Type Theory and Rocq 2025-2026

24-10-2025
12:45-14:45

1. (a) Give an inhabitant of the simple type $a \to (a \to b) \to b$, as a term of Church-style simple type theory.

   (You should not give a type derivation for this term, later in the exam there will be another exercise that asks for a type derivation in simple type theory.)

   $$\lambda x : a.\, \lambda f : a \to b.\, fx$$

   (b) Give the proof in minimal propositional logic that corresponds to this term.

   $$\dfrac{\dfrac{\dfrac{[a \to b^f] \quad [a^x]}{b}\, E{\to}}{(a \to b) \to b}\, I[f]{\to}}{a \to (a \to b) \to b}\, I[x]{\to}$$

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

   No, the proof does not have a detour. There is no introduction rule directly followed by a corresponding elimination rule.

   (d) Give a Rocq version of this proof, using tactics. You may only use the tactics `intro`, `intros`, `apply`, `exact` and `assumption`. The proof script should be in the place of the dots in:

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

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

   ```
   intros x f.
   apply f.
   apply x.
   ```

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

   $$\lambda xy.x(\lambda z.xyz)$$

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

We annotate the term with type variables:

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

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

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

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

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

When we apply this substitution to the type $a \to b \to d$, this gives one of the principal types of the term:

$$((c \to e) \to c \to e) \to (c \to e) \to c \to e$$

3. Give the full reduction graph of the untyped lambda term

$$2\,\mathsf{I}\,\mathsf{K}$$

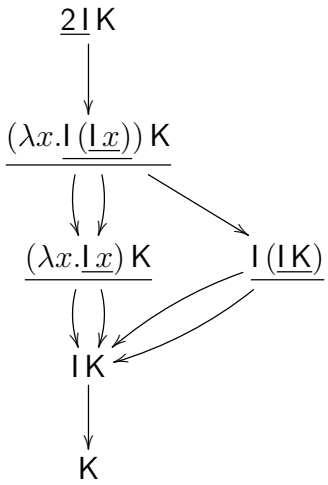
We use the customary abbreviations:

$$\begin{aligned} \mathsf{I} &:= \lambda x.x \\ \mathsf{K} &:= \lambda xy.x \\ 2 &:= \lambda fx.f(fx) \end{aligned}$$

Note that we do not reduce like in combinatory logic here. This means that, although the term 2 expects two arguments as a combinator, we reduce for example:

$$2\,\mathsf{I} \to_\beta \lambda x.\mathsf{I}\,(\mathsf{I}x)$$

If there are multiple beta steps between the same terms, draw this as multiple arrows. We recommend underlining each beta redex, to keep track of what is going on.



4. Consider the following proposition of predicate logic:

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

Now answer the following three sub-questions:

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

$$\dfrac{\dfrac{\dfrac{\dfrac{[\forall y.\, p(y)^H]}{p(x)}\, E\forall}{\exists z.\, p(z)}\, I\exists}{(\forall y.\, p(y)) \to \exists z.\, p(z)}\, I[H]\to}{\forall x.\, (\forall y.\, p(y)) \to \exists z.\, p(z)}\, I\forall$$

The only rule that has a variable condition is the $I\forall$ rule at the end. We should check that x is not free in any open assumptions. The rule is applied when there are no open assumptions, which means the variable condition is satisfied.

(b) Give the λP type that corresponds to this proposition. For this, the context will have to contain λP variables for the existential quantifier and for the corresponding introduction rule. Therefore, this type will need to be well-typed in the context:

$$\begin{array}{rcl} D & : & * \\ \mathsf{ex} & : & (D \to *) \to * \\ \mathsf{ex\_intro} & : & \prod P : D \to *.\ \prod x : D.\, Px \to \mathsf{ex}\, P \\ p & : & D \to * \end{array}$$

You are allowed to use mathematical notation for this type, or to use Rocq syntax.
(Note that we just ask for the type as an expression, and *not* for a λP type derivation.)

$$D \to (\prod y : D.\, py) \to \mathsf{ex}\,(\lambda z : D.\, pz)$$

In Rocq syntax this is:

```
D -> (forall y : D, p y) -> ex (fun z : D => p z)
```

(Alternatively, one can use the eta-reduct p in the place of $\lambda z : D.\, pz$.)

(c) Give the λP proof term that corresponds to the proof that you gave in sub-question (a), using the same context.
(Note that we just ask for the proof term as an expression, and *not* for a λP type derivation.)

$$\lambda x : D.\, \lambda H : (\prod y : D.\, py).\, \mathsf{ex\_intro}\,(\lambda z : D.\, pz)\, x\, (Hx)$$

In Rocq syntax this is:

```
fun (x : D) (H : forall y : D, p y) =>
  ex_intro (fun z : D => p z) x (H x)
```

5. Consider the following four preterms:

$$\lambda a : *.\, a$$
$$\lambda a : *.\, *$$
$$\prod a : *.\, a$$
$$\prod a : *.\, *$$

Or in Rocq syntax:

```
fun a : Prop => a
fun a : Prop => Prop
forall a : Prop, a
forall a : Prop, Prop
```

Now answer the following four sub-questions:

(a) Which of these preterms is well-typed in the calculus of constructions,
This exercise turned out to be somewhat ambiguous. The answer differs between the calculus of constructions $\lambda\mathsf{C}$ and the type theory of Rocq, the calculus of inductive constructions CIC. The first variant of notation and the phrasing of sub-question (a) suggests this is about $\lambda\mathsf{C}$. However, the second variant of the notation (Rocq syntax) suggests this is about

CIC. When grading, we considered an answer to either interpretation of the question to be correct.

In CIC, all four preterms are well-typed. However, in λC, the second preterm $\lambda a : *. *$ is not well-typed, as in the calculus of constructions $* \to \Box$ is not a valid type.

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

In λC:

$$\begin{aligned} \lambda a : *. a &: * \to * \\ \prod a : *. a &: * \\ \prod a : *. * \equiv * \to * &: \Box \end{aligned}$$

In Rocq:

```
   fun a : Prop => a  :  Prop -> Prop
fun a : Prop => Prop  :  Prop -> Type
  forall a : Prop, a  :  Prop
forall a : Prop, Prop  :  Type
```

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

In both variants of the question, the last two preterms of the four have a type that is a sort, so these are the types.

(d) Of the ones that are well-typed types: which are inhabited in an empty context? For the inhabited types, give an inhabitant.

The type $\prod a : *. a$ is not inhabited, as it is the impredicative encoding of falsity.

The type $\prod a : *. * \equiv * \to *$ is inhabited, as it is the type of the first preterm from our list, $\lambda a : *. a$.

6. (a) Give a type derivation in simple type theory of the judgment:

$$\vdash (\lambda x : a. x) : a \to a$$

$$\dfrac{\dfrac{}{x : a \vdash x : a}}{\vdash (\lambda x : a. x) : a \to a}$$

(b) Give a type derivation in the pure type system $\lambda\to$ of the judgment:

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

For the rules of $\lambda\to$ see page 8.

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

7. Consider the following Rocq definition of a type for Booleans:

```
Inductive bool : Set :=
| true : bool
| false : bool.
```

We want to define a function `if_then_else` that chooses which argument to return based on a Boolean. It has type:

```
if_then_else
     : forall A : Set, bool -> A -> A -> A
```

Now answer the following three sub-questions:

(a) Define the function `if_then_else` using `Definition` and `match`.

```
Definition if_then_else (A : Set)
    (b : bool) (x y : A) : A :=
  match b with
  | true => x
  | false => y
  end.
```

(b) Give the type of the dependent recursion principle `bool_rec` of the type `bool`.

```
forall P : bool -> Set,
  P true ->
  P false ->
  forall b : bool, P b
```

(c) Define the function `if_then_else` using an application of `bool_rec` to appropriate arguments.

```
Definition if_then_else (A : Set)
    (b : bool) (x y : A) : A :=
  bool_rec (fun _ : bool => A) x y b
```

8. We can also use an impredicative encoding in $\lambda 2$ of the Booleans, which is defined as:

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

Now answer the following two sub-questions:

(a) Give appropriate definitions of constants:

$$\begin{aligned} \mathsf{true}_2 &: \mathsf{bool}_2 \\ \mathsf{false}_2 &: \mathsf{bool}_2 \\ \mathsf{if\_then\_else}_2 &: \prod a : *.\, \mathsf{bool}_2 \to a \to a \to a \end{aligned}$$

$$\begin{aligned} \mathsf{true}_2 &:= \lambda a : *.\, \lambda x : a.\, \lambda y : a.\, x \\ \mathsf{false}_2 &:= \lambda a : *.\, \lambda x : a.\, \lambda y : a.\, y \\ \mathsf{if\_then\_else}_2 &:= \lambda a : *.\, \lambda b : \mathsf{bool}_2.\, \lambda x : a.\, \lambda y : a.\, b\ a\ x\ y \end{aligned}$$

Alternatively, one can use the eta-reduct in the last definition:

$$\mathsf{if\_then\_else}_2 := \lambda a : *.\, \lambda b : \mathsf{bool}_2.\, b\ a$$

(b) Give the two reductions that show that $\mathsf{if\_then\_else}_2$ behaves like an eliminator. For both, you should only give the start and end terms, and not all the intermediate beta steps.

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

The intermediate steps, which should not be given in the answer, are:

$$\begin{aligned} \mathsf{if\_then\_else}_2\ A\ \mathsf{true}_2\ M\ N &\to_\beta (\lambda b : \mathsf{bool}_2.\, \lambda x : A,\ \lambda y : A.\, b\ A\ x\ y)\ \mathsf{true}_2\ M\ N \\ &\to_\beta (\lambda x : A,\ \lambda y : A.\, \mathsf{true}_2\ A\ x\ y)\ M\ N \\ &\to_\beta (\lambda y : A.\, \mathsf{true}_2\ A\ M\ y)\ N \\ &\to_\beta \mathsf{true}_2\ A\ M\ N \\ &\to_\beta (\lambda x : A.\, \lambda y : A.\, x)\ M\ N \\ &\to_\beta (\lambda y : A.\, M)\ N \\ &\to_\beta M \end{aligned}$$

$$\begin{aligned} \mathsf{if\_then\_else}_2\ A\ \mathsf{false}_2\ M\ N &\to_\beta (\lambda b : \mathsf{bool}_2.\, \lambda x : A,\ \lambda y : A.\, b\ A\ x\ y)\ \mathsf{false}_2\ M\ N \\ &\to_\beta (\lambda x : A,\ \lambda y : A.\, \mathsf{false}_2\ A\ x\ y)\ M\ N \\ &\to_\beta (\lambda y : A.\, \mathsf{false}_2\ A\ M\ y)\ N \\ &\to_\beta \mathsf{false}_2\ A\ M\ N \\ &\to_\beta (\lambda x : A.\, \lambda y : A.\, y)\ M\ N \\ &\to_\beta (\lambda y : A.\, y)\ N \\ &\to_\beta N \end{aligned}$$

9. Consider the proof of strong normalization of simply typed lambda calculus, in which the types are mapped to sets of untyped lambda terms.

   Now answer the following three sub-questions:

   (a) Give the definition of the set $[\![a \to a]\!]$, where a is a base type.

   The definition of the semantics of simple types as sets of untyped lambda terms is:

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

   This means that $[\![a \to a]\!]$ consists of the terms F for which it holds that for all strongly normalizing terms N, the term FN is also strongly normalizing.

   (b) Show that this set has the property $[\![a \to a]\!] \neq \varnothing$.

   For all types A, the set $[\![A]\!]$ contains all terms of the form $xN_1 \ldots N_k$, where x is any variable and all N_i are strongly normalizing. Specifically, this means that the term x is an element of $[\![a \to a]\!]$.

   As another example, the term $\lambda x.x$ is typable (à la Curry) with $a \to a$ in the empty context, and from the main proposition about this semantics it follows that $\lambda x.x \in [\![a \to a]\!]$.

   (c) Show that this set has the property $[\![a \to a]\!] \neq \mathsf{SN}$.

   The term $\omega := \lambda x.xx$ is strongly normalizing, but $\omega\omega$ is not strongly normalizing, and by applying the criterion from sub-question (a), it follows that $\omega \in \mathsf{SN}$ but $\omega \notin [\![a \to a]\!]$.

   (In these two last sub-questions, you may use the lemma and the proposition about the semantics from the lecture.)