# Type Theory and Coq 2013

second first opportunity
21-01-2014

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.)

$$\dfrac{\dfrac{\dfrac{[A \to B^y] \qquad \dfrac{[(A \to B) \to A^x] \qquad [A \to B^y]}{A}\, E{\to}}{B}\, E{\to}}{(A \to B) \to B}\, I[y]{\to}}{((A \to B) \to A) \to (A \to B) \to B}\, I[x]{\to}$$

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

$$\lambda x : (A \to B) \to A\,.\lambda y : A \to B\,.y(xy)$$

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

$$\vdash \lambda x : (A \to B) \to A\,.\lambda y : A \to B\,.y(xy) : ((A \to B) \to A) \to (A \to B) \to B$$

(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 $\Gamma_1 := x : (A \to B) \to A,\ y : A \to B$.

$$\dfrac{\dfrac{\dfrac{\Gamma_1 \vdash y : A \to B \qquad \dfrac{\Gamma_1 \vdash x : (A \to B) \to A \qquad \Gamma_1 \vdash y : A \to B}{\Gamma_1 \vdash xy : A}}{\Gamma_1 \vdash y(xy) : B}}{x : (A \to B) \to A \vdash \lambda y : A \to B\,.y(xy) : (A \to B) \to B}}{\vdash \lambda x : (A \to B) \to A\,.\lambda y : A \to B\,.y(xy) : ((A \to B) \to A) \to (A \to B) \to B}$$

(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.

Yes, there are no $\beta$-redexes in the term. The following term is not in normal form:

$$\lambda x : (A \to B) \to A \,.\, \lambda y : A \to B \,.\, (\lambda z : B.\, z)(y(xy))$$

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 $\lambda C$ 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.)

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{[\forall x.P(x) \to Q(x)^{H_1}]}{P(x) \to Q(x)}\,E\forall \qquad \cfrac{[\forall x.P(x)^{H_2}]}{P(x)}\,E\forall}{Q(x)}\,E{\to}}{\forall x.Q(x)}\,I\forall}{(\forall x.P(x)) \to \forall x.Q(x)}\,I[H_2]{\to}}{(\forall x.P(x) \to Q(x)) \to (\forall x.P(x)) \to \forall x.Q(x)}\,I[H_1]{\to}$$

(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 $x$ should not occur freely in any of the available assumptions. Yes, this condition is satisfied, as the available assumptions at the use of this rule do not contain any free variables at all.

(c) Give the type of $\lambda C$ that corresponds to the formula in the context $\Gamma_2$. (Use the syntax for dependent products from Femke's course notes, *i.e.*, written using $\Pi$ and with explicit types.)

$$(\Pi x : D.\, Px \to Qx) \to (\Pi x : D.\, Px) \to \Pi x : D.\, Qx$$

(d) Give a $\lambda C$ proof term for the type from the previous subexercise. (See page 8 for the typing rules.)

$$\lambda H_1 : (\Pi x : D. Px \to Qx).\, \lambda H_2 : (\Pi x : D. Px).\, \lambda x : D.\, H_1 x (H_2 x)$$

3. Consider the following term of $\lambda C$:

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

(a) Give the type of $\mathsf{and}_2$ in $\lambda C$. (See page 8 for the typing rules.)

$$* \to * \to *$$

(b) Is $\mathsf{and}_2$ also typeable in $\lambda 2$? Explain your answer.
No, it is not. It also needs the product rule with $s_1 = s_2 = \Box$, and therefore we need to be at least in $\lambda\omega$.

(c) Give a term of $\lambda C$ that inhabits the following type:

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

$$\lambda A, B : *.\, \lambda x : A.\, \lambda y : B.\, \lambda C : *.\, \lambda f : A \to B \to C.\, fxy$$

(d) Give a term of $\lambda C$ that inhabits the following type:

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

$$\lambda A, B : *.\, \lambda p : \mathsf{and}_2\, A\, B.\, pA(\lambda a : A.\, \lambda b : B.\, a)$$

4. Consider the $\lambda C$ type $* \to *$.

(a) Give the $\lambda C$ typing judgment (without a derivation) that gives the *kind* of this type.

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

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

$$\dfrac{\vdash * : \square \qquad \dfrac{\vdash * : \square \qquad \vdash * : \square}{A : * \vdash * : \square}}{\vdash * \to * : \square}$$

(c) Give an *inhabitant* in $\lambda C$ of this type that is not the identity.

$$\lambda A : *. \, A \to A$$

5. Consider the following terms of the untyped $\lambda$-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.

All these terms are confluent, because $\beta$-reduction in the $\lambda$-calculus is confluent (by *e.g.* Takahashi's proof).

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

The term $K^*$ is in normal form, and hence it is SN. The term $\Omega$ reduces to itself, and hence it is not SN. The term $M$ reduces to itself, and hence it is not SN.

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

The term $K^*$ is SN, and thus WN. The only reduction from $\Omega$ is to itself, and hence it is not WN. The term $M$ reduces to the normal form $\lambda x.x$, hence it is WN.

(d) Which of these terms are typeable in simply typed $\lambda$-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.

The terms $\Omega$ and $K^*\Omega$ are not SN, and thus not typeable because that would contradict the SN property of $\lambda\!\to$. The term $K_*$ is typeable, and has principle type $A \to B \to B$.

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`?

```
list : Type -> Type
nil : forall A : Type, list A
cons : forall A : Type, A -> list A -> list A
```

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

```
cons nat 3 (cons nat 1 (cons nat 1 (nil nat)))
```

(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.)

```
forall A : Type, forall P : list A -> Type,
  P (nil A) ->
  (forall (x : A) (l : list A),
    P l -> P (cons A x l)) ->
  forall l, P l
```

(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`.

```
Fixpoint map (A B : Type)
    (f : A -> B) (l : list A) : list B :=
  match l with
  | nil => nil B
  | cons x l => cons B (f x) (map A B f l)
  end
```

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

```
Definition map (A B : Type)
    (f : A -> B) : list A -> list B :=
  list_rect A
    (fun l : list A => list B)
    (nil B)
    (fun (x : A) (l : list A) => cons B (f x))
```

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?

Types correspond to (topological) spaces, and equality corresponds to paths between these spaces. (Equalities between equality proofs correspond to homotopies.)

(b) Write a function `transport` with type

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

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

$$\lambda A : \texttt{Type}.\, \lambda P : A \to \texttt{Type}.\, \lambda x, y : A.\, \lambda p : x = y.\, \lambda H : P x.\\ \texttt{eq\_rect}\, A\, x\, (\lambda z : A.\, \lambda q : x = z.\, P z)\, H\, y\, p$$

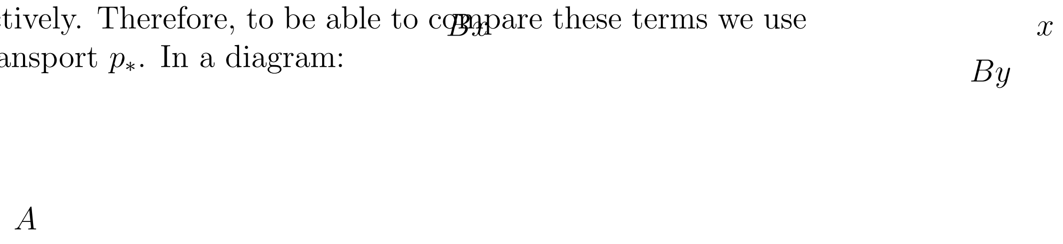
```
Definition transport :=
  fun A : Type =>
  fun P : A -> Type =>
  fun x y : A => fun p : x = y =>
  fun H : P x =>
  eq_rect A x
    (fun z : A => fun q : x = z => P z) H y p.
```

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

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

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

The terms $fx$ and $fy$ have different types, namely $Bx$ and $By$, respectively. Therefore, to be able to compare these terms we use the transport $p_*$. In a diagram:



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

$\lambda A : \texttt{Type}.\, \lambda P : A \to \texttt{Type}.\, \lambda f : (\Pi x : A.Px).\, \lambda x : A.$
$\quad \texttt{eq\_rect}\, A\, x\, (\lambda z : A.\, \lambda q : x = z.\, q_*(fx) = fz)(\texttt{refl}\,(Px)\,(fx))$

```
Definition apd :=
  fun A : Type =>
  fun P : A -> Type =>
  fun f : (forall x : A, P x) =>
  fun x : A =>
  eq_rect A x (fun z : A => fun q : x = z =>
    transport A P x z q (f x) = f z)
  (refl (P x) (f x)).
```

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

The two most important features are:

- Higher inductive types, these extend conventional inductive types with path contructors. Examples are: the interval, the circle, and quotients.
- Univalence, an axiom that enables one to prove that isomorphic types are equal. For equal, it enables one to prove that the binary naturals are equal to the unary naturals.

A consequence of both these features is functional extensionality. Other features as universe polymorphism, or universe scaling, are also allowed in case proper examples are given.