Homework 2: Principles of System T
15-814: Types and Programming Languages
Fall 2016
Instructor: Karl Crary
TA: Ankush Das

Out: Sep 28, 2016 (5 pm)
Due: Oct 5, 2016 (11 pm)

Notes:

• Welcome to 15-814’s second homework assignment!
• Please email your work as a PDF file to ankushd@cs.cmu.edu titled “15-814 Homework 2”. Your PDF should be named “<your-name>-hw2-sol.pdf”.

1 Termination in System T

Gödel’s System T, presented in Appendix A, as we defined it in class, has the valuable property that any program we can write will evaluate to a value in a finite number of steps. In this section, we will look at how to prove this fact using Tait’s reducibility method, which is an instance of the ubiquitous technique of logical relations. The theorem we want to prove is the following:

Theorem (Normalization): If $\Gamma \vdash e : \tau$, then there exists $v \text{ val}$ such that $e \rightarrow^* v$.

(Here, $\rightarrow^*$ is the transitive closure of the step judgment $\rightarrow$.) We might hope to prove this theorem directly by induction on the typing judgment. However, this approach is insufficient. The case for the application rule (App) is demonstrative.

$$
\begin{array}{c}
\Gamma \vdash e_1 : \tau' \rightarrow \tau \\
\Gamma \vdash e_2 : \tau'
\end{array} \quad \text{(App)}
\end{array}
\begin{array}{c}
\Gamma \vdash e_1 e_2 : \tau
\end{array}
$$

In this case, our induction hypotheses tell us that $e_1 \rightarrow^* v_1$ and $e_2 \rightarrow^* v_2$ for some values $v_1$ and $v_2$. By preservation and the appropriate canonical forms lemma, we know that $v_1 = \lambda x : \tau'. e'$ for some $e'$. It follows that $e_1 e_2 \rightarrow^* v_1 e_2 \rightarrow [e_2/x]e'$. Unfortunately, we are now stuck, as we have no information about the behavior of $[e_2/x]e'$.

We will solve this by generalizing, proving a stronger statement which gives us more information as an induction hypothesis. Specifically, we will define a reducibility predicate $\text{Red}_\tau(e)$ and prove the following theorem.

Theorem A: If $\Gamma \vdash e : \tau$, then $\text{Red}_\tau(e)$.

Since we’ll define $\text{Red}_\tau$ such that $\text{Red}_\tau(e)$ implies the existence of $v \text{ val}$ with $e \rightarrow^* v$, this will give normalization as a special case. The definition will go by structural induction on the type $\tau$, which makes $\text{Red}_\tau$ what is called a logical relation. (In particular, it is a unary logical relation; we will encounter binary logical relations, such as logical equivalence $e \sim_\tau e'$, later in the course.) Actually, we will prove an even more general theorem in order to account for open terms; to state it concisely, we first want to define some notation for substitutions.
**Definition:** A substitution $\gamma = \{x_1 \mapsto e_1, \ldots, x_n \mapsto e_n\}$ is a finite mapping from variables to terms. Given an expression $e$, we write $\gamma(e)$ for the expression $[e_1, \ldots, e_n/x_1, \ldots, x_n]e$, that is, the simultaneous substitution in $e$ of each expression $e_i$ for its corresponding variable $x_i$. For $\gamma$ as above, we define $\gamma \models \Gamma$ to mean that $\Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n$ for some $\tau_1, \ldots, \tau_n$ such that $\text{Red}_{\tau_i}(e_i)$ holds for $1 \leq i \leq n$.

Now we state the theorem we will actually prove:

**Theorem B:** If $\Gamma \vdash e : \tau$ and $\gamma \models \Gamma$ then $\text{Red}_{\tau}(\gamma(e))$.

Theorem A follows as the special case where $\Gamma = \cdot$ and $\gamma = \langle \rangle$. Finally, we define the predicate $\text{Red}_r$ by structural induction on $\tau$:

- $\text{Red}_{\tau_1 \rightarrow \tau_2}(e)$ holds if
  1. $\cdot \vdash e : \tau_1 \rightarrow \tau_2$,
  2. there exists $v \text{ val}$ such that $e \mapsto^* v$, and
  3. for any $e'$ such that $\text{Red}_{\tau_1}(e')$, we have $\text{Red}_{\tau_2}(ee')$.

- $\text{Red}_{\text{nat}}(e)$ holds if
  1. $\cdot \vdash e : \text{nat}$,
  2. there exists $v \text{ val}$ such that $e \mapsto^* v$, and
  3. $v \downarrow$, where $v \downarrow$ is a judgment defined by

$$
\begin{array}{c}
\frac{\text{Red}_{\tau_1}(e)}{\downarrow} \quad (\downarrow Z) \\
\frac{e \mapsto^* v \quad v \text{ val} \quad v \downarrow}{\text{Red}_{\tau_2}(s(e) \downarrow \text{val})} (\downarrow \text{S})
\end{array}
$$

Note that $\text{Red}_{\tau_1 \rightarrow \tau_2}(e)$ is defined in terms of $\text{Red}$ at the structurally smaller types $\tau_1$ and $\tau_2$, so the definition is well-founded. To get you started on the proof, and to see how this definition succeeds where the previous attempt failed, here is the (App) case:

- **Case (App):** We have $\Gamma \vdash e_1 e_2 : \tau$ with $\Gamma \vdash e_1 : \tau' \rightarrow \tau$ and $\Gamma \vdash e_2 : \tau'$ for some $\tau'$. Per the theorem statement, we assume we are given $\gamma \models \Gamma$ and want to prove that $\text{Red}_{\tau}(\gamma(e_1 e_2))$. By definition of substitution, we have that $\gamma(e_1 e_2) = \gamma(e_1) \gamma(e_2)$. Moreover, our induction hypotheses tell us that $\text{Red}_{\tau' \rightarrow \tau}(\gamma(e_1))$ and $\text{Red}_{\tau'}(\gamma(e_2))$. From condition 3 in the definition of $\text{Red}_{\tau' \rightarrow \tau}$, we know that for any $e'$ with $\text{Red}_{\tau'}(e')$ we have $\text{Red}_{\tau}(\gamma(e_1) e')$. Taking $e' = \gamma(e_2)$ thus gives our goal.

With the right definition $\text{Red}$ in hand, the (App) case follows almost trivially. On the other hand, the (LAM) case becomes more difficult. In general, though, proving the theorem is the easy part of a logical relations argument – the hard part is choosing the right theorem to prove.

To complete the proof, you’ll need the following lemma.

**Lemma (Closure under Head Expansion):** If $\text{Red}_{\tau}(e')$ and $\cdot \vdash e : \tau$ and $e \mapsto e'$, then $\text{Red}_{\tau}(e)$.

**Task 1** Prove closure under head expansion.
With the help of Preservation, closure under head expansion extends to apply when \( e \mapsto e' \) in multiple steps (you can use this without proof).

**Task 2** Prove the remaining cases of Theorem B.

## 2 Programming in System T

Consider an extension of Gödel’s T with products. We will call this language LNS.

\[
\begin{align*}
\tau &::= \tau_1 \times \tau_2 \mid \text{nat} \\
e &::= \cdots \mid z \mid s(e) \mid \text{natrec}(e; e; x.y.e)
\end{align*}
\]

(The rules for products are as in PFPL 10; we’ll assume lazy evaluation for successor.) First we’ll get some practice working with nats. You may find it helpful to break large definitions into intermediate ones, give them names, and refer to them in the larger definition.

**Task 3**

1. Define \( \text{mult} \), where \( \text{mult} \ m \ n \mapsto m \times n \).
2. Define \( \text{minus} \), where \( \text{minus} \ m \ n \mapsto m - n \) if \( m > n \). It should produce 0 otherwise.
3. Define \( \text{leq} \), where \( \text{leq} \ m \ n \mapsto s(z) \) if \( m \leq n \) and \( \text{leq} \ m \ n \mapsto z \) otherwise.
4. Define \( \text{mod} \), where \( \text{mod} \ m \ n = m \mod n \).

In class, we briefly mentioned iteration and primitive recursion in relation to the operator \( \text{natrec} \). In this context, \( \text{natrec} \) in LNS corresponds to primitive recursion and may be called the recursor. We also defined an alternate form \( \text{rec}(e; e_0; y.e_1) \) which corresponds to iteration.

\[
\begin{align*}
e &\mapsto e' \\
\text{rec}(e; e_0; y.e_1) &\mapsto \text{rec}(e'; e_0; y.e_1) \\
\text{rec}(z; e_0; y.e_1) &\mapsto e_0 \\
\text{rec}(s(e); e_0; y.e_1) &\mapsto [\text{rec}(e; e_0; y.e_1)/y]e_1
\end{align*}
\]

It is simple to define \( \text{rec}(e; e_0; x.e_1) \) in terms of \( \text{natrec}(e; e_0; x.y.e_1) \); simply ignore \( x \). Using the other constructs of LNS, it is also possible to define \( \text{natrec} \) in terms of \( \text{rec} \).

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

\[\text{1} \text{We will write } \pi \text{ to indicate the representation of a natural number } n \text{ as an element of type } \text{nat}.\]
A System T

A.1 Statics

\[
\begin{align*}
\frac{}{\Gamma, x : \tau \vdash x : \tau} \quad (\text{HYP}) \quad \frac{}{\Gamma \vdash z : \text{nat}} \quad (\text{Z}) \quad \frac{\Gamma \vdash e : \text{nat}}{\Gamma \vdash s(e) : \text{nat}} \quad (\text{S}) \\
\frac{\Gamma \vdash e : \text{nat}}{\Gamma \vdash e_0 : \tau} \quad \frac{\Gamma, x : \tau \vdash x : \tau \quad \Gamma \vdash e : \text{nat}, y : \tau \vdash e_1 : \tau}{\Gamma \vdash \text{natrec}(e; e_0; x.y.e_1) : \tau} \quad (\text{REC}) \\
\frac{\Gamma, x : \tau' \vdash e : \tau}{\Gamma \vdash \lambda x : \tau'. e : \tau' \rightarrow \tau} \quad (\text{LAM}) \quad \frac{\Gamma \vdash e_1 : \tau' \rightarrow \tau \quad \Gamma \vdash e_2 : \tau'}{\Gamma \vdash e_1e_2 : \tau} \quad (\text{APP})
\end{align*}
\]

A.2 Dynamics

\[
\begin{align*}
\frac{}{z \text{ val}} \quad (\text{Z-V}) \quad \frac{}{s(e) \text{ val}} \quad (\text{S-V}) \quad \frac{e \mapsto e'}{\text{natrec}(e; e_0; x.y.e_1) \mapsto \text{natrec}(e'; e_0; x.y.e_1)} \quad (\text{Rec-S}) \\
\frac{}{\text{natrec}(z; e_0; x.y.e_1) \mapsto e_0} \quad (\text{Rec-IZ}) \\
\frac{}{\text{natrec}(s(e); e_0; x.y.e_1) \mapsto [e, \text{natrec}(e; e_0; x.y.e_1)/x, y]e_1} \quad (\text{Rec-IS}) \\
\frac{}{\lambda x : \tau. e \text{ val}} \quad (\text{LAM-V}) \quad \frac{e_1 \mapsto e'_1}{e_1e_2 \mapsto e'_1e_2} \quad (\text{APP-S}) \quad \frac{(\lambda x : \tau. e)' \mapsto [e'/x]e'}{(\lambda x : \tau. e)'} \quad (\text{APP-I})
\end{align*}
\]