Constructive Logic
Fall 2000

Carnegie Mellon University
Frank Pfenning

Assignment 4

Out: Friday Sep 29

Due: Thursday Oct 5

1. Primitive Recursion over nat (30 Points)

For each of the following three functions give first a **specification** and then an **implementing term**. Follow the example of *double* given in the lecture notes. You may freely reuse the functions from the lecture notes and define your own auxiliary functions.

- $power2 : \mathbf{nat} \rightarrow \mathbf{nat}$. For $n \in \mathbf{nat}$ the term $power2\ n$ should compute 2^n.
- $power : \mathbf{nat} \rightarrow \mathbf{nat} \rightarrow \mathbf{nat}$. For $n, m \in \mathbf{nat}$ the application $power\ n\ m$ should reduce to n^m.
- $fib : \mathbf{nat} \rightarrow \mathbf{nat}$. For $n \in \mathbf{nat}$ the term $fib\ n$ computes the nth Fibonacci number, where $fib\ 0 = 0$, $fib\ 1 = 1$, $fib\ 2 = 1$, $\ldots 2, 3, 5, 8, 13$, etc. In this sequence, every number except the first two is the sum of the two preceding numbers
  Hint: You will need a hack similar to the one in lecture.

2. Primitive Recursion over list (30 Points)

Again, give specifications and implementations for the following functions.

- $filter : (\tau \rightarrow \mathbf{bool}) \rightarrow \tau\,\mathbf{list} \rightarrow \tau\,\mathbf{list}$. For $p \in \tau \rightarrow \mathbf{bool}$, $l \in \tau\,\mathbf{list}$ the call $filter\ p\ l$ returns a sublist l' of l which contains only those elements $x \in \tau$ for which $p\ x$ returns **true**.
- $exists : (\tau \rightarrow \mathbf{bool}) \rightarrow \tau\,\mathbf{list} \rightarrow \mathbf{bool}$. For $p \in \tau \rightarrow \mathbf{bool}$, $l \in \tau\,\mathbf{list}$ the result of $exists\ p\ l$ should be **true** if $p\ x$ returns **true** for any list element $x \in \tau$, otherwise **false**.
- $nth : \mathbf{nat} \rightarrow \tau\,\mathbf{list} \rightarrow \tau \rightarrow \tau$. For $n \in \mathbf{nat}$, $l \in \tau\,\mathbf{list}$ and $a \in \tau$ the call $nth\ n\ l\ a$ should return the nth element of the list l, where we start counting in the head with 0. The value a should be returned in any exceptional case.

3. Encoding of bool (20 Points)

In the lecture the type constructors $\rightarrow$, $\times$, $\mathbf{1}$ and $\mathbf{0}$ were introduced, which are isomorphic to implication, conjunction, truth and falsehood. Here we complete the picture giving the *sum type* constructor '+' which is isomorphic to disjunction. The rules are:

– Formation:

$$\frac{\sigma\ \mathit{type} \qquad \tau\ \mathit{type}}{\sigma + \tau\ \mathit{type}}\ +\mathrm{F}$$

– Introduction:

$$\frac{\Gamma \vdash t \in \sigma}{\Gamma \vdash \mathbf{inl}\, t \in \sigma + \tau}\ +\mathrm{I_L} \qquad \frac{\Gamma \vdash t \in \tau}{\Gamma \vdash \mathbf{inr}\, t \in \sigma + \tau}\ +\mathrm{I_R}$$

– Elimination:

$$\frac{\Gamma \vdash r \in \sigma + \tau \qquad \Gamma, x \in \sigma \vdash s \in \rho \qquad \Gamma, y \in \tau \vdash t \in \rho}{\Gamma \vdash \mathbf{case}\, r\, \mathbf{of}\, \mathbf{inl}\, x \Rightarrow s \mid \mathbf{inr}\, y \Rightarrow t : \rho}\ +\mathrm{E}$$

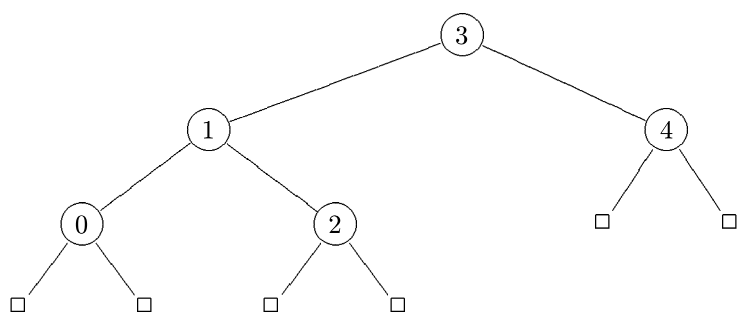
Now we can *define* a type of booleans as a two-element set: $\mathit{Bool} = \mathbf{1} + \mathbf{1}$. Convince yourself that the type Bool has exactly 2 normal elements and define:

- The truth values $\mathit{tt}, \mathit{ff} \in \mathit{Bool}$.
- A term $\mathit{ifThenElse} : \mathit{Bool} \to \tau \to \tau \to \tau$. For $b \in \mathit{Bool}$ and $s, t \in \tau$ the call $\mathit{ifThenElse}\ b\ s\ t$ should return s if $b = \mathit{tt}$ and t otherwise.
- A term $\mathit{xor} : \mathit{Bool} \to \mathit{Bool} \to \mathit{Bool}$ that implements exlusive-or.

Again, give specifications and implementations.

4. Binary Trees (20 Points)

In the same manner as natural numbers and lists, we want to introduce *labelled complete binary trees* as an inductive datatype. Each interior node carries a label and has exactly two child nodes. Each leaf has neither a label nor a child node. Here is an example of a tree carrying natural numbers:



- Give formation, introduction and elimination rules for the data type $\tau\, \mathbf{tree}$ which should have the two constructors **leaf** and **node**.
- Give a specification and an implementation for each of the functions $\mathit{count} : \tau\, \mathbf{tree} \to \mathbf{nat}$ and $\mathit{traverse} : \tau\, \mathbf{tree} \to \tau\, \mathbf{list}$. The function *count* counts the number of labels in the given tree (that is 5 in our example) and *traverse* sequentializes all labels into a list such that the leftmost is first and the rightmost is last ([0,1,2,3,4] in our example). Reuse functions defined in the lecture.

Good luck!