# Assignment 5:
# A Monad for I/O

15-312: Foundations of Programming Languages
Daniel Spoonhower (spoons@cs.cmu.edu)

Out: Thursday, October 16, 2003
Due: Thursday, October 23, 2003 (1:30 pm)

50 points total

In this assignment, we add simple input and output to the generic framework of monads. Both input and output are represented as potentially infinite streams of integers $n_1 \cdot n_2 \cdot \cdots$, where $\epsilon$ is the empty stream.

The worlds of the monadic framework are therefore pairs $(S_I, S_O)$ of potentially infinite streams of integers, where $S_I$ represents the input stream, and $S_O$ is the output stream which is initially empty. We have the following new monadic (that is, effectful) expressions.

- `read` $\div$ `int` which reads and thereby consumes an integer from the input stream. It returns $0$ if the input stream is empty.
- `eof` $\div$ `bool` which returns `true` if the input stream is empty and `false` otherwise.
- `write`$(e) \div 1$ which writes the value of $e$ (which must be an integer) to the output stream.

### 1. Typing rules (5 pts)

Give the rules for typing the new expressions (`read`, `eof`, `write`). You do not need to repeat the generic rules for the monad.

## 2. Operational semantics (10 pts)

Present the new transition rules for the structured operational semantics. According to the monadic framework, the transitions should have one of the two forms

$$\begin{array}{rcl} \langle (S_I, S_O), m \rangle & \mapsto & \langle (S_I', S_O'), m' \rangle \\ e & \mapsto & e' \end{array}$$

## 3. Stating the progress theorem (5 pts)

Carefully formulate the progress theorem which is appropriate for the setting above. You may assume the input and output streams are well-formed.

## 4. Proving the progress theorem (15 pts)

Prove the progress theorem. Show all the cases concerned with the monadic constructs: `val`, `let val`, `read`, `eof` and `write`. If you need a value inversion property (also known as the canonical forms property), please state it explicitly, but you don't need to prove it. If you need to generalize the progress theorem from question 3, please explicitly state the generalization.

## 5. Non-recursive programming (5 pts)

Define `copyOne` : $\bigcirc 1$ which reads one integer from the input stream and writes it to the output stream. Be sure that your program typechecks according to the generic rules for the monad and your typing rules above.

## 6. Recursive programming (10 pts)

Recall that our recursive binding construct $\texttt{rec}(\tau, x.e)$ allows us to write recursive expressions of arbitrary type $\tau$. Using this construct, define

$$\texttt{copy} : \bigcirc 1$$

which reads the entire input stream and copies it to the output stream. (Again, we recommend that you verify that your program is well-typed.)