : \tau_1$ (by assumptions). Therefore, $\cdot \vdash e\ e'' : \tau_2$ by (App).
   (c) Since $e \mapsto e'$, we have $e\ e'' \mapsto e'\ e''$ by (App-S).

   Therefore, $\mathsf{Red}_{\tau_2}(e\ e'')$ by I.H..

Task 2 *Prove the remaining cases of Theorem 3. You may state (without proof) lemmas about substitution, but be sure to check that they are actually true.*

Solution: In the proof, we will use $\circ$ to denote composition of finite maps. For example, if $\gamma = \{x_1 \hookrightarrow e_1, \ldots, x_n \hookrightarrow e_n\}$, $\gamma' = \{x'_1 \hookrightarrow e'_1, \ldots, x'_n \hookrightarrow e'_n\}$ then $\gamma' \circ \gamma = \{x'_1 \hookrightarrow e'_1, \ldots, x'_n \hookrightarrow e'_n, x_1 \hookrightarrow e_1, \ldots, x_n \hookrightarrow e_n\}$. We implicitly assume that the variable names mapped by γ, γ' do not clash. Furthermore, we will use (without proof) the congruence of $\mapsto^*$ with the congruence rules, and the following lemmas[1]:

Lemma 1 (Simultaneous substitution decomposition) *If $\gamma \Vdash \Gamma$, then γ only substitutes closed terms for variables, and in particular, composed finite maps containing γ can be decomposed: $(\gamma' \circ \gamma)(e) = \gamma'(\gamma(e))$*

[1]These were my original solutions, but I think that some of my uses of the substitution lemma were unnecessary, see footnote in (Lam) case. The decomposition lemma is probably too much detail here, but observe that you cannot, in general, conclude $(\gamma' \circ \gamma)e = \gamma'(\gamma(e))$ without further assumptions. For example, $\{x \hookrightarrow y, y \hookrightarrow x\}y = x$ but $\{x \hookrightarrow y\}(\{y \hookrightarrow x\}y) = y$.

Lemma 2 (Substitution) *If $\gamma \Vdash \Gamma$, and $\Gamma, \Gamma' \vdash e : \tau$, then γ only substitutes closed terms for variables, and in particular, $\Gamma' \vdash \gamma(e) : \tau$.*

Proof: The proof proceeds by induction on $\Gamma \vdash e : \tau$ (the case for (App) is omitted). (I write the form of $\Gamma \vdash e : \tau$ followed by the rulename for each relevant case). Note that by definition of Red_τ, whenever $\mathsf{Red}_\tau(e)$, then $\cdot \vdash e : \tau$. This also implies that e is a closed term with no free variables.

Case $\Gamma, x : \tau \vdash x : \tau$ (Hyp): By assumption, we have $\gamma \Vdash \Gamma, x : \tau$, i.e. $\gamma = \{..., x \hookrightarrow e\}$ by definition of $\Vdash$. In particular, we have $\mathsf{Red}_\tau(e)$, but $e = \gamma(x)$ by definition of substitution on variable x so we are done.

Case $\Gamma \vdash \mathtt{z} : \mathtt{nat}$ (Z): We have $\gamma(\mathtt{z}) = \mathtt{z}$, so we show $\mathsf{Red}_{\mathtt{nat}}(\mathtt{z})$.

1. $\cdot \vdash \mathtt{z} : \mathtt{nat}$ by (Z).
2. $\mathtt{z}$ val by (Z-V) and $\mathtt{z} \mapsto^* \mathtt{z}$ by reflexivity.
3. $\mathtt{z} \downarrow$ by ($\downarrow$-Z).

Case $\Gamma \vdash \mathtt{s}(e) : \mathtt{nat}$ (S): We need to show $\mathsf{Red}_{\mathtt{nat}}(\gamma(\mathtt{s}(e)))$, i.e. $\mathsf{Red}_{\mathtt{nat}}(\mathtt{s}(\gamma(e)))$ by definition of substitution.

From the premise of the rule, we have $\Gamma \vdash e : \mathtt{nat}$, and by assumption $\gamma \Vdash \Gamma$. By I.H., we have $\mathsf{Red}_{\mathtt{nat}}(\gamma(e))$. By definition of $\mathsf{Red}_{\mathtt{nat}}$, we have $\cdot \vdash \gamma(e) : \mathtt{nat}$, $\gamma(e) \mapsto^* v$ for some v val and $v \downarrow$.

1. $\cdot \vdash \mathtt{s}(\gamma(e)) : \mathtt{nat}$ by (S).
2. $\mathtt{s}(\gamma(e))$ val by (S-V) and $\mathtt{s}(\gamma(e)) \mapsto^* \mathtt{s}(\gamma(e))$ by reflexivity.
3. $\mathtt{s}(\gamma(e)) \downarrow$ by ($\downarrow$-S) (note that the premises of the rule are satisfied by what we have above).

Case $\Gamma \vdash \mathtt{natrec}(e; e_0; x.y.e_1) : \tau$ (Rec): We need to show $\mathsf{Red}_\tau(\gamma(\mathtt{natrec}(e; e_0; x.y.e_1)))$, i.e. $\mathsf{Red}_\tau(\mathtt{natrec}(\gamma(e); \gamma(e_0); x.y.\gamma(e_1)))$ by definition of substitution.

From the premises of the rule, we have $\Gamma \vdash e : \mathtt{nat}$, $\Gamma \vdash e_0 : \tau$ and $\Gamma, x : \mathtt{nat}, y : \tau \vdash e_1 : \tau$. By assumption, we also have $\gamma \Vdash \Gamma$.

By the substitution lemma on $\Gamma, x : \mathtt{nat}, y : \tau \vdash e_1 : \tau$, we have $x : \mathtt{nat}, y : \tau \vdash \gamma(e_1) : \tau$. By I.H. on the first two premises, we have $\mathsf{Red}_{\mathtt{nat}}(\gamma(e))$ and $\mathsf{Red}_\tau(\gamma(e_0))$. From these, we also have $\cdot \vdash \gamma(e) : \mathtt{nat}$ and $\cdot \vdash \gamma(e_0) : \tau$. Hence, $\cdot \vdash \mathtt{natrec}(\gamma(e); \gamma(e_0); x.y.\gamma(e_1)) : \tau$ by (Rec).

From $\mathsf{Red}_{\mathtt{nat}}(\gamma(e))$, we also have $\gamma(e) \mapsto^* v$ for some v where v val and $v \downarrow$. By congruence with (Rec-S), we have $\mathtt{natrec}(\gamma(e); \gamma(e_0); x.y.\gamma(e_1)) \mapsto^* \mathtt{natrec}(v; \gamma(e_0); x.y.\gamma(e_1))$.

Using closure under head expansion, it suffices to show $\mathsf{Red}_\tau(\mathtt{natrec}(v; \gamma(e_0); x.y.\gamma(e_1)))$. We do this by nested induction on the $v \downarrow$ judgement. Note that we have $\cdot \vdash (\mathtt{natrec}(v; \gamma(e_0); x.y.\gamma(e_1))) : \tau$ and $\cdot \vdash v : \mathtt{nat}$ by type preservation.

Sub-case ($\downarrow$-Z): We have $v = \mathtt{z}$, so $\mathtt{natrec}(\mathtt{z}; \gamma(e_0); x.y.\gamma(e_1)) \mapsto \gamma(e_0)$ by (Rec-IZ). Since $\mathsf{Red}_\tau(\gamma(e_0))$, we have $\mathsf{Red}_\tau(\mathtt{natrec}(\mathtt{z}; \gamma(e_0); x.y.\gamma(e_1)))$ by closure under head expansion.

Sub-case ($\downarrow$-S): We have $v = \mathtt{s}(e')$ for some e', v' such that $e' \mapsto^* v'$, $v' val$ and $v' \downarrow$. By inversion on the typing judgment for v, we have $\cdot \vdash e' : \mathtt{nat}$. Therefore, $\mathsf{Red}_{\mathtt{nat}}(e')$ by definition. By the nested I.H. on $v' \downarrow$, we have $\mathsf{Red}_\tau(\mathtt{natrec}(v'; \gamma(e_0); x.y.\gamma(e_1)))$. We also have $\cdot \vdash \mathtt{natrec}(e'; \gamma(e_0); x.y.\gamma(e_1)) : \tau$ by (Rec), and therefore, $\mathsf{Red}_\tau(\mathtt{natrec}(e'; \gamma(e_0); x.y.\gamma(e_1)))$ by closure under head expansion and congruence with (Rec-S).

Let γ' be the mapping $\{x \hookrightarrow e', y \hookrightarrow \mathtt{natrec}(e'; \gamma(e_0); x.y.\gamma(e_1))\}$. We have that $\gamma' \circ \gamma \Vdash \Gamma, x : \mathtt{nat}, y : \tau$. Therefore, by the outer I.H., we have $\mathsf{Red}_\tau(\gamma' \circ \gamma(e_1))$ i.e. $\mathsf{Red}_\tau(\gamma'(\gamma(e_1)))$ since γ, γ' are disjoint finite maps and γ only contains closed terms. Finally, we have $\texttt{natrec}(\texttt{s}(e'); \gamma(e_0); x.y.\gamma(e_1)) \mapsto \gamma'(\gamma(e_1))$ by (Rec-IS), and $\mathsf{Red}_\tau(\texttt{natrec}(\texttt{s}(e'); \gamma(e_0); x.y.\gamma(e_1)))$ by closure under head expansion.

Case $\Gamma \vdash \lambda x{:}\tau'.e : \tau' \to \tau$ (Lam): By premise of the rule, we have $\Gamma, x : \tau' \vdash e : \tau$. We need to show $\mathsf{Red}_{\tau' \to \tau}(\gamma(\lambda x{:}\tau'.e))$, i.e. $\mathsf{Red}_{\tau' \to \tau}(\lambda x{:}\tau'.\gamma(e)))$ by definition of substitution.

1. By the substitution lemma, we have $x : \tau' \vdash \gamma(e) : \tau$. Therefore, $\cdot \vdash \lambda x{:}\tau'.\gamma(e) : \tau' \to \tau$ by (Lam)[2].
2. $\lambda x{:}\tau'.\gamma(e)$ val by (Lam-V), and $\lambda x{:}\tau'.\gamma(e) \mapsto^* \lambda x{:}\tau'.\gamma(e)$ by reflexivity.
3. Consider e' such that $\mathsf{Red}_{\tau'}(e')$. Let γ' be the mapping $\{x \hookrightarrow e'\}$. Consider the composed mapping $\gamma' \circ \gamma$. We have $\gamma' \circ \gamma \Vdash \Gamma, x : \tau'$, so by I.H., we have $\mathsf{Red}_\tau(\gamma' \circ \gamma(e))$ i.e. $\mathsf{Red}_\tau(\gamma'(\gamma(e)))$ since γ, γ' are disjoint finite maps and γ only contains closed terms. We have $(\lambda x{:}\tau'.\gamma(e))e' \mapsto \gamma'(\gamma(e))$ by (App-I). Since $\cdot \vdash e' : \tau'$, we also have $\cdot \vdash ((\lambda x{:}\tau'.\gamma(e))e') : \tau$ by (App). Therefore, by closure under head expansion, $\mathsf{Red}_\tau((\lambda x{:}\tau'.\gamma(e))e')$.

□

Task 3 *For each definition below, briefly explain (in 1-2 lines) the intuition behind your answer.*

1. *Define* `mult`*, where* `mult` $\overline{m}\,\overline{n} \mapsto^* \overline{m * n}$.[3]
2. *Define* `minus`*, where* `minus` $\overline{m}\,\overline{n} \mapsto^* \overline{m - n}$ *if* $m > n$*. It should produce* 0 *otherwise.*
3. *Define* `leq`*, where* `leq` $\overline{m}\,\overline{n} \mapsto^*$ `s(z)` *if* $m \le n$ *and* `leq` $\overline{m}\,\overline{n} \mapsto^*$ `z` *otherwise.*
4. *Define* `mod`*, where* `mod` $\overline{m}\,\overline{n} = \overline{m \mod n}$.

Solution: I write out the "ML" code for these definitions.

1. We use a helper function to do addition and then implement multiplication by recursive addition.

```
fun plus Z n = n
|   plus (S m) n = S (plus m n);
fun mul Z n = Z
|   mul (S m) n = plus n (mul m n);
```

$\texttt{plus} = \lambda m{:}\texttt{nat}, n{:}\texttt{nat}.\texttt{natrec}(m; n; x.y.\texttt{s}(y))$

$\texttt{mult} = \lambda m{:}\texttt{nat}, n{:}\texttt{nat}.\texttt{natrec}(m; \texttt{z}; x.y.\texttt{plus}\ n\ y)$

2. We use a helper function to define the (bounded) predecessor, which allows us to define (bounded) minus by recursively taking the predecessor.

```
fun pred Z = Z
|   pred (S m) = m;
fun minus m Z = m
|   minus m (S n) = pred (minus m n);
```

[2]Many students got this directly from the substitution lemma since we already know $\Gamma \vdash e : \tau' \to \tau$ and $\Gamma \Vdash \gamma$.

[3]We will write $\overline{n}$ to indicate the representation of a natural number n as an element of type `nat`.

$\texttt{pred} = \lambda m{:}\texttt{nat}.\texttt{natrec}(m; \texttt{z}; x.y.x)$

$\texttt{minus} = \lambda m{:}\texttt{nat}, n{:}\texttt{nat}.\texttt{natrec}(n; m; x.y.\texttt{pred}\ y)$

3. We define an if-(zero)-then-else combinator, and use it together with the definition of minus.

```
fun ifz Z t f = t
|   ifz (S m) t f = f;
fun leq m n = ifz (minus m n) (S Z) Z;
```

$\texttt{ifz} = \lambda m{:}\texttt{nat}, t{:}\tau, f{:}\tau.\texttt{natrec}(m; t; x.y.f)$

$\texttt{leq} = \lambda m{:}\texttt{nat}, n{:}\texttt{nat}.\texttt{ifz}\ (\texttt{minus}\ m\ n)\ (\texttt{s}(\texttt{z}))\ \texttt{z}$

4. The definition for `mod` works using the following property: $m \mod n = (m - 1 + 1) \mod n = ((m-1) \mod n+1) \mod n$. This works if $n > 0$ but modulus by 0 is undefined, so we set $m \mod 0 = 0$.

```
fun mod Z n = Z
|   mod (S m) n =
  let val y = mod m n in          (* y is m mod n *)
    ifz (leq n (S y)) (S y) Z     (* this implements (y+1) mod n *)
  end
```

$\texttt{mod} = \lambda m{:}\texttt{nat}, n{:}\texttt{nat}.\texttt{natrec}(m; \texttt{z}; x.y.\texttt{ifz}\ (\texttt{leq}\ n\ \texttt{s}(y))\ (\texttt{s}(y))\ \texttt{z})$

I got a variety of great alternative solutions for this question. For example:

(a) Repeatedly subtracting n from m until $m < n$.

(b) Add 1 modulo n for m times (similar to mine above).

(c) Using $m - n * (m/n)$ with the definition of / given in class.

(d) Finding the largest k such that $k * n \leq m$, then $m - k * n$.

(e) One of the above, but using the fixed-point iteration introduced in class.

Task 4 *Define* $\texttt{natrec}(e; e_0; x.y.e_1)$ *in terms of* `rec`.

Solution: Intuitively, we need to track both x, y provided by `natrec`, so we make `rec` return a pair instead. The first projection is used to return the current value of the natural number that is being recursed on, while the second projection tracks the usual result of the recursive call.

$\texttt{natrec}(e; e_0; x.y.e_1) = \pi_2(\texttt{rec}(e; (\texttt{z}, e_0); z.\texttt{s}(\pi_1(z)), [\pi_1(z), \pi_2(z)/x, y](e_1)))$