Type Theory and Coq 2013

first first opportunity
07-01-2014

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 term of the untyped lambda calculus:

$$\lambda x.\lambda y.\lambda z.x(yz)zy$$

   (a) Write this term with brackets that show how the lambdas and applications associate.

   (b) Give a most general type of this term, where the term is taken to be a term of Curry-style simple type theory. (You do not need to explain how you obtained this type, nor why it is a most general type.)

   (c) Give the term of Church-style simple type theory that corresponds to the untyped lambda term and that has the type from the previous subexercise.

   (d) Give a type of this term in Curry-style simple type theory that is *not* a most general type.

2. Consider the formula of first order propositional logic:

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

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

   (b) Give the proof term of Church-style simple type theory of this proof.

   (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.)

3. Consider the formula of first order predicate logic:

$$(\forall x.R(x,c)) \to (\forall x.\exists y.R(x,y))$$

Furthermore we have the λC context:

$$\begin{array}{lrcl} \Gamma_3 & :\equiv & D & : & *\,, \\ & & c & : & D\,, \\ & & R & : & D \to D \to *\,, \\ & & \mathsf{ex} & : & \Pi A : *.\,(A \to *) \to *\,, \\ & & \mathsf{ex\_intro} & : & \Pi A : *.\,\Pi P : (A \to *).\,\Pi x : A.\,Px \to \mathsf{ex}\,A\,P \end{array}$$

(a) Give a proof in first order predicate logic of this formula. (Write all the 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_3. (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.

(e) The type judgment for the term from the previous subexercise, which encodes a proof of first order predicate logic, is a judgment of λC. Is it also a judgment of λP?

4. Consider the term of λC:

$$\mathsf{ex}_2 \;:\equiv\; \lambda A : *.\,\lambda P : (A \to *).\,\Pi Q : *.\,(\Pi x : A.\,P\,x \to Q) \to Q$$

(a) Give the type of ex_2 in λC. (See page 5 for the typing rules of λC, in case you need those.)

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

$$\Pi A : *.\,\Pi P : (A \to *).\,\Pi x : A.\,Px \to \mathsf{ex}_2\,A\,P$$

5. Consider the λC type $a \to *$ in the context $a : *$.

   (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 of λC. You do not need to give names for the typing rules in the derivation tree.)

   (c) Give also an *inhabitant* in λC of this type.

   (d) Give the λC typing judgment (without a derivation) for this inhabitant.

6. Consider the Coq inductive type for unlabeled binary trees:

```
Inductive tree : Set :=
| leaf : tree
| node : tree -> tree -> tree.
```

   (a) Give a Coq term that represents a tree with four leaves.

   (b) Give the type of the dependent induction principle `tree_rect` for this inductive type. (You can write this induction principle using Coq syntax or using PTS syntax, whatever is your preference.)

   (c) Give the type of the corresponding non-dependent induction principle `tree_rect_nondep` for this inductive type.

   (d) Write a function `mirror` that mirrors the tree using a combination of `Fixpoint` and `match`. (For instance

```
mirror (node (node (node leaf leaf) leaf) leaf) =
node leaf (node leaf (node leaf leaf))
```

   should hold.)

   (e) Now also write the `mirror` function using the non-dependent induction priciple used as a primitive recursor.

7. Consider the type that in HoTT is used for equality, 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 $(\mathsf{eq}\, A\, x\, y)$.

(a) What is *path induction*?

(b) Give the path induction principle that corresponds to the `eq` type. (This rule is called J' in the master thesis of Dijkstra, is called $\mathsf{ind}'_{=_A}$ in the HoTT book, and is called `eq_rect` in Coq.) You may write it either using Coq syntax, using PTS syntax, or in the shape of a derivation rule.

(c) Prove symmetry of equality using path induction in the style of the HoTT book.

(d) Give the type that represents this symmetry statement.

(e) Give the proof term (using J') of the symmetry proof that inhabits the type from the previous subexercise.

(f) Explain why the homotopical interpretation of equality makes clear that the path induction principle cannot be used to show that all proofs of $x =_A x$ are equal to `refl`. Or, more explicitly, why from $p : x =_A x$ one cannot deduce using this principle that

$$p =_{(x =_A x)} \texttt{refl}\, A\, x$$

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

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