Type Theory and Coq 2024-2025

31-10-2024
12:45-14:45

1. Apply the principal typing algorithm PT to establish whether the following lambda term is typable in simple type theory:

$$\lambda f x.f(fxx)$$

Give all intermediate steps of the algorithm. Also, if this term is typable then explicitly give a principal type.

We annotate all variables and applicative subterms with type variables:

$$\lambda \overset{a}{f}\overset{b}{x}.\ \overbrace{\overset{a}{f}(\overbrace{\overset{a}{f}\overset{b}{x}}^{d}\overset{b}{x})}^{c}\ :\ a \to b \to c$$

This gives rise to the following equations, which we simplify according to the PT algorithm:

$$\left\{\begin{array}{rcl} \underline{a} & = & e \to c \\ a & = & b \to d \\ d & = & b \to e \end{array}\right. \overset{\text{(I)}}{\Longleftrightarrow} \left\{\begin{array}{rcl} a & = & e \to c \\ e \to c & = & b \to d \\ d & = & b \to e \end{array}\right. \overset{\text{(II)}}{\Longleftrightarrow} \left\{\begin{array}{rcl} a & = & e \to c \\ \underline{e} & = & b \\ c & = & d \\ d & = & b \to e \end{array}\right. \overset{\text{(I)}}{\Longleftrightarrow}$$

$$\left\{\begin{array}{rcl} a & = & b \to c \\ e & = & b \\ \underline{c} & = & d \\ d & = & b \to b \end{array}\right. \overset{\text{(I)}}{\Longleftrightarrow} \left\{\begin{array}{rcl} a & = & b \to d \\ e & = & b \\ c & = & d \\ \underline{d} & = & b \to b \end{array}\right. \overset{\text{(I)}}{\Longleftrightarrow} \left\{\begin{array}{rcl} a & = & b \to b \to b \\ e & = & b \\ c & = & b \to b \\ d & = & b \to b \end{array}\right.$$

Therefore the type $a \to b \to c$ becomes one of the principal types of this term:

$$(b \to b \to b) \to b \to b \to b$$

2. Consider the following proposition of minimal propositional logic:

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

Now answer the following five sub-questions:

(a) Give a natural deduction proof of this proposition.

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

(b) Does this proof have detours? Explain your answer.

No, there is no introduction rule followed by an elimination rule for the same connective. There is an implication elimination rule, but it does not eliminate an implication that was just introduced.

(c) Give the proof term from simple type theory that corresponds to this proof.

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

(d) Give a full type derivation of this proof term in simple type theory.

We abbreviate $\Gamma_0 := x : (a \to b) \to c,\ y : b$.

$$\dfrac{\dfrac{\dfrac{}{\Gamma_0 \vdash x : (a \to b) \to c} \quad \dfrac{\dfrac{}{\Gamma_0,\ z : a \vdash y : b}}{\Gamma_0 \vdash \lambda z : a.\, y : a \to b}}{\Gamma_0 \vdash x\,(\lambda z : a.\, y) : c}}{x : (a \to b) \to c \vdash \lambda y : b.\, x\,(\lambda z : a.\, y) : b \to c}}{\vdash \lambda x : (a \to b) \to c.\, \lambda y : b.\, x\,(\lambda z : a.\, y) : ((a \to b) \to c) \to b \to c)}$$

(e) Give a Coq proof script for this proposition. You may only use the tactics `intro`, `intros`, `apply`, `exact` and `assumption`. This proof script should be in the place of the dots in:

```
Lemma exercise_two (a b c : Prop) : ((a -> b) -> c) -> b -> c.
Proof.
...
Qed.
```

You do not need to copy the lines of the lemma already given here, just giving the tactics part is enough.

```
intros x y. apply x. intro z. apply y.
```

3. In the lecture we defined recursively:

$$\begin{aligned} x^* &:= x \\ (\lambda x.M)^* &:= \lambda x.M^* \\ ((\lambda x.P)N)^* &:= P^*[x := N^*] \\ (MN)^* &:= M^*N^* \qquad \text{if } M \text{ is not a } \lambda\text{-abstraction} \end{aligned}$$

We use the customary abbreviations:

$$\begin{aligned} \mathsf{S} &:= \lambda xyz.xz(yz) \\ \mathsf{K} &:= \lambda uv.u \end{aligned}$$

Now compute the four terms K^*, $(\mathsf{K}z)^*$, $(\mathsf{SKK})^*$ and $(\mathsf{SKK})^{**}$

Hint: note that $M^* = M$ when M does not contain any beta redexes.

We compute:

$$\mathsf{K}^* = \mathsf{K} \qquad (\mathsf{K} \text{ does not contain redexes})$$

$$\begin{aligned}(\mathsf{K}z)^* &= ((\lambda uv.u)z)^* \\ &= (\lambda v.u)^*[u := z^*] \\ &= (\lambda v.u)[u := z] \\ &= \lambda v.z\end{aligned}$$

$$\begin{aligned}(\mathsf{SKK})^* &= (\mathsf{SK})^*\mathsf{K}^* \\ &= \big((\lambda yz.xz(yz))^*[x := \mathsf{K}^*]\big)\mathsf{K} \\ &= (\lambda yz.\mathsf{K}z(yz))\mathsf{K}\end{aligned}$$

$$\begin{aligned}(\mathsf{SKK})^{**} &= (\lambda z.\mathsf{K}z(yz))^*[y := \mathsf{K}^*] \\ &= \big(\lambda z.(\mathsf{K}z)^*(yz)^*\big)[y := \mathsf{K}] \\ &= \big(\lambda z.(\lambda v.z)(yz)\big)[y := \mathsf{K}] \\ &= \lambda z.(\lambda v.z)(\mathsf{K}z)\end{aligned}$$

4. Consider the following proposition of predicate logic:

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

Now answer the following two sub-questions:

(a) Give a natural deduction proof of this proposition. For all relevant inference rules, show that the variable condition is satisfied.

$$\dfrac{\dfrac{\dfrac{\dfrac{[\forall y.\, \neg p(y)^{H_1}]}{\neg p(x)}\, E\forall \qquad [p(x)^H]}{\bot}\, E\neg}{\neg\forall y.\, \neg p(y)}\, I[H_1]\neg}{p(x) \to \neg\forall y.\, \neg p(y)}\, I[H]\to}{\forall x.(p(x) \to \neg\forall y.\, \neg p(y))}\, I\forall$$

The only rule with a variable condition in this proof is the $I\forall$ rule, and at that point there are no assumptions yet.

(b) Give the λP proof term that corresponds to this proof. For this term the context will need to contain $\bot$, where we then define $\neg A := A \to \bot$. Therefore, this proof term needs to be well-typed in the context:

$$D : *,\ p : D \to *,\ \bot : *$$

Note that we just ask for the proof term and *not* for a type derivation for this term.

You are allowed to either use mathematical notation for this term, or to use Coq syntax.

$$\lambda x : D.\, \lambda H : p\, x.\, \lambda H_1 : (\Pi y : D.\, p\, y \to \bot).\, H_1 x H$$

```
fun (x : D) (H : p x) (H1 : forall y : D, p y -> False) =>
  H1 x H
```

5. Consider the following four preterms:

$$\lambda n : \mathsf{nat}.\, n$$
$$\lambda n : \mathsf{nat}.\, \mathsf{nat}$$
$$\Pi n : \mathsf{nat}.\, n$$
$$\Pi n : \mathsf{nat}.\, \mathsf{nat}$$

Or in Coq syntax:

```
fun n : nat => n
fun n : nat => nat
forall n : nat, n
forall n : nat, nat
```

Now answer the following three sub-questions:

(a) Which of these preterms is well-typed in the calculus of constructions, with context:

$$\mathsf{nat} : *,\ z : \mathsf{nat},\ s : \mathsf{nat} \to \mathsf{nat}$$

The only one that is *not* well-typed is $\Pi n : \mathsf{nat}.\, n$. In a term $\Pi x : A.\, B$, the subterm B needs to be a type.

(b) For the ones that are well-typed: give their type.

$$\lambda n : \mathsf{nat}.\, n \ :\ \mathsf{nat} \to \mathsf{nat}$$
$$\lambda n : \mathsf{nat}.\, \mathsf{nat} \ :\ \mathsf{nat} \to *$$
$$\Pi n : \mathsf{nat}.\, \mathsf{nat} \ :\ *$$

(c) Of the ones that are well-typed: which are types?

A type is a term of which the type is a sort. Therefore only $\Pi n : \mathsf{nat}.\, \mathsf{nat}$ is a type.

6. Give λP derivations of the following five typing judgments:

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

In each derivation, you may replace a sub-derivation of a judgment earlier in the list by vertical dots. For the rules of λP see page 7.

Hint: it is often efficient to use weakening as early as possible.

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

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

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

$$\dfrac{\dfrac{\vdots}{a : *,\ x : a \vdash x : a} \qquad \dfrac{\vdots}{a : * \vdash a \to a : *}}{a : * \vdash (\lambda x : a.x) : a \to a}$$

$$\dfrac{\dfrac{\dfrac{\vdots}{a : * \vdash (\lambda x : a.x) : a \to a} \qquad \dfrac{\dfrac{}{\vdash * : \Box}}{a : * \vdash a : *}}{a : *,\ y : a \vdash (\lambda x : a.x) : a \to a} \qquad \dfrac{\vdots}{a : *,\ y : a \vdash y : a}}{a : *,\ y : a \vdash (\lambda x : a.x)\, y : a}$$

7. Consider the following Coq definition of a type for polymorphic lists:

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

This type has a dependent recursion principle `list_rec` with type:

```
list_rec
     : forall (A : Set) (P : list A -> Set),
       P (nil A) ->
       (forall (a : A) (l : list A), P l -> P (cons A a l)) ->
       forall l : list A, P l
```

We want to define the function `map` that maps a function over a list. It has type:

```
map
     : forall A B : Set, (A -> B) -> list A -> list B
```

Now answer the following two sub-questions:

(a) Define the function `map` using `Fixpoint` and `match`.
The cases of the match should be written `nil _` and `cons _ a l`. Naming the parameter in the patterns gives an error.

```
Fixpoint map (A B : Set)
    (f : A -> B) (l : list A) {struct l} : list B :=
  match l with
  | nil _ => nil B
  | cons _ a l => cons B (f a) (map A B f l)
  end.
```

(b) Define the function `map` as an application of `list_rec` to appropriate arguments.

```
Definition map (A B : Set)
    (f : A -> B) : list A -> list B :=
  list_rec A (fun l : list A => list B) (nil B)
    (fun (a : A) (l : list A) (l' : list B) =>
      cons B (f a) l').
```

8. Consider the following Coq definition:

```
Inductive foo (A B : Prop) : Prop :=
| bar : A -> B -> foo A B.
```

Now answer the following three sub-questions:

(a) Which logical connective is defined here under the name of `foo`.
Conjunction.

(b) Give the type of the dependent induction principle `foo_ind_dep`.

```
foo_ind_dep
     : forall (A B : Prop) (P : foo A B -> Prop),
       (forall (a : A) (b : B), P (bar A B a b)) ->
       forall H : foo A B, P H
```

(c) Give the type of the non-dependent induction principle `foo_ind`.

```
foo_ind
     : forall A B P : Prop, (A -> B -> P) -> foo A B -> P
```

9. The proof of normalization of $\lambda 2$ associates a saturated set of untyped lambda terms $[\![A]\!]_\rho$ with each type A.

   Now answer the following two sub-questions about these sets:

   (a) Put the three sets:

   $$\mathsf{SN}$$
   $$\{M \mid M \text{ has type } A \text{ in some context}\}$$
   $$[\![A]\!]_\rho$$

   in order of inclusion. You do not need to explain this order.

   $$\{M \mid M \text{ has type } A \text{ in some context}\} \subseteq [\![A]\!]_\rho \subseteq \mathsf{SN}$$

   (b) In the lecture an important proposition was given, where a property followed from the assumptions:

   $$x_1 : A_1, \ldots, x_k : A_k \vdash M : B$$

   and

   $$N_1 \in [\![A_1]\!]_\rho, \ldots, N_k \in [\![A_k]\!]_\rho$$

   in which ρ is an arbitrary valuation that maps variables to saturated sets. Give the conclusion of this proposition.

   $$M[x_1 := N_1, \ldots, x_k := N_k] \in [\![B]\!]_\rho$$