Programming Languages — Homework 8
Polymorphism

Due: Wednesday, 22 May 2013, 23:55

This assignment is graded out of 10 points.

1. [3 pts] In the simply-typed lambda calculus, the term $\lambda x.\ x\ x$ cannot be typed; that is, there is no type τ such that $\lambda x : \tau.\ x\ x$ will type-check.

   However, in System F, we can type a version of this term:

   $$\lambda x : \forall\alpha.\ \alpha \to \alpha.\ x[\forall\alpha.\ \alpha \to \alpha]\ x$$

   Show that this term is typeable by providing a typing derivation. The typing rules are on page 343 of Pierce.

2. [5 pts] Parametricity allows us to state some facts that must be true for a value of a given type. For instance, a value of type $\forall\alpha.\ \alpha \to \alpha$ either must be the identity function or it must be a function that does not terminate.

   For each of the following types, write a few sentences (at most) describing all possible values of the type, or state if there are no values of that type.

   We use Haskell notation for types, e.g., we use square brackets, $[\cdot]$, for lists. Assume functions have no side-effects (e.g., I/O) except, possibly, non-termination.

   (a) $\texttt{Int} \to \texttt{Int}$
   (b) $\forall\alpha.\ \alpha$
   (c) $\forall\alpha.\ [\alpha] \to \texttt{Bool}$
   (d) $\forall\alpha.\ \texttt{Int} \to \alpha$
   (e) $\forall\alpha.\ \forall\beta.\ \alpha \to \beta \to \alpha$
   (f) $\forall\alpha.\ \alpha \to \alpha \to \alpha$
   (g) $\forall\alpha.\ \forall\beta.\ (\alpha, \beta) \to \alpha$
   (h) $\forall\alpha.\ (\alpha, \alpha) \to \alpha$
   (i) $\forall\alpha.\ \forall\beta.\ \alpha \to (\alpha, \beta)$
   (j) $\forall\alpha.\ [\alpha] \to [\alpha]$

3. [2 pts] Use the Curry-Howard isomorphism to prove that $A \Rightarrow (B \Rightarrow A)$ is true for all propositions A and B. You need only show that there exists an expression with the corresponding type. Show the typing derivation for such an expression.

4. [2 pts] This question is for extra credit. By answering this question, you could earn up to 12 points out of 10 on this assignment.

   When discussing the Curry-Howard isomorphism, we avoided discussion of logical negation. Conjunction corresponds to product types (e.g., tuples in Haskell); disjunction corresponds to sum types (e.g., the `Either` datatype in Haskell); and negation corresponds to ... what? We can figure this out by observing that for a proposition A, $\neg A$ is equivalent to $\forall X.\ A \Rightarrow X$. The Haskell type corresponding to $\neg A$ is thus `forall x. a -> x`. The default Haskell type system does not allow us to write this type, but we can write it by running `ghc` with the command-line option `-XImpredicativeTypes`. We can then define the type alias:

   ```
   type Not a = forall x. a -> x
   ```

Note that (by parametricity) the only values of this type are functions that do not terminate.

Now, recall DeMorgan's laws:

$$\neg(A \wedge B) = \neg A \vee \neg B$$
$$\neg(A \vee B) = \neg A \wedge \neg B$$

These laws show that there is a duality between $\wedge$ and $\vee$. Similarly, through the Curry-Howard isomorphism there is a duality between product types and sum types. DeMorgan's laws hold for types as well. That is, we can transform a "negated" product into a sum of negations; we can transform a negated sum into a product of negations.

(a) Using pairs, `Either`, and `Not`, what are the types corresponding to the following four propositions:

$$A \wedge B \Rightarrow \neg(\neg A \vee \neg B)$$
$$A \vee B \Rightarrow \neg(\neg A \wedge \neg B)$$
$$\neg A \vee \neg B \Rightarrow \neg(A \wedge B)$$
$$\neg A \wedge \neg B \Rightarrow \neg(A \vee B)$$

(b) For each of the above propositions, define a Haskell function of the appropriate type to prove the proposition.

Submission

1. Complete the survey linked from the course web page after completing this assignment.
2. Submit your solutions on Moodle by 23:55 on Wednesday, 22 May 2013. Include your name in each file you submit.