

---

CMSC 22100/32100: Programming Languages

Homework 8

M. Blume

Due: November 25, 2008

---

1. Assuming a type `nat`, the following ML datatype lets us represent binary trees that carry natural numbers at their leaves:

```
datatype tree = Leaf of nat | Branch of tree * tree
```

- (a) Devise an encoding of such a type in the polymorphic  $\lambda$ -calculus. (Your solution should be inspired by the encoding of `list` as shown in Section 6.4 of the Lecture Notes.)
  - (b) Let us call the type that is the answer to the previous question `tree`. Show the corresponding  $\lambda$ -terms for the constructors `Leaf` and `Branch`. (`Leaf` should have type `nat  $\rightarrow$  tree`, and `Branch` should have type `tree  $\rightarrow$  tree  $\rightarrow$  tree`.)
  - (c) Show a  $\lambda$ -term (call it `sumtree`) of type `tree  $\rightarrow$  nat` that, when applied to an instance of a `tree` computes the sum of its leaves.
2. Using the encoding for `nat` shown in Section 6.4 of the Lecture Notes, define a  $\lambda$ -term (call it `fac`) of type `nat  $\rightarrow$  nat` that computes the factorial of its argument.

For this, remember that the version of the Polymorphic  $\lambda$ -calculus that we are using does not have general recursion: there is no `letrec` or `fun`, and there are no recursive types. However, computing the factorial of  $n$  does not require general recursion. It is sufficient to *iterate* a certain function  $n$  times. The Church-encoding of a natural number  $n$  represents precisely the ability of iterating *any* function whose argument- and result types match  $n$  times.

Therefore, the task is to find a suitable function to be iterated, to find the appropriate initial argument for it, and to find a way of extracting the factorial from its result.

*Hint:* You probably want to use other encoded types as part of your solution. In particular, product types (*a.k.a.* pairs) look promising!

3. (*extra credit*) Explain how function `pred` as shown in the subsection on “Naturals” in Section 6.4 of the Lecture Notes works.

Hints: The basic idea is to have a “nonstandard” representation of the integers greater than or equal to  $-1$  ( $\{-1, 0, 1, 2, 3, 4, \dots\}$ ) together with a “nonstandard” version of the successor (called *psucc*) function that works on those numbers. Your solution should involve finding answers to all of the following questions:

- What is the representation of  $-1$ ?
- What is the representation of  $n$  where  $n \geq 0$ ?
- How does *psucc* perform its job?
- How is *psucc* being used within the definition of **pred**?