Assignment 5
Mutual Recursion

15-814: Types and Programming Languages
Frank Pfenning

Due Tuesday, October 15, 2019

1 Mutually Recursive Types

Task 1 (L10.1, 30 points) It is often intuitive to define types in a mutually recursive way. As a simple example, consider how to define binary numbers in *standard form*, that is, not allowing leading zeros. We define binary numbers in standard form (*std*) mutually recursively with strictly positive binary numbers (*pos*).

$$\begin{array}{lcl} \mathit{std} & \cong & (\mathtt{E} : 1) + (\mathtt{B0} : \mathit{pos}) + (\mathtt{B1} : \mathit{std}) \\ \mathit{pos} & \cong & (\mathtt{B0} : \mathit{pos}) + (\mathtt{B1} : \mathit{std}) \end{array}$$

1. Using only *std*, *pos*, and function types formed from them, give all types of *E*, *B0*, and *B1* defined as follows:

$$\begin{array}{lcl} E & = & \mathsf{fold}\ (\mathtt{E} \cdot \langle\rangle) \\ B0 & = & \lambda x.\, \mathsf{fold}\ (\mathtt{B0} \cdot x) \\ B1 & = & \lambda x.\, \mathsf{fold}\ (\mathtt{B1} \cdot x) \end{array}$$

2. Define the types *std* and *pos* explicitly in our language using the ρ type former so that the isomorphisms stated above hold.

3. Does the function *inc* from Section L10.6 have type $\mathit{std} \to \mathit{pos}$? Rewrite it in the syntax from Section L10.4, where you may use the function *Unfold* (defined in that section). Then either explain where the typing fails or indicate that it has that type. You do not need to write out a typing derivation.

4. Write a function $\mathit{pred} : \mathit{pos} \to \mathit{std}$ that returns the predecessor of a strictly positive binary number. You may use pattern matching to define your function, but you must make sure it is correctly typed.

2 Mutually Recursive Functions

Task 2 (L10.2, 30 points) It is often convenient to define functions by mutual recursion. As a simple example, consider the following two functions on bit strings determining if it has *even or odd parity*.

$$
\begin{array}{lcl}
\mathit{bin} & \cong & (\mathtt{E} : 1) + (\mathtt{B0} : \mathit{bin}) + (\mathtt{B1} : \mathit{bin}) \\
\mathit{even} & : & \mathit{bin} \to \mathit{bool} \\
\mathit{odd} & : & \mathit{bin} \to \mathit{bool} \\
\mathit{even}\ E & = & \mathit{true} \\
\mathit{even}\ (\mathit{B0}\ x) & = & \mathit{even}\ x \\
\mathit{even}\ (\mathit{B1}\ x) & = & \mathit{odd}\ x \\
\mathit{odd}\ E & = & \mathit{false} \\
\mathit{odd}\ (\mathit{B0}\ x) & = & \mathit{odd}\ x \\
\mathit{odd}\ (\mathit{B1}\ x) & = & \mathit{even}\ x
\end{array}
$$

1. Write a function *parity* with a single fixed point constructor and use it to define *even* and *odd*. You may use pattern matching, but the pattern of recursion (and the fact you only need one fixed point) should be clear. Also, state the type of your *parity* function explicitly.

2. More generally, our simple recipe for implementing a recursively specified function using the fixed point constructor in our call-by-value language goes from the specification

$$
\begin{array}{lcl}
f & : & \tau_1 \to \tau_2 \\
f\,x & = & h\,f\,x
\end{array}
$$

   to the implementation

$$
f \quad = \quad \mathsf{fix}\ g.\ \lambda x.\ h\,g\,x
$$

   It is easy to misread these, so remember that by our syntactic convention, $h\,f\,x$ stands for $(h\,f)\,x$ and similarly for $h\,g\,x$. Give the type of h and show by calculation that f satisfies the given specification by considering $f\,v$ for an arbitrary value v of type τ_1.

3. A more general, mutually recursive specification would be

$$
\begin{array}{lcl}
f & : & \tau_1 \to \tau_2 \\
g & : & \sigma_1 \to \sigma_2 \\
f\,x & = & h_1\,f\,g\,x \\
g\,y & = & h_2\,f\,g\,y
\end{array}
$$

   Give the types of h_1 and h_2.

4. Show how to explicitly define f and g in our language from h_1 and h_2 using the fixed point constructor and verify its correctness by calculation as in part 2. You may use any other types in the language introduced so far (pairs, unit, sums, and recursive types).