# Homework 5: Representation Independence in System F

15-814: Types and Programming Languages
TA: Carlo Angiuli (cangiuli@cs.cmu.edu)

Out: 11/23/13 (updated 11/27/13)
Due: 12/6/13 (end of day)

Please submit your work as a PDF to `cangiuli@cs.cmu.edu`. Include the phrase "15-814 Homework 5" in the subject line of your email. In this assignment, we assume the same statics and dynamics for System F as in the previous assignment.

## 1 Logical Equivalence

The fact that logical equivalence is reflexive is the key result in our study of System F. For the most part, the proof proceeds along the same lines as the proof of hereditary termination for Gödel's T, which you completed in Homework 3.

However, the situation is more complicated here. We wish to define logical equivalence inductively on types, but the $\forall$ quantifier requires that we define it also for open types, in the same way that we had to consider hereditary termination of open terms in Gödel's T.

Say that $\delta$ is a *total substitution for a type context* $\Delta$ if it is a mapping from $\mathsf{dom}(\Delta)$ to closed types—that is, for each $t$ $\mathsf{type}$ in $\Delta$, $\cdot \vdash \delta(t)$ $\mathsf{type}$. Given a total substitution $\delta$ for $\Delta$ and a type $\Delta \vdash \tau$ $\mathsf{type}$, we can form a closed type $\hat{\delta}(\tau)$ by performing the substitution. Since terms can also contain free type variables, we can similarly substitute into any $\Delta; \Gamma \vdash M : \tau$ to obtain $\cdot; \Gamma \vdash \hat{\delta}(M) : \hat{\delta}(\tau)$.

As in Homework 3, $\gamma$ is a *total substitution for a term context* $\Gamma$ if it is a mapping from $\mathsf{dom}(\Gamma)$ to closed terms; then given $\gamma$ and $\Delta; \Gamma \vdash M : \tau$, we can perform the substitution and obtain $\Delta; \cdot \vdash \hat{\gamma}(M) : \tau$.

Given two total substitutions $\delta, \delta'$ for $\Delta$, an *assignment of splendid relations* between them is a choice of splendid relation $\eta(t)$ between $\delta(t)$ and $\delta'(t)$, for each $t \in \mathsf{dom}(\Delta)$. We notate this situation as $\eta : \delta \leftrightarrow \delta'$, and we notate the extension of $\eta$ mapping $t$ to $R$ by $\eta, t \hookrightarrow R$. The relation $R$ is *splendid* if it is closed under head expansion (a.k.a. converse evaluation) and observational equivalence.

For any type $\Delta \vdash \tau$ $\mathsf{type}$, and an assignment of relations $\eta : \delta \leftrightarrow \delta'$ for $\Delta$, we define a binary relation $[\![\tau]\!]_\eta$ in the following way, by induction on $\tau$:

**Definition 1** (Logical equivalence)**.** *For two terms $\Delta; \Gamma \vdash M, N : \tau$, two total substitutions $\delta, \delta'$ for $\Delta$, an assignment $\eta : \delta \leftrightarrow \delta'$, and total substitutions $\gamma, \gamma'$ such that $(\gamma, \gamma') \in [\![\Gamma]\!]_\eta$, we say that*

- $(M, N) \in [\![\mathbf{2}]\!]_\eta$ *if $\hat{\gamma}(\hat{\delta}(M)), \widehat{\gamma'}(\widehat{\delta'}(N))$ terminate, and $\hat{\gamma}(\hat{\delta}(M)) \simeq \widehat{\gamma'}(\widehat{\delta'}(N))$.*
- $(M, N) \in [\![\tau_1 \to \tau_2]\!]_\eta$ *if $\hat{\gamma}(\hat{\delta}(M)) \mapsto^* \lambda x : \tau_1 . M'$ and $\widehat{\gamma'}(\widehat{\delta'}(N)) \mapsto^* \lambda x : \tau_1 . N'$, and for any $(M'', N'') \in [\![\tau_1]\!]_\eta$, $([M''/x]M', [N''/x]N') \in [\![\tau_2]\!]_\eta$.*
- $(M, N) \in [\![t]\!]_\eta$ *if $\hat{\gamma}(\hat{\delta}(M)), \widehat{\gamma'}(\widehat{\delta'}(N))$ terminate, and $(\hat{\gamma}(\hat{\delta}(M)), \widehat{\gamma'}(\widehat{\delta'}(N))) \in \eta(t)$.*

- $(M, N) \in [\![\forall t.\tau]\!]_\eta$ *if* $\hat{\gamma}(\hat{\delta}(M)) \mapsto^* \Lambda t.M'$ *and* $\widehat{\gamma'}(\widehat{\delta'}(N)) \mapsto^* \Lambda t.N'$*, and for any closed types* $\tau_1, \tau_2$ *and splendid relation* $R$ *between them,* $([\tau_1/t]M', [\tau_2/t]N') \in [\![\tau]\!]_{\eta, t \hookrightarrow R}$.

*Finally,* $(\gamma, \gamma') \in [\![\Gamma]\!]_\eta$ *if for each* $(x : \tau) \in \Gamma$, $(\gamma(x), \gamma'(x)) \in [\![\tau]\!]_\eta$.

**Warning:** There are a lot of symbols here, so stop and make sure you understand the idea behind the definition. Although it looks much more complicated, this is just a generalization of the definitions of hereditary termination and of logical equivalence from the previous two homework assignments.

We are simply extending the usual relation to open terms by saying that free term variables can be instantiated with any closed terms in the relation, and free type variables can be "instantiated" with any splendid relation between closed types (where the instantiation is accomplished via an environment $\eta$ of interpretations of the type variables, since we cannot literally substitute the relation $R$ for the type variable $t$).

We need the following lemma, which says that this latter notion of instantiation coincides with substitution, in the case that we instantiate a type variable to $[\![\tau]\!]$. for some closed type $\tau$.

**Lemma 1.1.** *For* $\Delta \vdash \tau$ type *and* $\Delta, t$ type $\vdash \tau_1$ type*, total substitutions* $\delta, \delta'$ *for* $\Delta$*, and* $\eta : \delta \leftrightarrow \delta'$*, we have* $(M, N) \in [\![\tau_1]\!]_{\eta, t \hookrightarrow [\![\tau]\!]_\eta}$ *iff* $(M, N) \in [\![[\tau/t]\tau_1]\!]_\eta$.

Note that, for this lemma to make sense, we need to know that logical equivalence is itself splendid. Now we are ready to prove that logical equivalence is reflexive!

**Task 1.** *Prove that, if* $\Delta; \Gamma \vdash M : \tau$*, then for any total substitutions* $\delta, \delta'$ *for* $\Delta$*, assignment* $\eta : \delta \leftrightarrow \delta'$*, and total substitutions* $\gamma, \gamma'$ *for* $\Gamma$ *with* $(\gamma, \gamma') \in [\![\Gamma]\!]_\eta$*, it is the case that* $(\hat{\gamma}(\hat{\delta}(M)), \widehat{\gamma'}(\widehat{\delta'}(M))) \in [\![\tau]\!]_\eta$.

***Prove only the cases corresponding to the following two rules:***

$$\frac{\Delta, t \text{ type}; \Gamma \vdash M : \tau}{\Delta; \Gamma \vdash \Lambda t.M : \forall t.\tau} \qquad \frac{\Delta; \Gamma \vdash M : \forall t.\tau_1 \quad \Delta \vdash \tau \text{ type}}{\Delta; \Gamma \vdash M[\tau] : [\tau/t]\tau_1}$$

*(The other cases work out similarly to those on Homework 3.)*

## 2 Free Theorems

We can prove very strong theorems about arbitrary terms in System F, purely by virtue of their types. For example, in class, we proved that all terms of type $\forall t.t \to t$ are observationally equivalent to the polymorphic identity function $\Lambda t.\lambda x : t.\, x$.

These are often called "free theorems" because they are systematic to verify—they are all straightforward corollaries of Task 1. These serve as concrete motivations for our abstract definition of logical equivalence.

**Task 2.** *Every term of type* $\forall t.t \to t \to t$ *(i.e., every Church boolean) is observationally equivalent to either* $\Lambda t.\lambda x : t.\, \lambda y : t.\, x$ *or* $\Lambda t.\lambda x : t.\, \lambda y : t.\, y$.

**(Hint)** The proof is similar to the one for $\forall t.t \to t$, and mostly involves unwinding the definition of logical equivalence at the given type.

## 3 Representation Independence

Recall from the definition of logical equivalence that if we instantiate any polymorphic type at two different types whose terms we can relate, then the two instantiations must also be related.

Because we can encode existential types using polymorphism, we are able to reason about two different implementations of any interface.

In this task, we will consider two implementations of an existential type representing a list interface, and we will prove that users of either implementation will obtain the same result, modulo the relation between the two implementations. This is the essence of the abstraction theorem we discussed in class.

**Task 3.** *True or false—The abstraction theorem is the most important theorem in computer science. Justify your answer.*

**(Hint)** Answer freely, but justify your answer in either case.

To set up our example, we will extend System F with natural numbers, lists of natural numbers, and products. Since these can be Church-encoded, doing so does not add expressive power; it is purely for convenience.

$$\begin{aligned}\tau &:= \cdots \mid \mathsf{nat} \mid \mathsf{list} \mid \tau \times \tau \\ M &:= \cdots \mid \mathsf{z} \mid \mathsf{suc}(M) \mid \mathsf{natrec}(M, N_0, x.N_1) \mid \mathsf{nil} \mid \mathsf{cons}(M, M) \mid \mathsf{foldr}(M, N_0, x.y.N_1) \\ &\quad \mid (M, M) \mid \mathsf{fst}(M) \mid \mathsf{snd}(M)\end{aligned}$$

$$\frac{}{\Delta; \Gamma \vdash \mathsf{z} : \mathsf{nat}} \qquad \frac{\Delta; \Gamma \vdash M : \mathsf{nat}}{\Delta; \Gamma \vdash \mathsf{suc}(M) : \mathsf{nat}}$$

$$\frac{\Delta; \Gamma \vdash M : \mathsf{nat} \quad \Delta; \Gamma \vdash N_0 : \tau \quad \Delta; \Gamma, x : \tau \vdash N_1 : \tau}{\Delta; \Gamma \vdash \mathsf{natrec}(M, N_0, x.N_1) : \tau}$$

$$\frac{}{\Delta; \Gamma \vdash \mathsf{nil} : \mathsf{list}} \qquad \frac{\Delta; \Gamma \vdash M : \mathsf{nat} \quad \Delta; \Gamma \vdash N : \mathsf{list}}{\Delta; \Gamma \vdash \mathsf{cons}(M, N) : \mathsf{list}}$$

$$\frac{\Delta; \Gamma \vdash M : \mathsf{list} \quad \Delta; \Gamma \vdash N_0 : \tau \quad \Delta; \Gamma, x : \tau, y : \mathsf{nat} \vdash N_1 : \tau}{\Delta; \Gamma \vdash \mathsf{foldr}(M, N_0, x.y.N_1) : \tau}$$

$$\frac{\Delta; \Gamma \vdash M : \tau_1 \quad \Delta; \Gamma \vdash N : \tau_2}{\Delta; \Gamma \vdash (M, N) : \tau_1 \times \tau_2} \qquad \frac{\Delta; \Gamma \vdash M : \tau_1 \times \tau_2}{\Delta; \Gamma \vdash \mathsf{fst}(M) : \tau_1} \qquad \frac{\Delta; \Gamma \vdash M : \tau_1 \times \tau_2}{\Delta; \Gamma \vdash \mathsf{snd}(M) : \tau_2}$$

We use the usual dynamics for these constructs. In particular,

$$\begin{aligned}\mathsf{foldr}(\mathsf{nil}, N_0, x.y.N_1) &\mapsto N_0 \\ \mathsf{foldr}(\mathsf{cons}(M, L), N_0, x.y.N_1) &\mapsto [\mathsf{foldr}(L, N_0, x.y.N_1)/x][M/y]N_1\end{aligned}$$

Our first implementation of lists will be the $\mathsf{nil}, \mathsf{cons}, \mathsf{foldr}$ above. Our second will be pairs of a $\mathsf{nat}$ and $\mathsf{nat} \to \mathsf{nat}$, where the first is the length of the list, and the second sends any $n$ to the $n$th element from the end of the list. Both of these implement the existential type

$$\exists t.(t \times (\mathsf{nat} \to t \to t)) \times (\forall u.t \to u \to (u \to \mathsf{nat} \to u) \to u)$$

In the below definition, $\mathsf{pred}$ is the proper predecessor, and $\mathsf{sub}$ is proper subtraction. (If they would mathematically yield negative numbers, they instead return $\mathsf{z}$.) Note that $\overline{\mathsf{head}}$ and $\overline{\mathsf{tail}}$ are not needed to satisfy the list interface, but we define them for convenience.

$$\begin{aligned}\overline{\mathsf{list}} &:= \mathsf{nat} \times (\mathsf{nat} \to \mathsf{nat}) \\ \overline{\mathsf{nil}} &:= (\mathsf{z}, \lambda x : \mathsf{nat}.\, x) \\ \overline{\mathsf{cons}} &:= \lambda n : \mathsf{nat}.\, \lambda \ell : \overline{\mathsf{list}}.\, (\mathsf{suc}(\mathsf{fst}(\ell)), \lambda x : \mathsf{nat}.\, \mathsf{natrec}(\mathsf{sub}\, \mathsf{fst}(\ell)\, x, n, y.\mathsf{snd}(\ell)\, x)) \\ \overline{\mathsf{foldr}} &:= \forall u.\lambda \ell : \overline{\mathsf{list}}.\, \lambda m : u.\, \lambda n : (u \to \mathsf{nat} \to u). \\ &\qquad \mathsf{fst}(\mathsf{natrec}(\mathsf{fst}(\ell), (m, \mathsf{z}), x.(n\, \mathsf{fst}(x)\, (\mathsf{snd}(\ell)\, \mathsf{snd}(x)), \mathsf{suc}(\mathsf{snd}(x))))) \\ \overline{\mathsf{head}} &:= \lambda \ell : \overline{\mathsf{list}}.\, \mathsf{snd}(\ell)\, \mathsf{pred}(\mathsf{fst}(\ell)) \\ \overline{\mathsf{tail}} &:= \lambda \ell : \overline{\mathsf{list}}.\, (\mathsf{pred}(\mathsf{fst}(\ell)), \mathsf{snd}(\ell))\end{aligned}$$

**Warning:** It will be very difficult to complete the following tasks unless you convince yourself this implements lists. I recommend playing around with this code on paper, or in your favorite functional language.

Given $L : \mathsf{list}$ and $P : \overline{\mathsf{list}}$, say that $(L, P) \in S$ iff:

- $L \cong \mathsf{nil}$ and $\mathsf{fst}(P) \cong \mathsf{z}$, or
- $L \cong \mathsf{cons}(N, L')$, $\mathsf{fst}(P) \cong \mathsf{suc}(M)$ for some $M$, $\overline{\mathsf{head}}(P) \cong N$, and $(L', \overline{\mathsf{tail}}(P)) \in S$.

Because $S$ is defined only in terms of observational equivalence, it is clear that $S$ is splendid. Your task will be to prove that $\mathsf{nil}, \overline{\mathsf{nil}}$, $\mathsf{cons}, \overline{\mathsf{cons}}$, and $\mathsf{foldr}, \overline{\mathsf{foldr}}$ all preserve this relation in a certain sense. This implies that $\mathsf{list}$ and $\overline{\mathsf{list}}$ cannot be distinguished as implementations of the existential interface for lists.

You may use any theorems from this assignment or proven during class, including the fact that Task 1 remains true in the presence of $\mathsf{nat}$, $\mathsf{list}$, and $\times$. (For example, you will need that $\cong$ is a congruence, and is closed under reduction.) You will also need the following more specific lemmas:

**Lemma 3.1.** *If* $M : \mathsf{nat}$*, then* $\mathsf{pred}(\mathsf{suc}(M)) \cong M$*.*

**Lemma 3.2.** *If* $M : \mathsf{nat}$*, then* $\mathsf{sub}\, M\, M \cong \mathsf{z}$*.*

**Lemma 3.3.** *If* $(L, P) \in S$*, then for any* $N : \mathsf{nat}$*,* $(L, \overline{\mathsf{tail}}(\overline{\mathsf{cons}}\, N\, P)) \in S$*.*

**Lemma 3.4.** *For* $M : \mathsf{nat}$ *and* $m, n, \ell$ *of the correct types,*

$$\mathsf{snd}(\mathsf{natrec}(M, (m, \mathsf{z}), x.(n\, \mathsf{fst}(x)\, (\mathsf{snd}(\ell)\, \mathsf{snd}(x)), \mathsf{suc}(\mathsf{snd}(x))))) \cong M$$

*(The idea is that the second component keeps track of the current position in the list, so it ends at the length of the list.)*

If you feel you need any other lemmas (within reason), feel free to state them and assume them without proof. The goal of these tasks is not to force you to prove mundane lemmas about proper subtraction, etc.

**Task 4.** *Prove that* $(\mathsf{nil}, \overline{\mathsf{nil}}) \in S$*.*

**Task 5.** *Prove that for closed terms* $L : \mathsf{list}$*,* $P : \overline{\mathsf{list}}$*, and* $N : \mathsf{nat}$ *such that* $(L, P) \in S$*,* $(\mathsf{cons}(N, L), \overline{\mathsf{cons}}\, N\, P) \in S$*.*

**Task 6.** *Prove that for any closed type* $\tau$*, and any closed terms* $L : \mathsf{list}$*,* $P : \overline{\mathsf{list}}$*,* $M : \tau$*, and* $N : \tau \to \mathsf{nat} \to \tau$ *such that* $(L, P) \in S$*,* $(\mathsf{foldr}(L, M, x.y.N\, x\, y), \overline{\mathsf{foldr}}[\tau]\, P\, M\, N) \in [\![\tau]\!]$*.*

**(Hint)** Consider the two ways that $(L, P) \in S$ could be true.