# CS520 Problem Set 7 Solution

29th November 2006

**Problem 1 (Lemma 0.2 in HD 14)** Every closed, non-value terms $t$ can be uniquely decomposed into $E_{cbv}$ such that $t = E_{cbv}[v_1\ v_2]$ or $t = E_{cbv}[v_1 + v_2]$.

*Proof.* Induction on cases of $t$ that are closed and are not values. The cases are $t = t_1\ t_2$ and $t = t_1 + t_2$. Notice that subterms $t_1$ and $t_2$ may be values. The base cases are when both subterms are values, which cannot be decomposed by themselves with $E_{cbv} \neq [\,]$.

- case $t = t_1\ t_2$,
  - subcase, both $t_1, t_2$ are values $v_1, v_2$, then let $E_{cbv} = [\,]$ and $t$ can be decomposed as $t = E_{cbv}[v_1\ v_2]$.
  - subcase, $t_1$ is a value $v_1$ but $t_2$ is not a value. We know $t$ is closed, so $t_2$ has to be closed. By I.H. on $t_2$, we get a decomposition $t_2 = E_{cbv2}[t_2']$. Then let $E_{cbv} = v_1\ E_{cbv2}$, and $t$ can be decomposed as $t = E_{cbv}[t_2']$.
  - subcase, $t_1$ is not a value. We know $t$ is closed, so $t_1$has to be closed. By I.H. on $t_1$, we get a decomposition $t_1 = E_{cbv1}[t_1']$. Then let $E_{cbv} = E_{cbv1}\ t_2$, and $t$ can be decomposed as $t = E_{cbv}[t_1']$.
- case $t = t_1 + t_2$, ditto.

Notice that in every subcases of $t$, only one form of evaluation context can apply, so decomposition of all subcases are unique.

□

The idea of this proof is to show there is exactly one redex for every evaluable (closed, non-value) term. This lemma is necessary so Definition 0.3 defines a deterministic one-step evaluation. The decomposition $t = E_{cbv}[t']$ puts redex in $t'$.

**Problem 2 (Exercise 0.3 in HD 15)** Given a $t \in \mathcal{L}$ and consider a context $C$ for $t$ such that $t = C[nf]$ for some $nf$ (either a $x$ or $v$).

1. If $t$ is closed, show that $nf$ is a value (cannot be $x$).
2. If $t$ is not closed, show that $nf$ is not necessarily a value (may be $x$).

*Proof.* In general, if $nf$ is not closed, then $C[nf]$ is not closed. Prove the more general statement using induction on the structure of $C$. □

**Problem 3** Is exception handling in HD 16 more general than Pierce §14.3, or the other way around?

| System | Simplified HD 16 ($\mathcal{L}^+$) | Pierce §14.3 ($\mathcal{L}^p$) |
|---|---|---|
| Definition | $t \quad ::= \quad v$<br>$t\ t$<br>`raise` $t\ t$<br>`handle` $t\ t\ t$ | $t \quad ::= \quad \ldots$<br>`raise` $t$<br>`try` $t$ `with` $t$ |
| Redex | `raise` $ex\ v$<br>`handle` $ex\ v\ t$ | `raise` $v$<br>`try` $t$ `with` $v$ |

Figure 0.1: Comparing simplified HD 16 ($\mathcal{L}^+$) and Pierce §14.3 ($\mathcal{L}^p$).

- To encode $\mathcal{L}^p$ in $\mathcal{L}^+$: simply fix $ex$ to a constant name throughout a term, so all exceptions have the same name, and they can carry arbitrary values.
- To encode $\mathcal{L}^+$ in $\mathcal{L}^p$: at first glance, $\mathcal{L}^p$ looks more restrictive because it doesn't distinguish exception names. However, item 4 in p. 177 explains how extensible variant type can be used. Each case of the variant type $T_{exn}$ (1) has a label that can encode exception name, and (2) carries a value. If `try` $t$ `with` $v$ captures an unintended exception, make $v$ raise it again.

Therefore, both systems have the same expressiveness in exception handling.