Type Theory and Coq 2015-2016

29-06-2016

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. (a) Consider the three untyped lambda terms:

$$\begin{aligned} I &:= \lambda x.\, x \\ K &:= \lambda x.\, \lambda y.\, x \\ S &:= \lambda x.\, \lambda y.\, \lambda z.\, xz(yz) \end{aligned}$$

   For each of these three terms give a most general type in the Curry-style simply typed lambda calculus.

   (b) Give the three terms of the Church-style simply typed lambda calculus that correspond to the typings in the previous subexercise. (I.e., give versions of these terms where types for the variables are given explicitly.)

   (c) Give a Church-style typed lambda term for the term

$$I\,I$$

   where I is the lambda term given above.

   (d) Give a full type derivation in the simply typed lambda calculus for the term from the previous subexercise.

   (e) Give the natural deduction proof that corresponds to the lambda term in the previous two subexercises according to the Curry-Howard isomorphism.

   (f) Does the proof from the previous subexercise contain a detour? Explain your answer. If so, also give the normal form of this proof.

2. (a) Give a natural deduction proof of the propositional formula

$$a \wedge b \to b \wedge a$$

(b) Give the proof term for this proof according to the Curry-Howard isomorphism. You can use in this term the three functions:

$$\begin{aligned} \mathsf{conj} &: \Pi a : *.\, \Pi b : *.\, a \to b \to (a \wedge b) \\ \mathsf{proj}_1 &: \Pi a : *.\, \Pi b : *.\, (a \wedge b) \to a \\ \mathsf{proj}_2 &: \Pi a : *.\, \Pi b : *.\, (a \wedge b) \to b \end{aligned}$$

(c) Give definitions of proj_1 and proj_2 using the recursor of the conjuction:

$$\mathsf{and\_ind} : \Pi a : *.\, \Pi b : *.\, \Pi c : *.\, (a \to b \to c) \to (a \wedge b) \to c$$

3. (a) Give a natural deduction proof in minimal predicate logic of:

$$\forall x.\, ((\forall y.\, p(x, y)) \to p(x, x))$$

(b) Give the proof term that corresponds to the proof from the previous subexercise under the Curry-Howard isomorphism.

(c) Give the full λP judgement (including the context) that gives the typing of the term from the previous subexercise. (Note that you do not need to give the *derivation* of that judgment.)

4. In this exercise we work in the context:

$$\Gamma := \mathsf{nat} : *,\ \mathsf{O} : \mathsf{nat},\ \mathsf{S} : \mathsf{nat} \to \mathsf{nat}$$

(a) One can encounter the following three expressions:

$$\begin{aligned} M_1 &:= \lambda x : \mathsf{nat}.\, \mathsf{nat} \\ M_2 &:= \Pi x : \mathsf{nat}.\, \mathsf{nat} \\ M_3 &:= \forall x : \mathsf{nat}.\, \mathsf{nat} \end{aligned}$$

Explain what each of these expressions mean.

(b) What are the types of these three expressions M_1, M_2 and M_3?

(c) For each of the expressions M_1, M_2 and M_3, give a term, where the expression occurs as a *proper* subterm. These terms should be well typed in the context Γ of this exercise.

5. (a) Is the following expression well-typed in the calculus of constructions λC?

$$(\lambda x : *. \, x)(\Pi y : *. \, y)$$

   (b) If your answer to the previous subexercise was 'yes', give the type of this expression. If it was 'no', explain why this is not well-typed.

   (c) Give the full λC derivation of the type judgement

$$\vdash (\lambda x : *. \, x) : * \to *$$

   You can find the rules of λC on page 6 of this test.

6. (a) Give the Coq definition of an inductive type `tree` for binary trees where the leaves do not have a label, but where the nodes are both labeled with a natural number and with a color (red or black). If you like, you can use the Coq type bool for the color, but you can also define a Coq type color for yourself, if you prefer that.

   (b) Give the Coq type of the induction principle of the type you have just defined.

   (c) Give the Coq definition of a recursive function count_nodes that counts the number of nodes in the tree. (The function that adds two natural numbers is called `plus`.)

   (d) Give the Coq definition of a predicate not_red_root that says that a tree does not have a red root (where the leaves are taken to be black).

   (e) Give the Coq definition of an inductive predicate okay that says that in a tree of the type that you just defined, a red node will never have a red child.

7. We want a Coq formalization of the semantics of a very small imp-like language. The syntax of this language will be:

$$a ::= n \mid x \mid (a_1 \mathbin{\dot{-}} a_2)$$
$$c ::= \mathsf{skip} \mid (x := a) \mid (c_1 ; c_2) \mid (\mathsf{while}\ a\ \mathsf{do}\ c\ \mathsf{od})$$

   We will interpret the arithmetic expressions a as natural numbers, where subtraction is 'cut-off' subtraction (this is zero if the result would have been negative, so $4 \mathbin{\dot{-}} 3 = 1$, but $3 \mathbin{\dot{-}} 4 = 0$), and we will interpret the condition of the while as 'true' if the number is not equal to zero and 'false' if it is equal to zero. For convenience we also will use natural numbers as the identifiers for the variables x.

(a) Write Coq definitions of the syntax of this language as inductive types. Call the types that you define id (for the identifiers), aexp and com.

(b) Write a Coq definition for a type that represents the states of this language. Call this type state.

(c) Write a Coq definition for the evaluation function aeval that corresponds to $[\![a]\!]_s$. The cut-off subtraction function in Coq is called minus.

(d) The rules of a big step semantics for this language are:

$$\frac{[\![a]\!]_s = n}{(x := a, s) \Downarrow s[x \mapsto n]}$$

$$\frac{(c_1, s) \Downarrow s' \quad (c_2, s') \Downarrow s''}{(c_1; c_2, s) \Downarrow s''}$$

$$\frac{[\![a]\!]_s = 0}{(\mathsf{while}\ a\ \mathsf{do}\ c\ \mathsf{od}, s) \Downarrow s}$$

$$\frac{[\![a]\!]_s \neq 0 \quad (c, s) \Downarrow s' \quad (\mathsf{while}\ a\ \mathsf{do}\ c\ \mathsf{od}, s') \Downarrow s''}{(\mathsf{while}\ a\ \mathsf{do}\ c\ \mathsf{od}, s) \Downarrow s''}$$

Formalize these rules as an inductive relation in Coq. Call the relation

$$\mathsf{ceval} : \mathsf{com} \to \mathsf{state} \to \mathsf{state} \to \mathsf{Prop}$$

You can use a function

$$\mathsf{update} : \mathsf{state} \to \mathsf{id} \to \mathsf{nat} \to \mathsf{state}$$

for $s[x \mapsto n]$ without defining it.

(e) Someone extended this until she also had a small step semantics of this language formalized in Coq, as a relation:

$$\mathsf{cstep} : (\mathsf{com} \times \mathsf{state}) \to (\mathsf{com} \times \mathsf{state}) \to \mathsf{Prop}$$

Give the Coq *statement* that says that the semantics given by ceval and the semantics given by cstep correspond to each other. You can use the function

$$\mathsf{star} : \Pi X : \mathsf{Set}.\, (X \to X \to \mathsf{Prop}) \to (X \to X \to \mathsf{Prop})$$

that gives the reflexive and transitive closure of a relation, without defining it.

8. The proof of strong normalization of the simply typed lambda calculus that was presented in the course associates a set of untyped lambda terms $[\![A]\!]$ to each simple type A. These sets are called *saturated* sets and are defined in a way that they have the two key properties:

   - Each lambda term that can be typed (in the style of Curry) with type A will be in $[\![A]\!]$.
   - Each term in $[\![A]\!]$ will be strongly normalizing.

   Now answer the following questions:

   (a) The recursive definition of $[\![A]\!]$, where A is a type of the simply typed lambda calculus, has the structure:

   $$\begin{aligned} [\![a]\!] &:= \mathsf{SN} \qquad\qquad \text{for } a \text{ an atomic type} \\ [\![A \to B]\!] &:= \ldots \end{aligned}$$

   Here SN is the set of all strongly normalizing untyped lambda terms. Complete this definition by filling in the dots for the second case.

   (b) Prove with simultaneous induction that

   - $[\![A]\!] \subseteq \mathsf{SN}$
   - $xN_1 \ldots N_k \in [\![A]\!]$ when $N_1, \ldots, N_k \in \mathsf{SN}$

   (If you do not know the answer to the previous subexercise, at least prove the base case.)

Typing rules of the type theory λC

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

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