Functional Programming – Series 7

In this series of exercises we will write a *type checker* for expressions.

Exercise 1. Given are embedded languages for *expressions* and for *types*:

$$
\begin{aligned}
\textbf{data}\ \mathit{Expr} = {} & \mathit{Const}\ \mathit{Int} \\
\mid {} & \mathit{Var}\ \mathit{String} \\
\mid {} & \mathit{BinOp}\ \mathit{String}\ \mathit{Expr}\ \mathit{Expr} \\
\mid {} & \mathit{App}\ \mathit{Expr}\ \mathit{Expr}
\end{aligned}
$$

$$
\begin{aligned}
\textbf{data}\ \mathit{Type} = {} & \mathit{IntType} \\
\mid {} & \mathit{FunType}\ \mathit{Type}\ \mathit{Type}
\end{aligned}
$$

The meaning of these constructions is as follows:

$\mathit{Const}\ n$	the constant n
$\mathit{Var}\ x$	the variable x
$\mathit{BinOp}\ \text{“+”}\ e_0\ e_1$	the expression $e_0 + e_1$
$\mathit{App}\ f\ e$	the expression $f\ e$,
	i.e., *App* is for function application
$\mathit{IntType}$	for type *Int*
$\mathit{FunType}\ t_0\ t_1$	for type $t_0 \to t_1$

Define a function

$$\mathit{typeOf} :: \mathit{Env} \to \mathit{Expr} \to \mathit{Type}$$

that determines the type of an expression given an *environment*, i.e., given the types of elementary variables and functions, defined as follows:

$$\textbf{type}\ \mathit{Env} = [(\mathit{String}, \mathit{Type})]$$

The environment should contain types for at least the operations $+$, $*$, $-$, where these operations are indicated as strings “+”, etcetera.

Hint: Haskell has a function *lookup* (note that the result type is a *Maybe* value):

$$\mathit{lookup} :: \mathit{Eq}\ a \Rightarrow a \Rightarrow [(a, b)] \Rightarrow \mathit{Maybe}\ b$$

Exercise 2. Extend the embedded languages above with an expression for *if-the-else*, and adapt the types where approprite. The environment now should contain operations && and || for booleans as well.

Exercise 3. Extend the expressions and types with 2-tuples and 3-tuples.

Exercise 4. Add lambda expressions to your language, and extend the function *typeOf* for them. A lambda expression contains the types of the formal parameters, in the following form:

$$\backslash x :: a \rightarrow expr$$

where a is a type. Thus, such lambda expressions are slightly different from lambda expressions in Haskell: the formal parameter x is explicitly annotated with its type. Thus, if t is the type of *expr*, then the type of this lambda term should be

$$FunType\ a\ t$$

Note that you'll have to choose an appropriate constructor to represent lambda expressions.

Remark. Note that above no *type variables* are requested. That would require *substitution* and *unification*, which is at this moment not requested.