Resit Type Theory and Coq 2018-2019

10-07-2019

1. This exercise is about *simple type theory* and *propositional logic.*

   (a) Give a proof in minimal propositional logic that contains a *detour* of the formula:

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

   (Note: if you do not know what a detour is, or you cannot find a proof with a detour, you can get partial points for a proof of this formula without a detour.)

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

   (b) Give the proof term in (Church-style) simple type theory of the proof from the previous subexercise, which is a lambda term with type:

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

   $$\lambda f : a \to a \to b.\, \lambda x : a.\, (\lambda g : a \to b.\, gx)(fx)$$

   (c) Give the normal form of the term from the previous subexercise. Explain your answer.

   By contracting the sole β redex $(\lambda g : a \to b.\, gx)(fx) \to_\beta fxx$, one gets the normal form (= term without redexes):

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

   (d) Give a derivation of the typing judgement of the term in normal form from the previous subexercise.

   Using the abbreviation

   $$\Gamma := f : a \to a \to b,\, x : a$$

we get:

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{\Gamma \vdash f : a \to a \to b} \quad \overline{\Gamma \vdash x : a}}{\Gamma \vdash fx : a \to b} \qquad \overline{\Gamma \vdash x : a}}{\Gamma \vdash fxx : b}}{f : a \to a \to b \vdash (\lambda x : a.\, fxx) : a \to b}}{\vdash (\lambda f : a \to a \to b.\, \lambda x : a.\, fxx) : (a \to a \to b) \to a \to b}$$

(e) Give the most general type of the lambda term:

$$\lambda xyz.\, x(yzz)y$$

You do not need to show that this is the most general type, or how you obtained it, just giving the type is sufficient.

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

2. This exercise is about *dependent types* and *predicate logic*.

(a) Give a proof in minimal predicate logic of the formula:

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

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

(b) Give the proof term in λP of the proof from the previous subexercise. Use the type D for the domain that is being quantified over.

$$\lambda H : (\Pi x : D.\, \Pi y : D.\, rxy).\, \lambda x : D.\, Hxx$$

(c) Give the full λP typing judgement (i.e., including the λP context) of the term in normal form from the previous subexercise.
(Note: you do *not* need to give the *derivation* of this judgment.)

$$\begin{aligned} D : *, r : D \to D \to * \vdash {} & \lambda H : (\Pi x : D.\, \Pi y : D.\, rxy).\, \lambda x : D.\, Hxx \\ & : (\Pi x : D.\, \Pi y : D.\, rxy) \to \Pi x : D.\, rxx \end{aligned}$$

(d) Give the four proof rules of minimal predicate logic, including variable condition(s).

$$\dfrac{\begin{matrix}[A^H]\\ \vdots \\ B\end{matrix}}{A \to B}\, I[H]{\to} \qquad\qquad \dfrac{\begin{matrix}\vdots \\ A \to B\end{matrix} \qquad \begin{matrix}\vdots \\ A\end{matrix}}{B}\, E{\to}$$

$$\dfrac{\begin{matrix}\vdots \\ A\end{matrix}}{\forall x.\, A}\, I\forall \qquad\qquad \dfrac{\begin{matrix}\vdots \\ \forall x.\, A\end{matrix}}{A[x := t]}\, E\forall$$

The variable condition: in the $I\forall$ rule, the variable x should not be free in any open assumptions.

And in the $I[H]{\to}$ rule, the assumption $[A^H]$ is allowed to occur zero, one or more times in the subderivation.

(e) What is the formula in minimal predicate logic that has as proof term:

$$\lambda H_1 : (\Pi x{:}D.\, px \to qx).\, \lambda x : D.\, \lambda H_2 : (qx \to \bot).\, \lambda H_3 : px.\, H_2(H_1 x H_3)$$

In this term $\bot$ is a type that corresponds to an atomic formula $\bot$. In your answer you may abbreviate formulas $A \to \bot$ as $\neg A$, but this is not required.

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

3. This exercise is about *polymorphism* and *second order propositional logic.*

(a) Give a proof in minimal second order propositional logic of the formula:

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

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

(b) Give the proof term in $\lambda 2$ for the proof from the previous subexercise.

$$\lambda a : *.\, \lambda x : a.\, \lambda b : *.\, \lambda y : b.\, \lambda z : b.\, y$$

Or, in other notation:

$$\Lambda a.\, \lambda x : a.\, \Lambda b.\, \lambda y : b.\, \lambda z : b.\, y$$

(c) Call the proof term of the previous subexercise M, and its type A. Is the term $MAMAM$ well-typed or not? If so, what is its type? Or if not, why not? Explain your answer.

Yes, it is well-typed. We have:

$$\begin{aligned} M &:= \lambda a : *.\, \lambda x : a.\, \lambda b : *.\, \lambda y : b.\, \lambda z : b.\, y \\ A &:= \Pi a : *.\, a \to \Pi b : *.\, b \to b \to b \end{aligned}$$

So we have the typings:

$$\begin{aligned} A &: * \\ M &: \Pi a : *.\, a \to \Pi b : *.\, b \to b \to b \\ MA &: A \to \Pi b : *.\, b \to b \to b \\ MAM &: \Pi b : *.\, b \to b \to b \\ MAMA &: A \to A \to A \\ MAMAM &: A \to A \end{aligned}$$

Or, in the other notation:

$$\begin{aligned} M &:= \Lambda a.\, \lambda x : a.\, \Lambda b.\, \lambda y : b.\, \lambda z : b.\, y \\ A &:= \forall a. a \to \forall b.\, b \to b \to b \end{aligned}$$

with typings:

$$\begin{aligned} A &: * \\ M &: \forall a.\, a \to \forall b.\, b \to b \to b \\ MA &: A \to \forall b.\, b \to b \to b \\ MAM &: \forall b.\, b \to b \to b \\ MAMA &: A \to A \to A \\ MAMAM &: A \to A \end{aligned}$$

(d) One can define *lists* over a given type A impredicatively in $\lambda 2$ as:

$$\mathsf{list}_A := \Pi l : *.\, l \to (A \to l \to l) \to l$$

Give definitions of nil_A and cons_A with types:

$$\begin{aligned} \mathsf{nil}_A &: \mathsf{list}_A \\ \mathsf{cons}_A &: A \to \mathsf{list}_A \to \mathsf{list}_A \end{aligned}$$

$$\begin{aligned} \mathsf{nil}_A &:= \lambda l : *.\, \lambda n : l.\, \lambda c : (A \to l \to l).\, n \\ \mathsf{cons}_A &:= \lambda h : A.\, \lambda t : \mathsf{list}_A.\, \lambda l : *.\, \lambda n : l.\, \lambda c : (A \to l \to l).\, ch(tlnc) \end{aligned}$$

(e) Explain why abstracting the type of the elements in the list by defining

$$\mathsf{list} := \lambda a : *.\, \Pi l : *.\, l \to (a \to l \to l) \to l$$

which would have type

$$\mathsf{list} : * \to *$$

is not allowed in $\lambda 2$.

The type $* \to *$ is not allowed in $\lambda 2$, as it requires the 'rule' $(\square, \square, \square)$ that is not available in $\lambda 2$.

4. This exercise is about the typing rules of *pure type systems* and the *lambda cube.*

For the typing rules of the lambda cube, see page 10 of this exam.

(a) Give a derivation in $\lambda\underline{\omega}$ of the judgement:

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

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash * : \Box}}{b : * \vdash b : *} \quad \dfrac{\dfrac{}{\vdash * : \Box} \quad \dfrac{}{\vdash * : \Box}}{b : * \vdash * : \Box}}{b : *,\, a : *,\, \vdash b : *} \quad \dfrac{\dfrac{\dfrac{}{\vdash * : \Box} \quad \dfrac{}{\vdash * : \Box}}{b : * \vdash * : \Box} \quad \dfrac{\dfrac{\dfrac{}{\vdash * : \Box} \quad \dfrac{}{\vdash * : \Box}}{b : * \vdash * : \Box} \quad \dfrac{\dfrac{}{\vdash * : \Box} \quad \dfrac{}{\vdash * : \Box}}{b : * \vdash * : \Box}}{b : *,\, a : * \vdash * : \Box}}{b : * \vdash * \to * : \Box}}{b : * \vdash \lambda a : *.\, b : * \to *}$$

Alternatively, one can weaken the judgement $b : * \vdash * \to * : \Box$ first:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash * : \Box}}{b : * \vdash b : *} \quad \dfrac{\dfrac{}{\vdash * : \Box} \quad \dfrac{}{\vdash * : \Box}}{b : * \vdash * : \Box}}{b : *,\, a : * \vdash b : *} \quad \dfrac{\dfrac{\dfrac{}{\vdash * : \Box} \quad \dfrac{\dfrac{}{\vdash * : \Box} \quad \dfrac{}{\vdash * : \Box}}{a : * \vdash * : \Box}}{\vdash * \to * : \Box} \quad \dfrac{}{\vdash * : \Box}}{b : * \vdash * \to * : \Box}}{b : * \vdash \lambda a : *.\, b : * \to *}$$

(b) Disjunction can be impredicatively defined as:

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

List the systems of the lambda cube in which this term is typable. Explain your answer.

Like in the corresponding exercise from the first exam, the systems in which this term is typable are $\lambda\omega$ and $\lambda P\omega = \lambda C$.

(c) The systems of the lambda cube all satisfy the *Church-Rosser* property. State what this means.

If $M \twoheadrightarrow_\beta M_1$ and $M \twoheadrightarrow_\beta M_2$, there is an N with $M_1 \twoheadrightarrow_\beta N$ and $M_2 \twoheadrightarrow_\beta N$.

(d) The systems of the lambda cube all satisfy the property of *decidability of type checking*. State what this means.

Given a precontext Γ and preterms M and A, it is decidable whether the judgment $\Gamma \vdash M : A$ is derivable.

5. This exercise is about *inductive types* and *recursive functions.*

(a) We want a datatype for lists of Booleans. Define an inductive type `listb` of type `Set` using Coq syntax for this datatype. An example of an element of this type might be:

```
  Consb true (Consb false (Consb true Nilb))
```

Remember that the Coq type for Booleans is called `bool`.

```
Inductive listb : Set :=
  Nilb : listb | Consb : bool -> listb -> listb.
```

(b) Give the type of the recursion principle `listb_rec` for the inductive type from the previous subexercise.

```
forall A : listb -> Set,
  A Nilb ->
  (forall (b : bool) (l : listb), A l -> A (Consb b l)) ->
  forall l : listb, A l
```

(c) Define a function `count_trues` that counts the number of `true`s using `Fixpoint` and `match`. The count for the example list should be two, as there are two `true`s in this list. Remember that the Coq type for natural numbers is called `nat`, and the function for addition on natural numbers is called `plus`.

```
Fixpoint count_trues (l : listb) {struct l} : nat :=
  match l with
  | Nilb => O
  | Consb true l' => S (count_trues l')
  | Consb false l' => count_trues l'
  end.
```

(d) Define the same function using `listb_rec`.

```
Definition count_trues' : listb -> nat :=
  listb_rec (fun _ => nat)
    O (fun b _ n => if b then S n else n).
```

The term `if b then` . . . `else` . . . also can be written as:

```
  match b with
  | true => ...
  | false => ...
  end
```

(e) Define an *inductive* predicate

$$\texttt{all\_true} : \texttt{listb} \to \texttt{Prop}$$

that states that the list only consist of `true` elements. For example the following type should be inhabited:

```
  all_true (Consb true (Consb true Nilb))
```

```
Inductive all_true : listb -> Prop :=
| all_true_Nilb : all_true Nilb
| all_true_Consb : forall l : listb,
    all_true l -> all_true (Consb true l).
```

(f) Give a definition of inequality $\leq$ on natural numbers as an inductively defined relation.

```
Inductive le (n : nat) : nat -> Prop :=
| le_n : le n n
| le_S : forall m : nat, le n m -> le n (S m).
```

6. This exercise is about *guarded type theory*.

(a) In recursive definitions over inductive types, Coq requires *structural recursion* to enforce *strong normalization* of the reduction relation. Name the counterparts of 'structural recursion' and 'normalization' for *co*-inductive types in Coq, and explain what the terms for these counterparts mean.

The counterpart of 'structural recursion' is 'guardedness', and the counterpart of 'normalization' is 'productivity'.

Guardedness means that a recursive call only is allowed under a constructor. Productivity means that after a finite computation always an extra constructor will be produced.

(b) The counterpart of the natural numbers as a coinductive type in Coq would be:

```
CoInductive conat : Set :=
  O : conat | S : conat -> conat.
```

Use `CoFixpoint` to define an element of type `conat` that does not have a counterpart in the inductive natural numbers `nat`.

```
CoFixpoint S_omega : conat := S S_omega.
```

This corresponds to the 'infinite' term $S(S(S(S \cdots)))$.

We will now look at a guarded type theory (in Curry-style). The syntax of the types and terms and 'clock contexts' of this theory is:

$$\begin{aligned}
A &::= a \mid A \to A \mid \mathbb{1} \mid A + A \mid A \times A \mid \mu a.A \mid \triangleright A \mid \Box A \\
M &::= x \mid \lambda x.M \mid M M \mid \\
&\qquad \star \mid \mathsf{inl}\, M \mid \mathsf{inr}\, M \mid \mathsf{case}\, M\, \mathsf{of}\, x.M; x.M \mid (M, M) \mid \mathsf{fst}\, A \mid \mathsf{snd}\, A \\
&\qquad \mathsf{next}\, M \mid M \circledast M \mid \mathsf{box}\, M \mid \mathsf{unbox}\, M \mid \mathsf{force}\, M \mid \\
&\qquad \mathsf{cons}_{\mu a.A}\, M \mid \mathsf{primrec}_{\mu a.A}\, M \mid \mathsf{dfix}\, M \\
\Delta &::= \varnothing \mid \kappa
\end{aligned}$$

In this a and x are respectively type and term variables.

Unlike in the paper by Niccolò and Niels that we have studied in the course, we here leave functions related to 'weakening' of clock contexts implicit, so we do not explicitly write $\uparrow$, up or down.

(c) We define two types in this system:

$$\mu a.\, \mathbb{1} + a$$
$$\mu a.\, \mathbb{1} + \triangleright a$$

Explain what these two types represent, and what is the difference between them.

The first type is the standard way to define the type of natural numbers using μ. The second type is the coinductive counterpart to the natural numbers that we already saw in the previous subexercise.

(d) Complete the typing rule of dfix by filling in the dots in the rule:

$$\frac{\Gamma \vdash_\kappa M : \ \ldots}{\Gamma \vdash_\kappa \mathsf{dfix}\, M : \triangleright A}$$

$$\frac{\Gamma \vdash_\kappa M : \triangleright A \to A}{\Gamma \vdash_\kappa \mathsf{dfix}\, M : \triangleright A}$$

(e) If we use the abbreviation $\overline{\mathbb{N}} := \mu a.\, \mathbb{1} + \triangleright a$ we can define a function

$$S : \triangleright\overline{\mathbb{N}} \to \overline{\mathbb{N}}$$

Use this function together with dfix to define a term S^ω with

$$S^\omega : \overline{\mathbb{N}}$$

that corresponds to the 'infinite' term $S(S(S(S \cdots)))$.

$$S^\omega := S\,(\mathsf{dfix}\, S)$$

Incidentally, it is not required for the exercise, but the definition of S is:

$$S := \lambda x.\, \mathsf{cons}_{\overline{\mathbb{N}}}\,(\mathsf{inr}\, x)$$