CS 336	**Programming Languages**	**Homework Solution 3**
Winter 2005		**Due 2/10/05**

1 Problem 1. (25 pts)

Modify the treatment of lists in Section 11.12 (Figure 11-13) to use a case expression in place of primitive functions head, tail, and isnil, and show that the Progress theorem holds for the result (in contrast to the original treatment, for which Progress does not hold (see the solution for Exercise 11.12.1)).

The modified syntax for list terms is:

$$\begin{array}{lll} t & ::= & \ldots \\ & & \texttt{nil}[T] \\ & & \texttt{cons}[T]\ t_1\ t_2 \\ & & \texttt{case}[T]\ t_1\ \texttt{of nil} => t_2 \mid \texttt{cons}\ x_1\ x_2 => t_3 \end{array}$$

The evaluation rules for the case expression are:

$$\texttt{case}[\mathtt{S}]\ \texttt{nil}[\mathtt{T}]\ \texttt{of nil} \ => \ \mathtt{t}_1 \mid \texttt{cons}\ \mathtt{x}_1\ \mathtt{x}_2\ => \ \mathtt{t}_2 \longrightarrow \mathtt{t}_1 \qquad \text{(E-CaseNil)}$$

$$\texttt{case}[\mathtt{S}]\ \texttt{cons}[\mathtt{T}]\ \mathtt{v}_1\ \mathtt{v}_2\ \texttt{of nil} \ => \ \mathtt{t}_1 \mid \texttt{cons}\ \mathtt{x}_1\ \mathtt{x}_2\ => \ \mathtt{t}_2 \longrightarrow \mathtt{t}_2 \qquad \text{(E-CaseCons)}$$

$$\frac{\mathtt{t} \longrightarrow \mathtt{t}'}{\texttt{case}[\mathtt{S}]\ \mathtt{t}\ \texttt{of nil} \ => \ \mathtt{t}_1 \mid \texttt{cons}\ \mathtt{x}_1\ \mathtt{x}_2\ => \ \mathtt{t}_2 \longrightarrow \texttt{case}[\mathtt{S}]\ \mathtt{t}'\ \texttt{of nil} \ => \ \mathtt{t}_1 \mid \texttt{cons}\ \mathtt{x}_1\ \mathtt{x}_2\ => \ \mathtt{t}_2} \qquad \text{(E-Case)}$$

The typing rule for the case expression is:

$$\frac{\Gamma \vdash \mathtt{t}_1 : \texttt{List}\ \mathtt{T}_1 \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T} \qquad \Gamma,\ \mathtt{x}_1 : \mathtt{T}_1,\ \mathtt{x}_1 : \texttt{List}\ \mathtt{T}_1 \vdash \mathtt{t}_3 : \mathtt{T}}{\Gamma \vdash \texttt{case}[\mathtt{T}_1]\ \mathtt{t}_1\ \texttt{of nil} \ => \ \mathtt{t}_2 \mid \texttt{cons}\ \mathtt{x}_1\ \mathtt{x}_2\ => \ \mathtt{t}_3 : \mathtt{T}} \qquad \text{(T-Case)}$$

The progress theorem states:

Theorem [Progress]: $\vdash \mathtt{t} : \mathtt{T} \ \Rightarrow\ \mathtt{t}$ is a value, or $\exists \mathtt{t}'.\, \mathtt{t} \longrightarrow \mathtt{t}'$.

Proof: We give just the new case for case expressions.

Case: $\mathtt{t} = \texttt{case}[\mathtt{T}_1]\ \mathtt{t}_1\ \texttt{of nil} \ => \ \mathtt{t}_2 \mid \texttt{cons}\ \mathtt{x}_1\ \mathtt{x}_2\ => \ \mathtt{t}_3$

We assume that the hypothesis $\vdash \mathtt{t} : \mathtt{T}$ holds. Then by the appropriately extended Inversion Lemma, we have

$(1)\quad \vdash \mathtt{t}_1 : \texttt{List}\ \mathtt{T}_1$

Then by the Induction Hypothesis, with have either

$(2a)\quad \mathtt{t}_1$ is a value, or

$(2b)\quad \mathtt{t}_1 \longrightarrow \mathtt{t}_1'$ for some $\mathtt{t}_1'$

If $(2a)$ is the case, then by the Canonical Forms lemma for the language with lists, either:

$(3a)\quad \mathtt{t}_1 \ = \ \texttt{nil}[\mathtt{T}_1]$ or

$(3b)\quad \mathtt{t}_1 \ = \ \texttt{cons}\ \mathtt{v}_1\ \mathtt{v}_2$ for some $\mathtt{v}_1$ and $\mathtt{v}_2$

If $(3a)$ is the case, then

$$(4a) \quad \texttt{t} \longrightarrow \texttt{t}_2 \text{ by (E-CaseNil)}$$

while if $(3b)$ is the case, then

$$(4b) \quad \texttt{t} \longrightarrow [\texttt{x}_1 \mapsto \texttt{v}_1,\, \texttt{x}_2 \mapsto \texttt{v}_2]\texttt{t}_3 \text{ by (E-CaseCons)}$$

If, on the other hand, $(2b)$ is true, then

$$(5) \quad \texttt{t} \longrightarrow \texttt{case}[\texttt{T}_1]\ \texttt{t}_1'\ \texttt{of nil} \ {=}{>}\ \texttt{t}_2 \mid \texttt{cons}\ \texttt{x}_1\ \texttt{x}_2\ {=}{>}\ \texttt{t}_3 \text{ by (E-Case)}$$

2 Problem 2. (15 pts)

Give a term whose evaluation does not terminate in the CBV lambda calculus with Nat, Bool, and Ref, but no fix operator.

Solution: The idea is to use a ref cell to create a function that calls itself recursively, each time on the same argument (or on an argument that is growing, rather than shrinking). I'll use a `let` syntax, which, as usual, abbreviates a lambda term applied to an argument. The following example is representative. It sets up a function that unconditionally calls itself on the same argument through the ref cell `r`

```
let r : Ref(Nat → Nat) = ref (λ n: Nat . n)
    in let f : Nat → Nat = λ x : Nat . ! r (x)
       in let u : Unit = r := f
          in f 0
```

3 Problem 3. (25 pts)

Do the inductive case for $\texttt{t} = \texttt{ref t1}$ in the proof of 13.5.3.

Solution: We assume that $\texttt{t} = \texttt{ref}\ \texttt{t}_1$ and that the hypotheses of the Theorem hold:

$$(1) \quad \Gamma|\Sigma \vdash \texttt{t} : \texttt{T}$$

$$(2) \quad \Gamma|\Sigma \vdash \mu$$

$$(3) \quad \texttt{t}|\mu \rightarrow \texttt{t}'|\mu'$$

By Inversion on (1), there exists a type $\texttt{T}_1$ such that

$$(4) \quad \texttt{T} = \texttt{Ref}\ \texttt{T}_1$$

$$(5) \quad \Gamma|\Sigma \vdash \texttt{t}_1 : \texttt{T}_1$$

There are two cases, according to the rule justifying (1).

Case: (3) by (E-REFL).
Here we know that $\texttt{t}_1$ is a value $\texttt{v}_1$, and we have

$$(6) \quad \texttt{ref}\ \texttt{v}_1 |\, \mu \rightarrow l \,|\, (\mu, l \mapsto \texttt{v}_1)$$

where l is a new location (not in the domains of Σ and μ), and $\texttt{t}' = l$. Let $\mu' = (\mu, l \mapsto \texttt{v}_1)$ and $\Sigma' = \Sigma, l : \texttt{T}_1$. Then

$$(7) \quad \Gamma|\Sigma' \vdash l : \texttt{Ref}\ \texttt{T}_1 \quad \text{by (T-LOC)}$$

and hence by (4)

$$(8) \quad \Gamma|\Sigma' \vdash \mathtt{t}' : \mathtt{T}$$

Finally, by (2) and the definition of Σ', we have

$$(9) \quad \Gamma|\Sigma' \vdash \mu' \quad \text{by (2) and the defn of } \Sigma'$$

Case: (3) by (E-REF).
Here there exists a term $\mathtt{t}_1'$ such that $\mathtt{t}' = \mathtt{ref}\ \mathtt{t}_1'$, and we have

$$(10) \quad \mathtt{ref}\ \mathtt{v}_1 |\, \mu \rightarrow \mathtt{ref}\ \mathtt{t}_1' \,|\, \mu'$$

where

$$(11) \quad \mathtt{t}_1 |\, \mu \rightarrow \mathtt{t}_1' \,|\, \mu'$$

Then by the Induction Hypothesis, there exists $\Sigma' \supseteq \Sigma$ such that

$$(12) \quad \Gamma|\Sigma' \vdash \mathtt{t}_1' : \mathtt{T}_1$$

$$(13) \quad \Gamma|\Sigma' \vdash \mu'$$

Then by (12) and the typing rule (T-REF) we have

$$(14) \quad \Gamma|\Sigma' \vdash \mathtt{t}' : \mathtt{T}$$

4 Problem 4. (25 pts)

Do the inductive case for $\emptyset|\Sigma \vdash \mathtt{t} : \mathtt{Ref\ T}$ in the proof of 13.5.7. [**Note:** This problem is ill-posed. The case structure for the proof will be based on the typing *rules*, so for the type Ref T there will actually be two cases, for the rules (T-LOC) and (T-REF).]

Solution: We do the two cases associated with (T-LOC) and (T-REF).

Case: $\emptyset|\Sigma \vdash l : \mathtt{Ref\ T}$ by (T-LOC).
Then $\mathtt{t} = l$, a value, and we are done.

Case: $\emptyset|\Sigma \vdash \mathtt{ref}\ \mathtt{t}_1 : \mathtt{Ref}\ \mathtt{T}_1$ by (T-REF).
Then by the Inversion Lemma, we have

$$(1) \quad \emptyset|\Sigma \vdash \mathtt{t}_1 : \mathtt{T}_1$$

Then by the induction hypothesis, either

(2) $\mathtt{t}_1$ is a value, v or

$$(3) \quad \forall \mu.\, \emptyset|\Sigma \vdash \mu \Rightarrow \exists \mathtt{t}_1'.\, \exists \mu'.\, \mathtt{t}_1 |\, \mu \rightarrow \mathtt{t}_1' |\, \mu'$$

Suppose (2) is the case, and that μ is a store such that $\emptyset|\Sigma \vdash \mu$. Then by (E-REFV), we will have

$$(4) \quad \mathtt{ref}\ \mathtt{v}_1 |\, \mu \rightarrow l \,|\, (\mu, l \mapsto \mathtt{v}_1)$$

where l is a new location. So, taking $\mathtt{t}' = l$ and $\mu' = (\mu, l \mapsto \mathtt{v}_1)$, we have

$$(5) \quad \mathtt{t} |\, \mu \rightarrow \mathtt{t}' \,|\, \mu'$$

If, on the other hand, $\mathtt{t}_1$ is not a value and (3) holds, then for any store μ such that $\emptyset|\Sigma \vdash \mu$ there exists a term $\mathtt{t}_1'$ such that

$$\mathtt{t}_1 |\, \mu \rightarrow \mathtt{t}_1' |\, \mu'$$

and therefore by rule (E-REF)

$$(5) \quad \mathtt{ref}\ \mathtt{t}_1 |\, \mu \rightarrow \mathtt{ref}\ \mathtt{t}_1' \,|\, \mu'$$

5 Problem 5. (15 pts)

Do Exercise 15.5.2 (page 198).

Solution: Part (1), a program that will be well-typed if `Ref` is contravariant (the first premise is dropped):

```
let r : Ref {a: Nat} = ref {a = 1}
    in (!r).b
```

Here the body expression type-checks under the assumption that

$$\texttt{Ref}\{\texttt{a} : \texttt{Nat}\} <: \texttt{Ref}\{\texttt{a} : \texttt{Nat}, \texttt{b} : \texttt{Nat}\}$$

which is a consequence of `Ref` being contravariant.

Part (2), a program that will be well-typed if `Ref` is covariant (the second premise is dropped):

```
let r : Ref {a: Nat, b: Nat} = ref {a = 1, b = 2}
    in _ : Unit = (r := {a = 1})
       in (!r).b
```

Here the second line type checks if `Ref` is covariant and hence $r : Ref\{a : Nat\}$. The third line then leads to a stuck state when we attempt to project the `b` field of the record value $\{\texttt{a} = 1\}$.