Problem Set #8

due November 26

Problem 1. Rewrite `inferTypes` from PS7 to use a state transformer instead of a "stateful" variable. Signature changes from

```
inferTypes :: TEnv -> Integer -> Exp -> (Subst, Type, Integer)
```

to

```
inferTypes :: TEnv -> Exp -> S.State Integer (Subst, Type)
```

Transfer your previous homework to file `Infer.hs`. The only change should be `inferTypes` and callers of `inferTypes`.

Problem 2. Next, write a parser for the lambda calculus following this grammar:

$$
\begin{array}{lcl}
\textit{lexp} & ::= & \textit{lapp} \mid \textit{labs} \mid \textit{lterm} \\
\textit{lapp} & ::= & \textit{lterm}\ \textit{lterm}\ \textit{ttail} \\
\textit{ttail} & ::= & \textit{lterm}\ \textit{ttail} \mid \epsilon \\
\textit{labs} & ::= & \lambda\ \texttt{identifier ->}\ \textit{lexp} \\
\textit{lterm} & ::= & \texttt{(}\ \textit{lexp}\ \texttt{)} \mid \texttt{identifier} \mid \texttt{integer} \mid \texttt{T} \mid \texttt{F}
\end{array}
$$

The function parses a lambda expression into the `Exp` datatype from PS7 and `Data.hs` is included in the starter code (again).

```
lexp :: P.Parser Exp
lexp = undefined

> P.runParser (start lexp) "\\f -> \\x -> f (f x)$"
[(ELambda "f" (ELambda "x" (EApp (EVar "f") (EApp (EVar "f") (EVar "x")))),"")]
```

We can create expressions (by parsing them from text) and infer types a lot easier:

```
> twice = fst $ head $ P.runParser (start lexp) "\\f -> \\x -> f (f x)$"
> I.canonicalize twice -- from PS7, now in Infer.hs
"(t1 -> t1) -> t1 -> t1"
```

Problem 3. Finally, extend the parser with handling of ambiguous grammars in the sense that the parser now finds *all* possible parse trees for a string. I will test with the following grammar that generates all strings of equal number of `a`'s and `b`'s:

$$s \ ::= \ \texttt{a}\ s\ \texttt{b}\ s \mid \texttt{b}\ s\ \texttt{a}\ s \mid \epsilon$$

The output of a parse of s should be the production sequence the parser finds where productions are numbered from left to right: $s ::= \texttt{a}\ s\ \texttt{b}\ s$ is 1, $s ::= \texttt{b}\ s\ \texttt{a}\ s$ is 2, and $s ::= \epsilon$ is 3. Naturally, the autograder expects that your parser respects this order when trying alternatives.

```
s :: P.Parser [Int]
s = undefined

> P.runParser (start s) "abab$"
[([1,2,3,3,3],""),([1,3,1,3,3],"")]
```

E.g., `[1,2,3,3,3]` is leftmost derivation $s \overset{1}{\Rightarrow}$ `a`s`b`$s \overset{2}{\Rightarrow}$ `ab`s`a`s`b`$s \overset{3}{\Rightarrow}$ `aba`s`b`$s \overset{3}{\Rightarrow}$ `abab`$s \overset{3}{\Rightarrow}$ `abab`.

Note: Download files `Data.hs`, `State.hs`, `Lexer.hs`, `Parser.hs` and `Infer.hs`. Minimal starter code is in `Ps8.hs`.