Type Theory and Coq 2023-2024

16-01-2024
12:45-14:45

1. Consider the lambda term:

$$\lambda xy.\, x(y(xy))$$

Apply the PT algorithm to this term, and determine either a principal type for it, or the fact that this term is not typable in simple type theory. Explicitly give all stages of the algorithm.

We annotate all variables and applicative subterms with type variables:

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

This gives rise to the following equations, which we simplify according to the PT algorithm (we underline the relevant types that the operation acts on):

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

$$\begin{cases} a &=& d \to c \\ b &=& e \to d \\ \underline{d \to c} &=& \underline{(e \to d) \to e} \end{cases} \overset{\text{(II)}}{\iff} \begin{cases} a &=& d \to c \\ b &=& e \to d \\ \underline{d} &=& \underline{e \to d} \\ c &=& e \end{cases} \overset{\text{(III)}}{\iff} \textsf{FAIL}$$

This means that the term is not typable in simple type theory.

2. Consider the following type of simple type theory:

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

Now answer the following two questions:

(a) Give an inhabitant of this type.

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

(b) Give a type derivation that shows that this inhabitant has the appropriate type.
$\Gamma := x : a \to b \to c,\ y : b,\ z : a$

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

3. Consider the following term of λP:

$$\lambda H : (\Pi x : D.\, \Pi y : D.\, Rxy).\, \lambda x : D.\, \lambda y : D.\, Hyx$$

In this term there are free variables

$$D : *$$
$$R : D \to D \to *$$

which represent the domain of quantification, and a binary relation on this domain.

Now answer the following three questions:

(a) This term is a proof term of a proof in predicate logic. Give the statement that is proved as a formula of predicate logic.

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

(b) Give the natural deduction proof of this statement that corresponds to this term. If you need to check a variable condition, indicate where this is the case, and why it holds.

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

For the $I\forall$ rules we need to check the variable condition that x and y are not free in any open assumption. The only open assumption at that point is $\forall x.\, \forall y.\, R(x, y)$, and although it does contain the variables x and y, they are not free in this.

There is a subtlety when substituting y in the first $E\forall$ rule. We need to rename the variable y in the quantifier, to prevent the free variable y that is being substituted from being captured.

(c) Does this proof contain a detour? Explain your answer.

No, it does not. There is no introduction rule that is directly followed by an elimination rule for the same connective. The proof term also does not contain a beta redex.

Note that you should not give a type derivation of this term.

4. Give type derivations in λP of the following judgment:

$$a : *,\ x : a,\ y : a \vdash y : a$$

For the rules of λP see page 7.

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

5. An impredicative definition of the Booleans in $\lambda 2$ is:

$$\mathsf{bool}_2 := \Pi a : *.\ a \to a \to a$$

Define three $\lambda 2$ terms (where ite stands for '*if then else*'):

$$\begin{aligned} \mathsf{true}_2 &: \mathsf{bool}_2 \\ \mathsf{false}_2 &: \mathsf{bool}_2 \\ \mathsf{ite}_2 &: \Pi a : *.\ \mathsf{bool}_2 \to a \to a \to a \end{aligned}$$

such that:

$$\begin{aligned} \mathsf{ite}_2\ A\ \mathsf{true}_2\ M\ N &\twoheadrightarrow_\beta M \\ \mathsf{ite}_2\ A\ \mathsf{false}_2\ M\ N &\twoheadrightarrow_\beta N \end{aligned}$$

It is sufficient that these reductions hold, you do not need to show this explicitly.

$$\begin{aligned} \mathsf{true}_2 &:= \lambda a : *.\ \lambda x : a.\ \lambda y : a.\ x \\ \mathsf{false}_2 &:= \lambda a : *.\ \lambda x : a.\ \lambda y : a.\ y \\ \mathsf{ite}_2 &:= \lambda a : *.\ \lambda z : \mathsf{bool}_2.\ \lambda x : a.\ \lambda y : a.\ z\ a\ x\ y \end{aligned}$$

(The reductions mentioned then are:

$$\begin{aligned}
\mathsf{ite}_2\ A\ \mathsf{true}_2\ M N &\equiv (\lambda a : *.\, \lambda z : \mathsf{bool}_2.\, \lambda x : a.\, \lambda y : a.\, z\, a\, x\, y)\ A\ \mathsf{true}_2\ M N \\
&\to_\beta (\lambda z : \mathsf{bool}_2.\, \lambda x : A.\, \lambda y : A.\, z\, A\, x\, y)\ \mathsf{true}_2\ M N \\
&\to_\beta (\lambda x : A.\, \lambda y : A.\, \mathsf{true}_2\, A\, x\, y)\ M N \\
&\to_\beta (\lambda y : A.\, \mathsf{true}_2\, A\, M\, y)\ N \\
&\to_\beta \mathsf{true}_2\, A\, M\, N \\
&\equiv (\lambda a : *.\, \lambda x : a.\, \lambda y : a.\, x)\, A\, M\, N \\
&\to_\beta (\lambda x : A.\, \lambda y : A.\, x)\, M\, N \\
&\to_\beta (\lambda y : A.\, M)\, N \\
&\to_\beta M
\end{aligned}$$

$$\begin{aligned}
\mathsf{ite}_2\ A\ \mathsf{false}_2\ M N &\equiv (\lambda a : *.\, \lambda z : \mathsf{bool}_2.\, \lambda x : a.\, \lambda y : a.\, z\, a\, x\, y)\ A\ \mathsf{false}_2\ M N \\
&\to_\beta (\lambda z : \mathsf{bool}_2.\, \lambda x : A.\, \lambda y : A.\, z\, A\, x\, y)\ \mathsf{false}_2\ M N \\
&\to_\beta (\lambda x : A.\, \lambda y : A.\, \mathsf{false}_2\, A\, x\, y)\ M N \\
&\to_\beta (\lambda y : A.\, \mathsf{false}_2\, A\, M\, y)\ N \\
&\to_\beta \mathsf{false}_2\, A\, M\, N \\
&\equiv (\lambda a : *.\, \lambda x : a.\, \lambda y : a.\, y)\, A\, M\, N \\
&\to_\beta (\lambda x : A.\, \lambda y : A.\, y)\, M\, N \\
&\to_\beta (\lambda y : A.\, y)\, N \\
&\to_\beta N
\end{aligned}$$

But as stated in the exercise, these reductions do not need to be included in an answer to get full points.)

6. We define an inductive type of lists of Booleans in Coq:

```
Inductive boollist : Set :=
| nil : boollist
| cons: bool -> boollist -> boollist.
```

The recursor of this type has type:

```
boollist_rec
     : forall P : boollist -> Set,
       P nil ->
       (forall (b : bool) (l : boollist), P l -> P (cons b l)) ->
       forall l : boollist, P l
```

Now answer the following two questions:

(a) Define a function `count_trues` which counts the number of elements in the list that are equal to `true`, using `Fixpoint` and `match`. This function should have type:

```
count_trues
     : boollist -> nat
```

If you like, you can abbreviate

```
match M with
| true => N_1
| false => N_2
end
```

as

```
if M then N_1 else N_2
```

```
Fixpoint count_trues (l : boollist) {struct l} : nat :=
  match l with
  | nil => O
  | cons b l => if b then S (count_trues l) else count_trues l
  end.
```

(b) Define the function `count_trues_rec` that computes the same count, by applying `boollist_rec` to appropriate arguments.

```
Definition count_trues_rec : boollist -> nat :=
  boollist_rec (fun (_ : boollist) => nat)
    O
    (fun (b : bool) (l : boollist) (n : nat) =>
       if b then S n else n).
```

7. We define an inductive type in Coq of :

```
Inductive vec (A : Set) : nat -> Set :=
| vnil : vec A O
| vcons : forall (n : nat) (x : A) (l : vec A n), vec A (S n).
```

Give the type of the (dependent) induction principle `vec_ind` for this type.

```
vec_ind
     : forall (A : Set) (P : forall n : nat, vec A n -> Prop),
       P 0 (vnil A) ->
       (forall (n : nat) (x : A) (v : vec A n),
        P n v -> P (S n) (vcons A n x v)) ->
       forall (n : nat) (v : vec A n), P n v
```

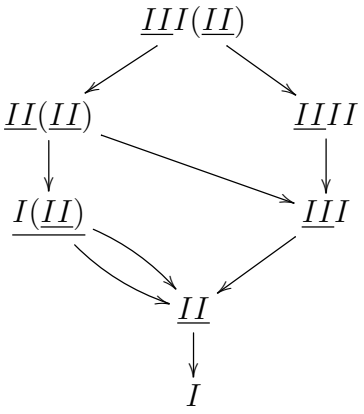
8. Consider the untyped lambda term:

$$M_8 := III(II)$$

In this we have as always that $I := (\lambda x.\, x)$.

Now answer the following two questions:

(a) Give the reduction graph of this term. If there are multiple ways to reduce a term to some other term, indicate this with multiple arrows.



In each term we have underlined the redexes.

(b) Compute M_8^* and $(M_8^*)^*$ and $((M_8^*)^*)^*$. Just giving the answers is enough, you do not need to explain how you obtained them.
We have:

$$\begin{aligned} I^* &= I \\ (II)^* &= ((\lambda x.x)I)^* = I^* = I \\ (III)^* &= (II)^* I^* = II \end{aligned}$$

and therefore:

$$\begin{aligned} M_8^* &= (III)^*(II)^* = III \\ (M_8^*)^* &= (III)^* = II \\ ((M_8^*)^*)^* &= (II)^* = I \end{aligned}$$

The definition of M^* is:

$$\begin{aligned} x^* &= x \\ (\lambda x.\, M)^* &= \lambda x.\, M^* \\ (MN)^* &= \begin{cases} P^*[x := N^*] & \text{if } M = \lambda x.\, P \\ M^* N^* & \text{otherwise} \end{cases} \end{aligned}$$

9. Consider the lambda term:

$$\lambda x.\, (\lambda f.\, f(f(fx)))\, (\lambda y.\, (\lambda z.\, z)\, y)$$

This term is typable in simple type theory. A typed version is:

$$\lambda x : a.\, (\lambda f : a \to a.\, f(f(fx)))\, (\lambda y : a.\, (\lambda z : a.\, z)\, y)$$

or in Coq notation:

```
fun x : a => (fun f : a -> a => f (f (f x)))
  (fun y : a => (fun z : a => z) y)
```

Now answer the following three questions:

(a) Indicate what the redexes in this term are. You can do this either in the untyped or in the typed version of the term.

$$\lambda x.\ \underbrace{(\lambda f.\, f(f(fx)))\,(\lambda y.\,\underbrace{(\lambda z.\, z)\, y}\,)}$$

The redexes in the typed term are in the corresponding places.

(b) For each of these redexes give its height.

The types of the lambda abstractions in the redexes of the typed term are respectively:

$$\begin{array}{ll} \lambda f.\, f(f(fx)) & (a \to a) \to a \\ \lambda z.\, z & a \to a \end{array}$$

This means that the heights of the redexes are:

$$\begin{array}{ll} (\lambda f.\, f(f(fx)))\,(\lambda y.\,(\lambda z.\, z)\, y) & 2 \\ (\lambda z.\, z)\, y & 1 \end{array}$$

(c) Indicate which redex or redexes may be contracted according to the reduction strategy from Turing's proof of weak normalization for simple type theory.

The maximal height is 2, so Turing's strategy is to pick a redex of height 2 that does not contain another redex of height 2. In this case the only redex that satisfies this is, is the only redex of height 2:

$$(\lambda f.\, f(f(fx)))\,(\lambda y.\,(\lambda z.\, z)\, y)$$

(After reducing this redex the term becomes:

$$\lambda x.\ (\lambda y.\,\underbrace{(\lambda z.\, z)\, y}\,)\,((\lambda y.\,\underbrace{(\lambda z.\, z)\, y}\,)\,((\lambda y.\,\underbrace{(\lambda z.\, z)\, y}\,)\, x))$$

which has *more* redexes. But they are of smaller height!)

The height of a redex $(\lambda x : A.\, M)\, N$ is the height of the type of $(\lambda x : A.\, M)$. The height function h is defined on types by:

$$\begin{array}{rl} h(a) = 0 & \text{for atomic types } a \\ h(A \to B) = \max(h(A) + 1, h(B)) & \end{array}$$

which implies that:

$$h(A_1 \to \cdots \to A_n \to a) = \max(h(A_1), \ldots, h(A_n)) + 1$$