CS411 Probs 3

A. Demers

27 Feb 2001 – due 6 Mar 2001

These problems deal with the following tiny language fragment LiTL:

$$e \ ::= \ n \mid (e_1 \theta e_2) \mid (\textbf{let } x \sim e_1 \textbf{ in } e_2) \mid x$$

LiTL includes arithmetic expressions and **let** blocks and little else. In particular there are no program variables or assignments, and no nonterminating constructs.

Question 1 [8 points] Give definitions by recursion on LiTL expressions for each of the following:

(a) $FV(e_1)$ – the free variables of expression e_1.

(b) $[e_2/x](e_1)$ – substitution of e_2 for x into e_1 with capture allowed.

(c) $[e_2//x](e_1)$ – "safe" (capture-avoiding) substitution of e_2 for x into e_1.

Before answering part (c) you should look at Question 2 – it may affect your answer.

Question 2 [10 points] Recall an expression is said to be *closed* if it has no free variables. Prove that the two notions of substitution defined in Question 1 coincide when the expression e_1 being substituted is closed. That is, prove

$$(FV(e_1) = \emptyset) \quad \Rightarrow \quad (\forall x, e_2)([e_1/x](e_2) \equiv [e_1//x](e_2))$$

This result should be intuitively clear – without free variables, capture cannot happen anyway, so it shouldn't matter whether we try to avoid it.

Discussion: In the remaining questions we'll explore eager and lazy evaluation rules for LiTL using a slightly different approach from the notes.

First we define an environment that is suitable for any of the evaluation strategies. That is, an environment ϕ is a finite set

$$\phi \quad = \quad \{\ldots\ x_i \sim \langle n_i, e_i, \phi_i \rangle\ \ldots\}$$

of bindings of names to both numbers (i.e., evaluated expressions) and expression - environment pairs (i.e. unevaluated expressions).

The intuition is that we can simultaneously keep track of both the lazy and eager versions of the environment, and prove something about how they are related in any derivation.

Because LiTL has no assignments, there is no need for a store in the evaluation relation; thus it has the form

$$\langle e, \phi \rangle \rightarrow_E n$$

The subscript "E" on $\rightarrow_E$ indicates "eager." Later we'll discuss "lazy dynamic" ($\rightarrow_D$) and "lazy static" ($\rightarrow_S$) evaluation relations as well.

Keeping this in mind, here are eager evaluation rules for LiTL:

$$\frac{}{\langle n, \phi \rangle \rightarrow_E n} \tag{E1}$$

$$\frac{\langle e_1, \phi \rangle \rightarrow_E n_1 \qquad \langle e_2, \phi \rangle \rightarrow_E n_2}{\langle (e_1 \theta e_2), \phi \rangle \rightarrow_E n} \qquad \text{where } n = n_1 \theta n_2 \tag{E2}$$

$$\frac{\langle e_1, \phi \rangle \rightarrow_E m \qquad \langle e_2, (\phi \oplus \{x \sim \langle m, e_1, \phi \rangle\}) \rangle \rightarrow_E n}{\langle (\textbf{let } x \sim e_1 \textbf{ in } e_2), \phi \rangle \rightarrow_E n} \tag{E3}$$

$$\frac{}{\langle x, \phi \rangle \rightarrow_E n} \qquad \text{where } (x \sim \langle n, e', \phi' \rangle) \in \phi \tag{E4}$$

These are straightforward eager evaluation rules except for the fact that in rule E3 we insert a copy of the unevaluated expression e_1 and its associated environment ϕ into the environment of the hypothesis, even though none of the other rules ever refers to this information. Clearly this carrying around of "syntactic history" has no effect on the derivability relation.

(End of Discussion)

Question 3 [8 points] We would like to show that the eager rules obey the following *Completeness Property* for closed expressions: If e is closed, then

$$(FV(e) = \emptyset) \quad \Rightarrow \quad (\exists n)(\forall \phi)(\langle e, \phi \rangle \rightarrow n)$$

This is most easily proved by proving the following somewhat stronger property:

Proposition (completeness of eager rules):

$$\mathbf{dom}(\phi) \supseteq FV(e) \quad \Rightarrow \quad (\exists n)\langle e, \phi \rangle \rightarrow_E n$$

This can be proved by induction on the structure of LiTL expressions, with a case for each constructor. Give the proof.

Question 4 [8 points] It is possible to change the above rules to give either lazy dynamic scope behavior, defining an evaluation relation $\rightarrow_D$, or lazy static scope behavior, defining an evaluation relation $\rightarrow_S$. This can be done by changing *only* the identifier-reference rule (E4). Give revised rules (D4) and (S4) that accomplish this.

Question 5 [6 points] Show by example that the dynamic (Dx) and static (Sx) rules disagree (even in this tiny language from which nontermination and assignment have been omitted). That is, give an expression e such that

$$\langle e, \{\} \rangle \rightarrow_D n \qquad \text{and} \qquad \langle e, \{\} \rangle \rightarrow_S m$$

where $m \neq n$. Note $\{\}$ is the initial empty environment, a.k.a. the empty set.

Question 6 [10 points] We define an environment ϕ to be *consistent* if

$$(x \sim \langle n, e', \phi' \rangle) \in \phi \quad \Rightarrow \quad \langle e', \phi' \rangle \rightarrow_E n$$

that is, the evaluated and unevaluated versions of the bindings in ϕ agree.

Use this definition, and the fact that an empty environment is consistent, to prove that the eager and static rules agree; that is,

$$\langle e, \{\} \rangle \rightarrow_E n \quad \Rightarrow \quad \langle e, \{\} \rangle \rightarrow_S n$$

for any closed expression e.