Type Theory and Coq 2013

second first opportunity
21-01-2014

Write your name on each paper that you hand in. Each subexercise is worth 3 points, 10 points are free, and the final mark is the number of points divided by 10. Write proofs, terms and types in this test according to the conventions of Femke's course notes. Good luck!

1. Consider the following formula of first order propositional logic:

$$((A \to B) \to A) \to (A \to B) \to B$$

   (a) Give a proof in first order propositional logic of this formula. (Write all names of the proof rules in the proof tree.)

   (b) Give the proof term in simply typed λ-calculus à la Church.

   (c) Give the type judgment for the term from the previous subexercise.

   (d) Give a derivation of the type judgment from the previous subexercise. (You do not need to give names for the typing rules in the derivation tree, and you may use abbreviations for contexts.)

   (e) Is the proof term from subexercise (1b) in normal form? If not, reduce it to a normal form. If yes, give a term of this type that is not in normal form.

2. Consider the following formula of first order predicate logic:

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

   Furthermore we have the following λC context:

$$\Gamma_2 := D : *\,, P : D \to *\,, Q : D \to *$$

   (a) Give a proof in first order predicate logic of this formula. (Write all names of the proof rules in the proof tree.)

(b) Which of the rules in this proof has a variable condition, what is this condition, and why is it satisfied?

(c) Give the type of λC that corresponds to the formula in the context Γ_2. (Use the syntax for dependent products from Femke's course notes, *i.e.*, written using Π and with explicit types.)

(d) Give a λC proof term for the type from the previous subexercise. (See page 5 for the typing rules.)

3. Consider the following term of λC:

$$\mathsf{and}_2 := \lambda A, B : *. \, \Pi C : *. \, (A \to B \to C) \to C$$

(a) Give the type of and_2 in λC. (See page 5 for the typing rules.)

(b) Is and_2 also typeable in $\lambda 2$? Explain your answer.

(c) Give a term of λC that inhabits the following type:

$$\Pi A, B : *. \, A \to B \to \mathsf{and}_2 \, A \, B$$

(d) Give a term of λC that inhabits the following type:

$$\Pi A, B : *. \, \mathsf{and}_2 \, A \, B \to A$$

4. Consider the λC type $* \to *$.

(a) Give the λC typing judgment (without a derivation) that gives the *kind* of this type.

(b) Give a derivation in λC of the judgment from the previous subexercise. (See page 5 for the typing rules. You do not need to give names for the typing rules in the derivation tree.)

(c) Give an *inhabitant* in λC of this type that is not the identity.

5. Consider the following terms of the untyped λ-calculus:

$$K^* := \lambda x.\lambda y.y \tag{1}$$
$$\Omega := (\lambda x.xx)(\lambda x.xx) \tag{2}$$
$$M := K^*\Omega \tag{3}$$

(a) Which of these terms are confluent? Explain your answer.

(b) Which of these terms are SN (strongly normalizing)? Explain your answer.

(c) Which of these terms are WN (weakly normalizing)? Explain your answer.

(d) Which of these terms are typeable in simply typed λ-calculus à la Curry? For the terms that are typeable, you have to give the most general type (but you do not need to explain how you have obtained this type, nor why it is a most general type). For the terms that are not typeable, you should explain why not.

6. Consider the Coq inductive type for lists:

```
Inductive list (A : Type) : Type :=
  | nil : list A
  | cons : A -> list A -> list A.
```

(a) What is the type of `list`, `nil`, and `cons`?

(b) Give a Coq term that represents the list of natural numbers (3 1 4).

(c) Give the type of the dependent recursion principle `list_rect` for this inductive type. (You are allowed to use either Coq or PTS syntax to give this induction principle.)

(d) Write a function `map` with type

```
forall A B : Type, (A -> B) -> list A -> list B
```

that maps a function over the elements of the list using a combination of `Fixpoint` and `match`.

(e) Now also write the `map` function using the recursion principle from subexercise (6c).

7. Consider the type for equality in HoTT, written using Coq syntax:

```
Inductive eq (A : Type) (x : A) : A -> Type :=
  | refl : eq A x x.
```

We write $x =_A y$ or even $x = y$ for $(\texttt{eq}\, A\, x\, y)$. The induction principle for equality is:

$$\frac{A : \texttt{Type} \quad x : A \quad P : \Pi z : A.\, x = z \to \texttt{Type} \quad H : Px(\texttt{refl}\, A\, x) \quad y : A \quad p : x = y}{\texttt{eq\_rect}\, A\, x\, P\, H\, y\, p : Pyp}$$

with reduction rule:

$$\texttt{eq\_rect}\, A\, x\, P\, H\, x\, (\texttt{refl}\, A\, x) \to_\iota H$$

(a) To what kind of mathematical objects do types and equality correspond using the homotopy interpretation of type theory?

(b) Write a function `transport` with type

$$\Pi A : \texttt{Type}.\, \Pi P : A \to \texttt{Type}.\, \Pi x, y : A.\, x = y \to Px \to Py$$

using the induction principle `eq_rect`. (You are allowed to use either Coq or PTS syntax.)

(c) Dependent functions are *fibrations* in the homotopy interpretation of type theory. That means, there is a function `apd` with type

$$\Pi A : \texttt{Type}.\, \Pi P : A \to \texttt{Type}.\, \Pi f : (\Pi x : A.Px).$$
$$\Pi x, y : A.\, \Pi p : x = y.\, p_*(fx) = fy,$$

where p_* denotes $\texttt{transport}\, A\, P\, x\, y\, p$. Draw a diagram to indicate why the use of p_* is necessary.

(d) Give the definition of `apd` using the induction principle `eq_rect`. (You are allowed to use either Coq or PTS syntax.)

(e) HoTT extends type theory with additional features. Name two of those features and give an example of their use.

In these rules the variables s, s_1 and s_2 range over the set of sorts $\{*, \Box\}$.

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

$$\frac{\Gamma \vdash A : s_1 \qquad \Gamma,\, x : A \vdash B : s_2}{\Gamma \vdash \Pi x : A.\, B : s_2}$$

conversion

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