Type Theory and Coq 2013

first first opportunity
07-01-2014

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.

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

   (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.)

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

   (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.

$$\lambda x : (a \to b \to (b \to a) \to c).\, \lambda y : (b \to a).\, \lambda z : b.\, x(yz)zy$$

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

$$(a \to a \to (a \to a) \to a) \to (a \to a) \to a \to a$$

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.)

$$\dfrac{\dfrac{\dfrac{\dfrac{[a \to b \to c^x] \quad [a^z]}{b \to c} E{\to} \quad [b^y]}{c} E{\to}}{a \to c} I[z]{\to}}{b \to a \to c} I[y]{\to}}{(a \to b \to c) \to (b \to a \to c)} I[x]{\to}$$

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

$$\lambda x : (a \to b \to c).\, \lambda y : b.\, \lambda z : a.\, xzy$$

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

$$\vdash (\lambda x : (a \to b \to c).\, \lambda y : b.\, \lambda z : a.\, xzy) : (a \to b \to c) \to (b \to a \to c)$$

(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.)
We use the abbreviation $\Gamma_2 :\equiv x : (a \to b \to c),\, y : b,\, z : a$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma_2 \vdash x : a \to b \to c \quad \Gamma_2 \vdash z : a}{\Gamma_2 \vdash xz : b \to c} \quad \Gamma_2 \vdash y : b}{\Gamma_2 \vdash xzy : c}}{x : (a \to b \to c),\, y : b \vdash \lambda z : a.\, xzy : a \to c}}{x : (a \to b \to c) \vdash \lambda y : b.\, \lambda z : a.\, xzy : b \to a \to c}}{\vdash \lambda x : (a \to b \to c).\, \lambda y : b.\, \lambda z : a.\, xzy : (a \to b \to c) \to (b \to a \to c)}$$

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:

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

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

$$\dfrac{\dfrac{\dfrac{\dfrac{[\forall x.R(x,c)^H]}{R(x,c)}\, E\forall}{\exists y.R(x,y)}\, I\exists}{\forall x.\exists y.R(x,y)}\, I\forall}{(\forall x.R(x,c)) \to (\forall x.\exists y.R(x,y))}\, I[H]\to$$

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

The $I\forall$ rule has the variable condition that the variable x should not occur free in any of the available assumptions. Yes, this variable condition is satisfied, as the only available assumption at that point is $\forall x.R(x,c)$, and x does not occur free in that assumption.

(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.)

$$(\Pi x : D.\, Rxc) \to (\Pi x : D.\, \mathsf{ex}\, D\, (\lambda y : D.\, Rxy))$$

(d) Give a λC proof term for the type from the previous subexercise.

$$\lambda H : (\Pi x : D.\, Rxc).\, \lambda x : D.\, \mathsf{ex\_intro}\, D\, (\lambda y : D.\, Rxy)\, c\, (Hx)$$

(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?

No it is not. The typing of ex and ex_intro also needs the product rule with $s_1 = s_2 = \Box$, and therefore we need to be at least in $\lambda P\underline{\omega}$.

4. Consider the term of λC:

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

(a) Give the type of ex_2 in λC. (See page 8 for the typing rules of λC, in case you need those.)

$$\Pi A : *.\, (A \to *) \to *$$

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

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

$$\lambda A : *.\, \lambda P : (A \to *).\, \lambda x : A.\, \lambda H_1 : Px.\, \lambda Q : *.\, \lambda H_2 : (\Pi x : A.\, P\, x \to Q).\, H_2 x H_1$$

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.

$$a : * \vdash (a \to *) : \Box$$

(b) Give a derivation in λC of the judgment from the previous subexercise. (See page 8 for the typing rules of λC. You do not need to give names for the typing rules in the derivation tree.)

$$\dfrac{\dfrac{\dfrac{}{\vdash * : \Box}}{a : * \vdash a : *} \qquad \dfrac{\dfrac{\dfrac{}{\vdash * : \Box} \quad \dfrac{}{\vdash * : \Box}}{a : * \vdash * : \Box} \qquad \dfrac{\dfrac{}{\vdash * : \Box}}{a : * \vdash a : *}}{a : *,\, x : a \vdash * : \Box}}{a : * \vdash a \to * : \Box}$$

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

$$\lambda x : a.\, a$$

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

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

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.

```
node (node (node leaf leaf) leaf) leaf
```

(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.)

```
forall P : tree -> Type,
  P leaf ->
  (forall t1 : tree, P t1 -> forall t2 : tree, P t2 ->
     P (node t1 t2)) ->
  forall t : tree, P t
```

$$\begin{array}{l}\Pi P : (\texttt{tree} \to *).\\ \quad P\,\texttt{leaf} \to\\ \quad (\Pi t_1 : \texttt{tree}.\, P t_1 \to \Pi t_2 : \texttt{tree}.\, P t_2 \to P\,(\texttt{node}\, t_1\, t_2)) \to\\ \quad \Pi t : \texttt{tree}.\, P t\end{array}$$

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

```
forall P : Type,
  P ->
  (tree -> P -> tree -> P -> P) ->
  tree -> P
```

$$\Pi P : *.\, P \to (\texttt{tree} \to P \to \texttt{tree} \to P \to P) \to \texttt{tree} \to P$$

(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.)

```
Fixpoint mirror (t : tree) {struct t} : tree :=
  match t with
  | leaf => leaf
  | node t1 t2 => node (mirror t2) (mirror t1)
  end.
```

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

```
tree_rect_nondep tree leaf
  (fun t1 : tree => fun m1 : tree =>
   fun t2 : tree => fun m2 : tree =>
     node m2 m1)
```

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*?

Path induction is the proof method where if one wants to prove a fact for all paths

$$p : x =_A y$$

it is sufficient to set y to x in the fact to be proved and then prove it with p everywhere replaced by

$$\texttt{refl}\, A\, x : x =_A x$$

(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.

$$\frac{A : \mathcal{U} \quad x : A \quad P : \forall y : A.\, x =_A y \to \mathcal{U} \quad H : P\, x\, (\texttt{refl}\, A\, x) \quad y : A \quad p : x =_A y}{J' A x P H y p : P y p}$$

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

We need to prove $y =_A x$ for all paths $p : x =_A y$. With path induction it is sufficient to prove this with y set to x. But then `refl` is a proof.

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

$$\forall A : \mathcal{U}.\, \forall x, y : A.\, x =_A y \to y =_A x$$

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

$$\begin{aligned} &\lambda A : \mathcal{U}.\, \lambda x, y : A.\, \lambda p : (x =_A y). \\ &\quad J' A\, x\, (\lambda y : A.\, \lambda q : (x =_A y).\, y =_A x)\, (\texttt{refl}\, A\, x)\, y\, p \end{aligned}$$

(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

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

This does not hold under this interpretation for the circle, because there is no homotopy between the interpretations of `loop` and `refl`.