Type Theory and Coq 2016-2017

27-06-2017

The following exercises are about the *simply typed lambda calculus* ($\lambda\!\to$, Simple Type Theory) and *minimal propositional logic*.

1. (a) Give a proof in minimal propositional logic of the formula

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

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

   (b) Are there any detours in your proof? Explain your answer.

   No, there are no detours in this proof. A detour is an introduction rule immediately followed by a corresponding elimination rule for the same connective, and this proof does not have that.

   (c) Give the proof term of Church-style simply typed lambda calculus that corresponds to your proof under the Curry-Howard isomorphism.

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

   (d) Give the full type derivation of the proof term from the previous subexercise.

   We use the abbreviation:

$$\Gamma_{xy} := x : a \to (b \to a) \to c,\ y : a$$

   The type derivation then becomes:

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

2. (a) Is the lambda term

$$\lambda xy.\, yx\,(\lambda z.\, xz)$$

typable in Curry-style simply typed lambda calculus? If so, give its principal type. Explain how you obtained your answer.

We annotate the term with type variables:

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

We then need to solve the equations:

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

A solution for this is:

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

Substituting this in the type of the term

$$a \to b \to d$$

gives

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

which will be one of the principal types.

(b) Is the lambda term

$$\lambda xy.\, yx\,(\lambda z.\, yz)$$

typable in Curry-style simply typed lambda calculus? If so, give its principal type. Explain how you obtained your answer.

We annotate the term with type variables:

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

We then need to solve the equations:

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

This leads to the equation:

$$a \to (c \to e) \to d = c \to e$$

And that gives the equations:

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

In this last equation the e occurs on both sides of the equation in a way makes the equation unsolvable. Therefore the term is not typable in Curry-style simply typed lambda calculus.

(c) Give a type of

$$\lambda x.\, x$$

in Curry-style simply typed lambda calculus which is *not* its principal type.

A principal type of this term is $a \to a$. An non-principal type for this term is for instance

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

(d) One speaks of 'the' principal type of a term, while a pricipal type is not really unique. If A and B are principal types of a term M, then how are A and B related?

They are identical up to renaming of type variables.

3. Simply typed lambda calculus has the properties SR, CR, WN and SN.

(a) What is the property of *Subject Reduction* (SR)?

If $\Gamma \vdash M : A$ and $M \to_\beta M'$ then also $\Gamma \vdash M' : A$.

(b) What is the *Church-Rosser* property (CR)?

If $M \twoheadrightarrow_\beta N$ and $M \twoheadrightarrow_\beta N'$ there is an P with $N \twoheadrightarrow_\beta P$ and $N' \twoheadrightarrow_\beta P$. This is also called *confluence*.

(c) What is the property of *Weak Normalization* (WN)?

If $\Gamma \vdash M : A$, then there exist terms M_1, M_2, ..., M_k, such that $M \to_\beta M_1 \to_\beta M_2 \to_\beta \cdots \to_\beta M_k$, and where M_k is in *normal form* (it cannot reduce any further).

(d) What is the property of *Strong Normalization* (SN)?

If $\Gamma \vdash M : A$, then there does not exist an infinite sequence of terms M_1, M_2, ..., such that $M \to_\beta M_1 \to_\beta M_2 \to_\beta \ldots$

Note that we do not ask you to indicate how these properties are proved. You should just state what they are.

The following exercises are about *dependently typed lambda calculus* (λP) and *minimal predicate logic*.

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

$$(\forall x.\, P(x) \to Q(x)) \to (\forall x.\, \neg Q(x) \to \neg P(x))$$

where $\neg A$ is defined to be $A \to \bot$, and in which $\bot$ is an atomic proposition for which we have no further rules (else the predicate logic would not be *minimal*).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\forall x.\, P(x) \to Q(x)^{H}]}{P(x) \to Q(x)}\, E\forall \quad \dfrac{}{[P(x)^{H_1}]}}{Q(x)}\, E\to \quad \dfrac{}{[\neg Q(x)^{H_0}]}}{\bot}\, E\to}{\neg P(x)}\, I[H_1]\to}{\neg Q(x) \to \neg P(x)}\, I[H_0]\to}{\forall x.\, \neg Q(x) \to \neg P(x)}\, I\forall}{(\forall x.\, P(x) \to Q(x)) \to (\forall x.\, \neg Q(x) \to \neg P(x))}\, I[H]\to$$

(b) Give the proof term of λP that corresponds to your proof under the Curry-Howard isomorphism. Call the type that corresponds to the domain of quantification 'D'.

$$\lambda H : (\Pi x : D.\, Px \to Qx).\, \lambda x : D.\, \lambda H_0 : \neg Qx.\, \lambda H_1 : Px.\, H_0(HxH_1)$$

(c) Give the full λP typing judgment (including the context, and with $\bot$ in the context too) of the typing of this term. Note that you do not need to give the *derivation* of this judgment.

$$\begin{gathered}\bot : *,\ D : *,\ P : (D \to *),\ Q : (D \to *) \vdash \\ \lambda H : (\Pi x : D.\, Px \to Qx).\, \lambda x : D.\, \lambda H_0 : \neg Qx.\, \lambda H_1 : Px.\, H_0(HxH_1) \\ :\ (\Pi x : D.\, Px \to Qx) \to \Pi x : D.\, \neg Qx \to \neg Px\end{gathered}$$

5. (a) Give the full λP type derivation of the judgment:

$$D : *,\ P : (D \to *),\ a : D \vdash Pa : *$$

When parts of this type derivation are identical, you do not need to replicate them, it will be enough to indicate where the repetition occurs. For example this might be useful for derivations of the judgments:

$$\dfrac{}{D : * \vdash (D \to *) : \Box}$$
$$D : *,\ P : (D \to *) \vdash D : *$$

It also might be efficient to use abbreviations for one or more contexts.

For the typing rules of λP, see page 12 of this test.

As suggested in the exercise, we first make the following derivations:

$$\dfrac{\dfrac{}{\vdash * : \Box}}{D : * \vdash D : *} \qquad \dfrac{\dfrac{}{\vdash * : \Box} \quad \dfrac{}{\vdash * : \Box}}{D : * \vdash * : \Box} \quad \dfrac{\dfrac{}{\vdash * : \Box}}{D : * \vdash D : *}}{D : * \vdash (D \to *) : \Box}$$

$$\dfrac{\dfrac{}{\vdash * : \Box}}{D : * \vdash D : *} \qquad \dfrac{\vdots}{D : * \vdash (D \to *) : \Box}}{D : *,\ P : (D \to *) \vdash D : *}$$

Also, as suggested in the exercise, we define an abbreviation:

$$\Gamma_{DPa} := D : *,\ P : (D \to *),\ a : D$$

The derivation for the judgment in the exercise then is:

$$\dfrac{\dfrac{\dfrac{\vdots}{D : * \vdash (D \to *) : \Box}}{D : *,\ P : (D \to *) \vdash P : (D \to *)} \qquad \dfrac{\vdots}{D : *,\ P : (D \to *) \vdash D : *}}{\Gamma_{DPa} \vdash P : (D \to *)} \qquad \dfrac{\vdots}{D : *,\ P : (D \to *) \vdash D : *}}{\Gamma_{DPa} \vdash a : D}}{\Gamma_{DPa} \vdash Pa : *}$$

(b) The generic typing rule for (dependent) function types in Pure Type Systems is:

$$\dfrac{\Gamma \vdash A : s_1 \quad \Gamma,\, x : A \vdash B : s_2}{\Gamma \vdash (\Pi x : A.\, B) : s_3}$$

What is the triple (s_1, s_2, s_3) in the instance of this rule in the type derivation from the previous subexercise?

$$(*, \Box, \Box)$$

(c) And what are the triples (s_1, s_2, s_3) of the rules that are allowed in λP?

$$(*, *, *)$$
$$(*, \Box, \Box)$$

The following exercise is about *inductive types* and *recursive functions*.

6. (a) Give Coq definitions of an inductive type of *cons*-lists (the usual kind) of natural numbers, and of an inductive type of *snoc*-lists of natural numbers. *Cons*-lists are lists where elements are added at the start of the list, and *snoc*-lists are lists where they are added at the end.

```
Inductive conslist : Set :=
  | nil : conslist
  | cons : nat -> conslist -> conslist.

Inductive snoclist : Set :=
  | lin : snoclist
  | snoc : snoclist -> nat -> snoclist.
```

(b) Give the (dependent) induction principle of the type of *snoc*-lists. You may use both Coq notation or mathematical notation for this.

```
forall P : snoclist -> Prop,
P lin ->
(forall l : snoclist, P l -> forall n : nat, P (snoc l n)) ->
forall l : snoclist, P l
```

Or, in mathematical notation:

$\Pi P : \mathbb{L} \to *.\, P\, [] \to (\Pi l : \mathbb{L}.\, P\, l \to \Pi n : \mathbb{N}.\, P\, (l :: n)) \to \Pi l : \mathbb{L}.\, P\, l$

where we write $\mathbb{N}$ for `nat`, $\mathbb{L}$ for `snoclist`, $[]$ for `lin`, and $(l :: n)$ for `(snoc` l n`)`.

(c) Write one or more recursive Coq functions using `Fixpoint`, to converts a *snoc*-list to a *cons*-list. You may not assume the *append* function to have been defined already.

```
Fixpoint conslist_snoc (l : conslist) (n : nat) {struct l} :
    conslist :=
  match l with
  | nil => cons n nil
  | cons h t => cons h (conslist_snoc t n)
  end.

Fixpoint conslist_of_snoclist (l : snoclist) {struct l} :
    conslist :=
  match l with
  | lin => nil
  | snoc h t => conslist_snoc (conslist_of_snoclist h) t
  end.
```

Or, in a tail-recursive way:

```
Fixpoint conslist_of_snoclist_tailrec
    (l : snoclist) (l' : conslist) {struct l} : conslist :=
  match l with
  | lin => l'
  | snoc h t => conslist_of_snoclist_tailrec h (cons t l')
  end.

Definition conslist_of_snoclist_alt l :=
  conslist_of_snoclist_tailrec l nil.
```

(The invariant of `conslist_of_snoclist_tailrec` is that during the recursion the concatenation of `l` and `l'` will stay the same, as the `t` shifts from `l` to `l'`.)

The following exercise is about the *polymorphic lambda calculus* ($\lambda 2$, System F), and about the reading list of this year about *logical relations*.

7. We extend System F with Booleans, which gives the grammar:

$$A := a \mid A \to A \mid \forall a.\, A \mid \mathsf{bool}$$
$$M := x \mid MM \mid \lambda x{:}A.\, M \mid MA \mid \Lambda a.\, M \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{if}\ M\ \mathsf{then}\ M\ \mathsf{else}\ M$$

   (a) Give the typing rule(s) of this system for the conditional, $\mathsf{if}\ M_1\ \mathsf{then}\ M_2\ \mathsf{else}\ M_3$.

$$\frac{\Gamma \vdash M_1 : \mathsf{bool} \quad \Gamma \vdash M_2 : A \quad \Gamma \vdash M_2 : A}{\Gamma \vdash (\mathsf{if}\ M_1\ \mathsf{then}\ M_2\ \mathsf{else}\ M_3) : A}$$

(b) Give the reduction rule(s) of this system for the conditional.

$$\begin{array}{l} \mathsf{if\ true\ then}\ M\ \mathsf{else}\ N\ \ \rightarrow_\iota\ M \\ \mathsf{if\ false\ then}\ M\ \mathsf{else}\ N\ \rightarrow_\iota\ N \end{array}$$

(c) In the Pure Type System presentation of $\lambda 2$, the function types are written as $\Pi x{:}A.B$ and there is just one kind of function abstraction, so there is no separate abstraction with a capital lambda $\Lambda a.M$. Give the syntax used in the Pure Type System presentation for:

$$\begin{array}{c} \forall a.\, A \\ \Lambda a.\, M \end{array}$$

$$\begin{array}{c} \Pi a : *.\, A \\ \lambda a : *.\, M \end{array}$$

(d) Show how the Booleans could also have had an impredicative definition as lambda terms, without having to add syntax for this. Specifically give definitions in System F *without* the Booleans for

$$\begin{array}{c} \mathsf{bool} \\ \mathsf{true} \\ \mathsf{false} \\ \mathsf{if}\ M_1\ \mathsf{then}\ M_2\ \mathsf{else}\ M_3 \end{array}$$

in a way that these behave as Booleans.

$$\begin{array}{rcl} \mathsf{bool} & := & \forall a.\, a \rightarrow a \rightarrow a \\ \mathsf{true} & := & \Lambda a.\, \lambda x : a.\, \lambda y : a.\, x \\ \mathsf{false} & := & \Lambda a.\, \lambda x : a.\, \lambda y : a.\, y \\ \mathsf{if}\ M_1\ \mathsf{then}\ M_2\ \mathsf{else}\ M_3 & := & M_1 A M_2 M_3 \end{array}$$

In this last line A is the type of M_2 and M_3.

(e) Show that the system presented here has a closed term N of type

$$\mathsf{bool} \to \mathsf{bool}$$

that is not the identity, in the sense that there exists a closed normal form M of type bool, for which the normal form of NM differs from M.

$$N := \lambda x : \mathsf{bool}.\, \mathsf{true}$$

With $M = \mathsf{false}$ we get that $N\,\mathsf{false}$ has normal form true, which differs from false.

We want to prove the 'free theorem' for our system that for the terms F of type

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

it *does* hold that it has to behave like the (polymorphic) identity, in the sense that if A is a closed type and M is a closed normal form of type A, then the normal form of $F\,A\,M$ is equal to M.

This means that the term $(F\,\mathsf{bool})$ of type $\mathsf{bool} \to \mathsf{bool}$ *has* to be the identity, and therefore F will not be able to make use of the fact when its argument is the Booleans. The function F will be *parametric*.

To prove this we define a logical relation.

We first define $\mathsf{Rel}(B, B')$ for each pair of types B and B' in our system, as the set of *all* relations between the sets of closed normal forms of types B and B' respectively.

We also define the normalization function $\mathsf{nf}(M)$ that maps a term to its β-normal form. This is a well-defined total function as the system has the Church-Rosser property and is Strongly Normalizing.

We now will define the *logical relation* $[\![A]\!]_\rho$. The subscript ρ is a function that maps the free type variables in A to triples of the form (B, B', R), where $R \in \mathsf{Rel}(B, B')$. We will have that $[\![A]\!]_\rho \in \mathsf{Rel}(A_1, A_2)$, where A_1 and A_2 are obtained from A by replacing the free type variables in A by the first and second components of their image under ρ.

The logical relation $[\![A]\!]_\rho$ is defined recursively by:

$$\begin{aligned}
[\![a]\!]_\rho &:= R \quad \text{when } \rho(a) = (B, B', R) \\
[\![A \to B]\!]_\rho &:= \ldots \\
[\![\forall a.\, A]\!]_\rho &:= \{(M, M') \mid \forall B, B', R \in \mathsf{Rel}(B, B'). \\
&\qquad\qquad (\mathsf{nf}(MB), \mathsf{nf}(M'B')) \in [\![A]\!]_{\rho[a \mapsto (B,B',R)]}\} \\
[\![\mathsf{bool}]\!]_\rho &:= \{(\mathsf{true}, \mathsf{true}), (\mathsf{false}, \mathsf{false})\}
\end{aligned}$$

(f) In the above definition we left out the case for $[\![A \to B]\!]_\rho$. Give this missing part of the definition.

$$\begin{aligned}
[\![A \to B]\!]_\rho := \{(M, M') \mid \forall N, N'. \\
\text{if } (N, N') \in [\![A]\!]_\rho \text{ then } (\mathsf{nf}(MN), \mathsf{nf}(M'N')) \in [\![B]\!]_\rho\}
\end{aligned}$$

(g) The notation $[\![-]\!]$ suggest that this logical relation is a *semantics* for this type system. This is called the *parametricity theorem* or the *fundamental property*. Give the instance of this theorem for closed terms.

If $\vdash M : A$, then $(\mathsf{nf}(M), \mathsf{nf}(M)) \in [\![A]\!]_\emptyset$.

(h) This parametricity theorem (the full one, including open terms) is proved by induction. What induction is used for this?

Induction on the type derivation of $\Gamma \vdash M : A$.

Now we prove that every closed term F of type $\forall a.\, a \to a$ has the property that for a closed type A and a closed normal form M of type A we have $\mathsf{nf}(F\,A\,M) = M$.

The parametricity theorem tells us that

$$(\mathsf{nf}(F), \mathsf{nf}(F)) \in [\![\forall a.\, a \to a]\!]_\emptyset$$

This means that for all A_1 and A_2 and $R \in \mathsf{Rel}(A_1, A_2)$ we have

$$(\mathsf{nf}(FA_1), \mathsf{nf}(FA_2)) \in [\![a \to a]\!]_{[a \mapsto (A_1, A_2, R)]}$$

And that means that for all M_1 and M_2 with

$$(M_1, M_2) \in [\![a]\!]_{[a \mapsto (A_1, A_2, R)]} = R$$

we have:

$$(\mathsf{nf}(F A_1\, M_1), \mathsf{nf}(F A_2\, M_2)) \in [\![a]\!]_{[a \mapsto (A_1, A_2, R)]} = R$$

Now from this we then want to deduce that

$$\mathsf{nf}(F A\, M) = M$$

(i) What are A_1, A_2, M_1, M_2 and R that will make the last step in this proof work?
Take $A_1 = A_2 = A$, $M_1 = M_2 = M$, and $R = \{(M, M)\}$.