Type Theory and Coq 2023-2024

Resit
2024-04-03
12:45–14:45

1. Consider the lambda term:

$$\lambda xyz.\, xy(z(zy))$$

Use the PT algorithm to either find a principal type for this term, or to establish that it is not typable in simple type theory. Explicitly give all the intermediate steps of the algorithm.

We annotate all variables and applicative subterms with type variables:

$$\lambda \overset{a}{x}\overset{b}{y}\overset{c}{z}.\ \underbrace{\overset{a}{x}\overset{b}{y}}_{d}(\overset{c}{z}\underbrace{(\overset{c}{z}\overset{b}{y})}_{f})\ \underbrace{\qquad\qquad}_{e}\ \underbrace{\qquad\qquad\qquad}_{g} \quad : a \to b \to c \to g$$

We then get the following equations, applying the operations of the PT algorithm:

$$\begin{cases} a = b \to d \\ \underline{c} = f \to e \\ c = b \to f \\ d = e \to g \end{cases} \overset{\text{(I)}}{\iff} \begin{cases} a = b \to d \\ c = f \to e \\ \underline{f \to e} = \underline{b \to f} \\ d = e \to g \end{cases} \overset{\text{(II)}}{\iff} \begin{cases} a = b \to d \\ c = f \to e \\ \underline{f} = b \\ e = f \\ d = e \to g \end{cases} \overset{\text{(I)}}{\iff}$$

$$\begin{cases} a = b \to d \\ c = b \to e \\ f = b \\ \underline{e} = b \\ d = e \to g \end{cases} \overset{\text{(I)}}{\iff} \begin{cases} a = b \to d \\ c = b \to b \\ f = b \\ e = b \\ \underline{d} = b \to g \end{cases} \overset{\text{(I)}}{\iff} \begin{cases} a = b \to b \to g \\ c = b \to b \\ f = b \\ e = b \\ d = b \to g \end{cases}$$

Therefore a principal type is:

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

2. Consider the following proof of minimal propositional logic:

$$\dfrac{\dfrac{\dfrac{\dfrac{[a \to b \to c^{H_1}] \quad [a^{H_3}]}{b \to c} E{\to} \quad [b^{H_2}]}{c} E{\to}}{a \to c} I[H_3]{\to}}{b \to a \to c} I[H_2]{\to}}{(a \to b \to c) \to (b \to a \to c)} I[H_1]{\to}$$

Give the corresponding proof term in Church-style simple type theory. Use the labels of the assumptions to name the variables.

$$\lambda H_1 : (a \to b \to c).\, \lambda H_2 : b.\, \lambda H_3 : a.\, H_1 H_3 H_2$$

3. Consider the following Coq script:

```
Definition id (a : Set) : a -> a.
intro x; apply x.
Defined.
Print id.
```

Now answer the following two questions:

(a) Give both the term and the type that the `Print` commando prints in Coq syntax, so using `fun` and `forall` and the sort `Set`.

```
id = fun (a : Set) (x : a) => x
     : forall a : Set, a -> a
```

(b) Give both the term and the type that the `Print` commando prints in the mathematical notation of the lambda cube, so using λ and Π and the sort $*$.

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

4. Consider the following lambda term of λP:

$$\lambda H_1 : (\Pi x : D.\, px).\, \lambda H_2 : (\Pi x : D.\, px \to qx).\, (\lambda x : D.\, H_2 x (H_1 x))c$$

This term is well-typed in the context:

$$\begin{aligned} D &: *\,, \\ p &: D \to *\,, \\ q &: D \to *\,, \\ c &: D \end{aligned}$$

Now answer the following six questions:

(a) What is the beta redex in this term.

$$(\lambda x : D.\, H_2 x (H_1 x))c$$

(b) Give the normal form of this term.

$$\lambda H_1 : (\Pi x : D.\, px).\, \lambda H_2 : (\Pi x : D.\, px \to qx).\, H_2 c(H_1 c)$$

(c) Give the natural deduction proof of predicate logic that corresponds to the term from the exercise.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\forall x.\, p(x) \to q(x)^{H_2}]}{p(x) \to q(x)}\, E\forall \quad \dfrac{[\forall x.\, p(x)^{H_1}]}{p(x)}\, E\forall}{q(x)}\, E\to}{\forall x.\, q(x)}\, I\forall}{q(c)}\, E\forall}{(\forall x.\, p(x) \to q(x)) \to q(c)}\, I[H_2]\to}{(\forall x.\, p(x)) \to (\forall x.\, p(x) \to q(x)) \to q(c)}\, I[H_1]\to$$

(d) Give the variable condition of this proof and show that it is satisfied.

The variable condition is that in the $I\forall$ rule, the variable x should not be free in the open assumptions H_1 and H_2, which is indeed the case, as these do not have free variables at all.

(e) What is the detour in this proof.

The detour consists of the $I\forall$ rule directly followed by the $E\forall$ rule.

(f) Give the normal form of the proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{[\forall x.\, p(x) \to q(x)^{H_2}]}{p(c) \to q(c)}\, E\forall \quad \dfrac{[\forall x.\, p(x)^{H_1}]}{p(c)}\, E\forall}{q(c)}\, E\to}{(\forall x.\, p(x) \to q(x)) \to q(c)}\, I[H_2]\to}{(\forall x.\, p(x)) \to (\forall x.\, p(x) \to q(x)) \to q(c)}\, I[H_1]\to$$

5. *(This exercise may take some time, consider postponing it until the end of the exam.)*

We want to give a derivation in λP of the judgement:

$$D : *,\, q : D \to *,\, c : D \vdash qc : *$$

See page 6 for the typing rules of λP. We do this in three stages, where in a later stage you may abbreviate a derivation that already has been given by vertical dots.

(a) Give a λP derivation of:

$$D : * \vdash D \to * : \Box$$

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

(b) In the remainder of this exercise we abbreviate:

$$\Gamma := D : *, q : D \to *$$

Give a λP derivation of:

$$\Gamma \vdash D : *$$

$$\dfrac{\dfrac{\dfrac{}{\vdash * : \Box}}{D : * \vdash D : *} \quad \dfrac{\vdots}{D : * \vdash D \to * : \Box}}{D : *, q : D \to * \vdash D : *}$$

(c) Give a λP derivation of:

$$\Gamma, c : D \vdash qc : *$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{D : * \vdash D \to * : \Box}}{\Gamma \vdash q : D \to *} \quad \dfrac{\vdots}{\Gamma \vdash D : *}}{\Gamma, c : D \vdash q : D \to *} \quad \dfrac{\dfrac{\vdots}{\Gamma \vdash D : *}}{\Gamma, c : D \vdash c : D}}{\Gamma, c : D \vdash qc : *}$$

6. The impredicative definition of conjunction in minimal second order propositional logic is:

$$A \wedge_2 B := (\forall c.\, (a \to b \to c) \to c)$$

Give the corresponding impredicative definition of disjunction, so give a definition of $A \vee_2 B$.

$$A \vee_2 B := (\forall c.\, (a \to c) \to (b \to c) \to c)$$

7. This exercise is about unlabeled binary trees. These just consist of binary nodes and leaves, and contain no data. Now answer the following four questions:

(a) Give the Coq definition of an inductuctive type `bintree` with constructors `leaf` and `node` that represents this kind of tree.

```
Inductive bintree : Set :=
| leaf : bintree
| node : bintree -> bintree -> bintree
```

(b) Give a Coq definition using `Fixpoint` and `match` of a function `count_leaves` that counts the number of leaves in a tree.

```
Fixpoint count_leaves (t : bintree) {struct t} : nat :=
  match t with
  | leaf => 1
  | node t1 t2 => count_leaves t1 + count_leaves t2
  end.
```

(c) Give the type of the dependent recursion principle `bintree_rec` for your type.

```
forall A : bintree -> Set,
  A leaf ->
  (forall t1 t2 : bintree, A t1 -> A t2 -> A (node t1 t2)) ->
  forall t : bintree, A t
```

(d) Give a Coq definition by applying this recursion principle to define a function `count_leaves_rec` that counts the leaves in a tree.

```
Definition count_leaves_rec :=
  bintree_rec (fun _ => nat) 1 (fun _ _ n1 n2 => n1 + n2).
```

8. This exercise is about the `even` predicate in Coq, which is defined as:

```
Inductive even : nat -> Prop :=
| even_0 : even 0
| even_SS : forall n : nat, even n -> even (S (S n))
```

Now answer the following three questions:

(a) Give the proof that four is even, i.e., a Coq term with type:

```
even (S (S (S (S 0))))
```

```
even_SS (S (S 0)) (even_SS 0 even_0)
```

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

```
forall P : (forall n : nat, even n -> Prop),
  P 0 even_0 ->
  (forall (n : nat) (H : even n), P n H ->
      P (S (S n)) (even_SS n H)) ->
    forall (n : nat) (H : even n), P n H
```

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

```
forall P : nat -> Prop,
  P 0 ->
  (forall n : nat, even n -> P n -> P (S (S n))) ->
  forall n : nat, even n -> P n
```

9. The proof of strong normalisation of $\lambda\!\rightarrow$ that was presented in the lectures associates a saturated set of untyped lambda terms $[\![A]\!]$ with each simple type A. This is defined recursively on the structure of the type as:

$$\begin{aligned} [\![a]\!] &= \mathsf{SN} && \text{for atomic types } a \\ [\![A \to B]\!] &= \ldots && \text{for types } A \text{ and } B \end{aligned}$$

Here SN it the set of strongly normalizing untyped lambda terms.

Complete this by replacing the dots by a proper definition of $[\![A \to B]\!]$.

$$\begin{aligned} [\![a]\!] &= \mathsf{SN} && \text{for atomic types } a \\ [\![A \to B]\!] &= \{M \mid \forall N \in [\![A]\!].\, MN \in [\![B]\!]\} && \text{for types } A \text{ and } B \end{aligned}$$