Type Theory and Coq 2022-2023

16-11-2022
10:30-13:00

The next four exercises are concerned with simple type theory and propositional logic.

1. Give a type derivation in Church-style simple type theory of the judgment:

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

You may abbreviate contexts in the style:

$$\Gamma_1 := x : (a \to b) \to a \to b,\ y : a \to b$$

$$\dfrac{\dfrac{\dfrac{}{\Gamma_1,\, z : a \vdash x : (a \to b) \to a \to b} \quad \dfrac{}{\Gamma_1,\, z : a \vdash y : a \to b}}{\Gamma_1,\, z : a \vdash xy : a \to b} \quad \dfrac{}{\Gamma_1,\, z : a \vdash z : a}}{\dfrac{\Gamma_1,\, z : a \vdash xyz : b}{\dfrac{}{\Gamma_1 \vdash x : (a \to b) \to a \to b} \quad \dfrac{\Gamma_1,\, z : a \vdash xyz : b}{\Gamma_1 \vdash (\lambda z : a.\, xyz) : a \to b}}}$$
$$\Gamma_1 \vdash x\,(\lambda z : a.\, xyz) : a \to b$$

2. Consider the following Coq script:

```
Parameter a b c : Prop.

Lemma S : (a -> b -> c) -> (a -> b) -> a -> c.
intros x y z. apply x.
- apply z.
- apply y. apply z.
Qed.
```

Give the (Church-style) term that Coq builds with this script, i.e., the term that will be printed by the command:

```
Print S.
```

You may write this term both in mathematical style or using Coq syntax, whatever you prefer.

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

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

3. Apply the PT algorithm to determine whether the following lambda term is typable in Curry-style simple type theory:

$$\lambda xy.x\,(\lambda z.zyx)$$

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

We annotate the term with type variables:

$$\underbrace{\lambda x^a y^b.\, x^a\,(\lambda z^c.\,\overbrace{\underbrace{z^c y^b}_{e}\, x^a}^{f})}_{d} \quad : a \to b \to d$$

This gives the equations:

$$\begin{aligned} a &= (c \to e) \to d \\ f &= a \to e \\ c &= b \to f \end{aligned}$$

We first substitute a everywhere:

$$\begin{aligned} a &= (c \to e) \to d \\ f &= ((c \to e) \to d) \to e \\ c &= b \to f \end{aligned}$$

Then we substitute f everywhere:

$$\begin{aligned} a &= (c \to e) \to d \\ f &= ((c \to e) \to d) \to e \\ c &= b \to ((c \to e) \to d) \to e \end{aligned}$$

As the equation for c now has c on the right hand side, the unification problem fails, and the term is *not* typable.

4. Consider the following formula of propositional logic:

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

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

$$\dfrac{\dfrac{\dfrac{[(a \wedge b) \wedge c^x]}{a \wedge b}\, El\wedge}{a}\, El\wedge \qquad \dfrac{\dfrac{\dfrac{[(a \wedge b) \wedge c^x]}{a \wedge b}\, Er\wedge}{b} \qquad \dfrac{[(a \wedge b) \wedge c^x]}{c}\, Er\wedge}{b \wedge c}\, I\wedge}{\dfrac{a \wedge (b \wedge c)}{(a \wedge b) \wedge c \to a \wedge (b \wedge c)}\, I[x]\to}$$

(b) Give the proof term that corresponds to this proof. In this term you may use the constants:

$$
\begin{array}{lcl}
a & : & * \\
b & : & * \\
c & : & * \\
\mathsf{and} & : & * \to * \to * \\
\mathsf{conj} & : & \Pi\, a : *.\ \Pi\, b : *.\ a \to b \to \mathsf{and}\ a\ b \\
\pi_1 & : & \Pi\, a : *.\ \Pi\, b : *.\ \mathsf{and}\ a\ b \to a \\
\pi_2 & : & \Pi\, a : *.\ \Pi\, b : *.\ \mathsf{and}\ a\ b \to b
\end{array}
$$

Or, in Coq syntax:

```
a     : Prop
b     : Prop
c     : Prop
and   : Prop -> Prop -> Prop
conj  : forall a b : Prop, a -> b -> and a b
proj1 : forall a b : Prop, and a b -> a
proj2 : forall a b : Prop, and a b -> b
```

You may write this term both in mathematical style or using Coq syntax, whatever you prefer.

$$
\begin{array}{l}
\lambda x : (\mathsf{and}\ (\mathsf{and}\ a\ b)\ c). \\
\quad \mathsf{conj}\ a\ (\mathsf{and}\ b\ c) \\
\qquad (\pi_1\ a\ b\ (\pi_1\ (\mathsf{and}\ a\ b)\ c\ x)) \\
\qquad (\mathsf{conj}\ b\ c \\
\qquad\quad (\pi_2\ a\ b\ (\pi_1\ (\mathsf{and}\ a\ b)\ c\ x)) \\
\qquad\quad (\pi_2\ (\mathsf{and}\ a\ b)\ c\ x))
\end{array}
$$

```
fun (x : and (and a b) c) =>
  conj a (and b c)
    (proj1 a b (proj1 (and a b) c x))
    (conj b c
      (proj2 a b (proj1 (and a b) c x))
      (proj2 (and a b) c x))
```

The next two exercises are concerned with dependent types and predicate logic.

5. Consider the following natural deduction proof in minimal predicate logic:

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

This proof contains a detour.

(a) Explain what the detour in this proof is.

A detour is an introduction rule that is directly followed by an elimination rule for the same connective. In this case it is the $I\forall$ rule that is followed by the $E\forall$ rule.

(b) Give the λP proof term that corresponds to this proof, in which you may use the constants:

$$\begin{array}{lcl} D & : & * \\ p & : & D \to * \\ q & : & D \to * \\ f & : & D \to D \to D \\ g & : & D \to D \end{array}$$

Or, in Coq syntax:

```
D : Set
p : D -> Prop
q : D -> Prop
f : D -> D -> D
g : D -> D
```

You may write this term both in mathematical style or using Coq syntax, whatever you prefer.

$$\lambda x : D.\, \lambda H_1 : (\Pi y : D.\, p\, y).\, \lambda H_2 : (\Pi y : D.\, p\, y \to q\, y).\, \underline{(\lambda y : D.\, H_2\, y\, (H_1 y))\, (f\, x\, (g\, x))}$$

```
fun (x : D) (H1 : forall y : D, p y)
            (H2 : forall y : D, p y -> q y) =>
  (fun y : D => H2 y (H1 y)) (f x (g x))
```

(c) Indicate the subterm that is the redex that corresponds to the detour.

The redex has been underlined in the term.

(d) Give the normal form of the proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{[\forall y.\, p(y) \to q(y)^{H_2}]}{p(f(x,g(x))) \to q(f(x,g(x)))}\, E\forall \qquad \dfrac{[\forall y.\, p(y)^{H_1}]}{p(f(x,g(x)))}\, E\forall}{q(f(x,g(x)))}\, E\to}{\big(\forall y.\, p(y) \to q(y)\big) \to q(f(x,g(x)))}\, I[H_2]\to}{\big(\forall y.\, p(y)\big) \to \big(\forall y.\, p(y) \to q(y)\big) \to q(f(x,g(x)))}\, I[H_1]\to}{\forall x.\, \Big(\big(\forall y.\, p(y)\big) \to \big(\forall y.\, p(y) \to q(y)\big) \to q(f(x,g(x)))\Big)}\, I\forall$$

6. Give derivations of the following two typing judgments. See page 9 for the typing rules of λP.

(a)

$$a : *,\ x : a \vdash a \to * : \Box$$

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

(b)

$$a : *,\ x : a,\ b : a \to * \vdash b\, x : *$$

In this second derivation you may replace instances of the first derivation by dots.

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

The next two exercises are concerned with polymorphic types and second order propositional logic.

7. Give a proof in second order propositional logic of the formula:

$$a \to \mathsf{or}_2\ a\, b$$

where we defined the impredicative version or_2 of disjunction by:

$$\mathsf{or}_2\ A\, B := \forall c.\, (A \to c) \to (B \to c) \to c$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[a \to c^y] \qquad [a^x]}{c}\, E{\to}}{(b \to c) \to c}\, I[z]{\to}}{(a \to c) \to (b \to c) \to c}\, I[y]{\to}}{\forall c.\, (a \to c) \to (b \to c) \to c}\, I\forall}{a \to \forall c.\, (a \to c) \to (b \to c) \to c}\, I[x]{\to}$$

8. In the context $b : *$, $t : b$ consider the Church-style $\lambda 2$ type:

$$\forall a.\, a \to b$$

(a) Write this type using PTS syntax, i.e., according to the grammar:

$$M ::= x \mid MM \mid \lambda x : M.\, M \mid \Pi x : M.\, M \mid s$$
$$s ::= * \mid \Box$$

$$\Pi a : *.\, \Pi x : a.\, b$$

(b) Give the $\lambda 2$ type of this type (its *kind*).

$$*$$

(c) Give an inhabitant of this type, in the given context.

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

(d) Give the typing judgment of this inhabitant. (Note that you only need to *state* the judgment, and do not need to *derive* it.)

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

The next two exercises are concerned with inductive types.

9. In Coq the natural numbers are defined by:

```
Inductive nat : Set :=
| O : nat
| S : nat -> nat.
```

Its dependent recursion principle `nat_rec` has type:

```
nat_rec :
  forall A : nat -> Set,
    A O ->
    (forall n : nat, A n -> A (S n)) ->
      forall n : nat, A n
```

We want to define the predecessor function `pred` with type:

```
pred : nat -> nat
```

The recursion equations that we want to capture are:

```
pred  O    = O
pred (S m) = m
```

In other words, for this function we define the predecessor of zero to be zero.

(a) Define this function `pred` using `Fixpoint` and `match`.

```
Fixpoint pred (n : nat) {struct n} : nat :=
  match n with
  | O => O
  | S m => m
  end.
```

(b) Define this function `pred` as an application of the recursor `nat_rec`.

```
Definition pred (n : nat) : nat :=
  nat_rec (fun n : nat => nat) O (fun (m : nat) (p : nat) => m)
    n.
```

10. (a) Define an inductive type `list` of polymorphic lists, with constructors `nil` and `cons`.

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

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

```
list_ind :
  forall (A : Set) (P : list A -> Prop),
    P (nil A) ->
    (forall (x : A) (l : list A), P l -> P (cons A x l)) ->
      forall l : list A, P l.
```

(c) Give the type of the non-dependent recursor `list_rec_nondep` of this type.

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

The next exercise is concerned with the Church-Rosser proof by Takahashi.

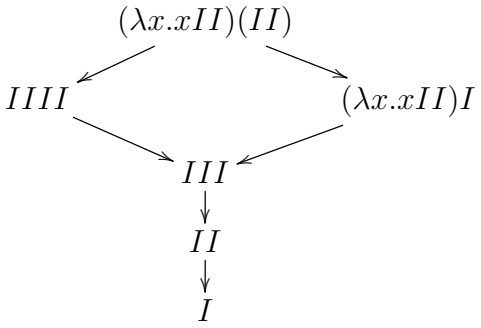
11. Consider the untyped lambda term:

$$(\lambda x.xII)(II)$$

where we abbreviate, as always:

$$I := \lambda x.x$$

(a) Give the full reduction graph for this term.



(b) For each term M in this graph, give the result of the full development M^*, as defined by:

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

We have:

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

and therefore:

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

The next exercise is concerned with the strong normalization proof using saturated sets.

12. (a) Give the recursive definition of the semantics $[\![A]\!]_\rho$ from the strong normalization proof for $\lambda 2$.

$$\begin{aligned}[\![a]\!]_\rho &:= \rho(a)\\ [\![A \to B]\!]_\rho &:= \{M \mid \forall N \in [\![A]\!]_\rho.\, MN \in [\![B]\!]_\rho\}\\ [\![\forall a.\, A]\!]_\rho &:= \bigcap_{X \in \mathsf{SAT}} [\![A]\!]_{\rho, a \mapsto X}\end{aligned}$$

(b) Explain what A, ρ and $[\![A]\!]_\rho$ are in this definition.

The argument A is any $\lambda 2$ type, ρ is any function that maps the $\lambda 2$ type variables to a saturated set of untyped lambda terms, and $[\![A]\!]_\rho$ is a saturated set of untyped lambda terms.