Final test: Type Theory and Coq 2010

18 january 2011, 10:30–12:30, HG00.308

The mark for this test is the total number of points divided by ten, where the first 10 points are free.

1. Give a term of the simply typed lambda calculus that corresponds to the untyped term

$$\lambda xz.\, x\,(\lambda y.\, z)$$

(6 points)

2. (a) Give a proof in minimal first order propositional logic of the proposition

$$(a \to b) \to (a \to c \to b)$$

(6 points)

   (b) Give a term of the simply typed lambda calculus that corresponds with this proof under the Curry-Howard isomorphism.

(6 points)

   (c) Give a type derivation of the type judgment of this term.

(6 points)

3. (a) Give a proof in minimal propositional logic that has a detour for the implication.

(6 points)

   (b) Normalize this proof.

(4 points)

   (c) Give the reduction in the simply typed lambda calculus that corresponds with this proof normalization.

(4 points)

4. (a) Give a proof in minimal first order predicate logic of the proposition

$$(\forall x.\, (P(x) \to Q(x))) \to \forall x.\, ((Q(x) \to R(x)) \to P(x) \to R(x))$$

(6 points)

(b) Give a term of the dependently typed lambda calculus λP that corresponds with this proof under the Curry-Howard isomorphism. Use the type D for the domain that is being quantified over.

(6 points)

5. (a) Give a type derivation in the polymorphic lambda calculus $\lambda 2$ of the type judgment

$$b : * \vdash (\Pi a : *.\, (a \to b)) : *$$

The typing rules of $\lambda 2$ are given on page 4 of this test.

(6 points)

(b) Suppose we have a function of type $\Pi a : *.\, (a \to b)$. Can we apply this function to its own type?

(4 points)

(c) Which of the type systems $\lambda\!\to$, λP and $\lambda 2$ are called *impredicative*?

(4 points)

6. (a) Give Coq definitions of two inductive types, one of *cons*-lists of natural numbers and one of *snoc*-lists of natural numbers. (Note that we are not talking about *views* here, we ask for two separate types. This also means that the *nil*s of the two kinds of lists will need to have different names.)

(4 points)

(b) Give the induction principle of the type of *snoc*-lists.

(6 points)

(c) Give a Coq definition of a recursive function that converts *snoc*-lists into *cons*-lists.

You may assume a function `append` has been defined on *cons*-lists.

(6 points)

7. For efficiency we want to use a type of *binary* natural numbers:

```
Definition binnat := list bool.
```

The elements of these lists correspond to the bits of the numbers. Now suppose we want to have a *view* of this type as *unary* numbers, for

example because we would like to define functions on it by primitive recursion.

For this we first define counterparts of the unary constructors on `binnat`:

```
Definition zero : binnat := cons false nil.
Definition succ : binnat -> binnat := ...
```

The definition of `zero` is a list consisting of a single `false`, representing a single 0 bit. The definition of the `succ` function is non-trivial and is omitted here.

(a) Now to use the method from *the view from the left* we need to define an inductive type `Unary`. This is the counterpart to the `Back` type in the example from Section 5 of the paper. Give the definition of this type.

Both Coq notation or the notation from the *the view from the left* paper are allowed.

(6 points)

(b) Furthermore a function `unary` from `binnat` to `Unary` needs to be defined that shows that every binary natural number has a unary view. This is the counterpart to the `back` function in the example from the paper. Give the type of this function.

The actual definition of this function is again non-trivial and does not need to be given.

(4 points)

Derivation rules of the Pure Type Systems λP and $\lambda 2$

In these rules the variable s ranges over the set of sorts $\{*, \Box\}$. The product rule differs between λP and $\lambda 2$.

axiom

$$\frac{}{\vdash * : \Box}$$

variable

$$\frac{\Gamma \vdash A : s}{\Gamma,\, x : A \vdash x : A}$$

weakening

$$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s}{\Gamma,\, x : C \vdash A : B}$$

application

$$\frac{\Gamma \vdash M : \Pi x : A.\, B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B[x := N]}$$

abstraction

$$\frac{\Gamma,\, x : A \vdash M : B \qquad \Gamma \vdash \Pi x : A.\, B : s}{\Gamma \vdash \lambda x : A.\, M : \Pi x : A.\, B}$$

product (λP)

$$\frac{\Gamma \vdash A : * \qquad \Gamma,\, x : A \vdash B : s}{\Gamma \vdash \Pi x : A.\, B : s}$$

product ($\lambda 2$)

$$\frac{\Gamma \vdash A : s \qquad \Gamma,\, x : A \vdash B : *}{\Gamma \vdash \Pi x : A.\, B : *}$$

conversion

$$\frac{\Gamma \vdash A : B \qquad \Gamma \vdash B' : s}{\Gamma \vdash A : B'} \text{ where } B =_\beta B'$$