Homework 3: Typed Lambda Calculus and Terminal Transition Systems

This homework is due Friday, March 1 (in class).

1. (10 points) For each of the following terms in the simply typed lambda calculus: (a) give the type and (b) a formal proof that the term has that type.

   (a) $\lambda f : \mathbf{o} \to \mathbf{o} \,.\, \lambda x : \mathbf{o} \,.\, f(x)$

   (b) $\lambda f : \mathbf{o} \to \mathbf{o} \,.\, \lambda g : \mathbf{o} \to \mathbf{o} \,.\, \lambda x : \mathbf{o} \,.\, f(g(x))$

2. (5 points) Is the following a term of the simply typed lambda calculus?

   $$\lambda f : \mathbf{o} \to \mathbf{o} \,.\, f(f)$$

   Why? (Give a precise, technical reason.)

3. (extra credit only) Modify the interpreter for the simply typed lambda calculus so that the interpreter can find a proof that a term has a given type, along with the its type. That is, define `type-of` as

   ```
   (define type-of (lambda (e eta) (second (proof-of-type (e eta)))))
   ```

   where `proof-of-type` returns a list consisting of a proof tree and a type. A proof tree has as its root a type judgement, and the result of `(proof-of-type e eta)` should be a list where the second element is the type from the root of the proof tree in the first element.

4. (15 points) Give a terminal transition system for a deterministic Turing machine with a one-way tape. You should include a description of the set of configurations (Γ), the set of final configurations (T), the axioms and inference rules that define the transition relation ($\Rightarrow$), and both the input ($\mathcal{I}$) and output ($\mathcal{O}$) functions.

   A Turing machine consists of a finite state control, which can be thought of as a partial function δ that takes a state (from a set of states K) and a character read from the tape and returns a new state, a character to write on the tape, and a direction to move (right or left). The machine starts out with an initial tape (a string of characters from some alphabet Σ), its read head positioned at the leftmost character of the tape, and its finite state control in the state q_0. For each move, the machine reads a character from the tape at the current position, and then depending on the current state, writes a character at the current position, moves left or right one character on the tape, and goes to into a new state (as dictated by δ). If δ is not defined for the current state and read character, or if an attempt is made to move off the left end of the tape, then the machine halts.

5. (30 points) This exercise asks you to implement a simple form of terminal transition systems in Scheme. The simplifications are that your program will not have to handle pattern matching or inference rules.

To be specific you are to implement a function `tts` that takes as an argument a list of transition rules of the form `(==>` E_1 E_2`)`, where E_1 and E_2 are S-expressions. That is, S-expressions are used for input (and output) of configurations. The result of an application of `tts` should be a function that, when applied to an S-expression (encoding a configuration) returns a S-expression encoding a final configuration, if there is one. A final configuration is a configuration to which no rules apply. Use `equal?` to test whether a transition rule applies to a given configuration.

For example, the result of the following expression should be `Z`.

```
(let ((my-system (tts '((==> x y) (==> y z)))))
  (my-system 'x))
```

Note that transition systems need not be deterministic, nor must they always be terminating.

6. (extra credit only) Add inference rules and/or pattern matching to your terminal transition system implementation.

7. (10 points) What are the limitations of terminal transition systems as a semantic description technique?