Type Theory and Coq 2014

26-01-2015

1. Consider the term of the untyped lambda calculus:

$$(\lambda x.x)(\lambda y.\,(\lambda z.z)\,y\,(\lambda w.w))$$

   (a) Give the normal form of this term.

$$\lambda y.y\,(\lambda w.w)$$

   (b) Give a most general type of this term, where the term is taken to be a term of Curry-style simple type theory. (You do not need to explain how you obtained this type, nor why it is a most general type.)

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

   (c) Give the term of Church-style simple type theory that corresponds to the untyped lambda term and that has the type from part (b).

$$(\lambda x{:}((a{\to}a){\to}b){\to}b.\,x)(\lambda y{:}(a{\to}a){\to}b.\,(\lambda z{:}(a{\to}a){\to}b.\,z)\,y\,(\lambda w{:}a.\,w))$$

   (d) Give a type of this term in Curry-style simple type theory that is *not* a most general type.

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

2. Consider the formula of first order propositional logic:

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

(a) Give a proof in first order propositional logic of this formula. (Write all the names of the proof rules in the proof tree.)

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

(b) Give the proof term of Church-style simple type theory of this proof.

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

(c) Give the type judgment for the term from part (b).

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

(d) Give a derivation of the type judgment from part (c). (You do not need to give names for the typing rules in the derivation tree, and you may use abbreviations for contexts.)

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

3. Consider the formula of first order predicate logic:

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

(a) Give a proof in first order predicate logic of this formula. (Write all the names of the proof rules in the proof tree.)

We prove the formula

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

which is an alpha-renamed version of the formula from the exercise:

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

(b) Which of the rules in this proof has a variable condition, what is this condition, and why is it satisfied?

The $I\forall$ rule has the variable condition that the variable that is generalized should not be free in any uncanceled assumption. The only assumption in this proof is $\forall x'.\forall y'.R(x',y')$ which has no free variables, so this condition is trivially satisfied.

(c) Give the type of λP that corresponds to the formula.

$$(\Pi x{:}D.\,\Pi y{:}D.\,R\,x\,y) \to (\Pi x{:}D.\,\Pi y{:}D.\,R\,y\,x)$$

(d) Give a λP proof term for the type from part (c).

$$\lambda H : (\Pi x{:}D.\,\Pi y{:}D.\,R\,x\,y).\,\lambda x{:}D.\,\lambda y{:}D.\,H\,y\,x$$

4. Consider the term of λC:

$$\mathsf{or}_2 \;:\equiv\; \lambda A : *.\,\lambda B : *.\,\Pi C : *.\,(A \to C) \to (B \to C) \to C$$

(a) Give the type of or_2 in λC. (See page 9 for the typing rules of λC, in case you need those.)

$$* \to * \to *$$

(b) Is or_2 also typeable in $\lambda 2$? Explain your answer.

No, this term is only typable in systems that contain $\lambda\omega$.

The type $* \to *$ is not allowed in $\lambda 2$, because it needs the rule $(\Box, \Box)$, and $\lambda 2$ only has the rules $(*, *)$ and $(\Box, *)$.

(c) Give a term of λC that inhabits the following λC type:

$$\Pi A : *.\, \Pi B : *.\, A \to \mathsf{or}_2\, A\, B$$

$$\lambda A{:}*.\, \lambda B{:}*.\, \lambda H{:}A.\, \lambda C{:}*.\, \lambda H_1{:}A{\to}C.\, \lambda H_2{:}B{\to}C.\, H_1\, A$$

(d) Give a term of λC that inhabits the following λC type:

$$\Pi A : *.\, \Pi B : *.\, \Pi C : *.\, \mathsf{or}_2\, A\, B \to (A \to C) \to (B \to C) \to C$$

$$\lambda A{:}*.\, \lambda B{:}*.\, \lambda C{:}*.\, \lambda H_0{:}(\mathsf{or}_2\, A\, B).\, \lambda H_1{:}A{\to}C.\, \lambda H_2{:}A{\to}C.\, H_0\, C\, H_1\, H_2$$

5. Consider the λC type $a \to a$ in the context $a : *$.

(a) Give the λC typing judgment (without a derivation) that gives the *kind* of this type.

$$a : * \vdash (a \to a) : *$$

(b) Give a derivation in λC of the judgment from part (a). (See page 9 for the typing rules of λC. You do not need to give names for the typing rules in the derivation tree.)

$$\dfrac{\dfrac{\dfrac{}{\vdash * : \Box}}{a : * \vdash a : *} \qquad \dfrac{\dfrac{\dfrac{}{\vdash * : \Box}}{a : * \vdash a : *} \quad \dfrac{\dfrac{}{\vdash * : \Box}}{a : * \vdash a : *}}{a : *,\, x : a \vdash a : *}}{a : * \vdash (\Pi x : a.\, a) : *}$$

(c) Give also an *inhabitant* in λC of this type.

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

(d) Give the λC typing judgment (without a derivation) for this inhabitant.

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

6. Consider the Coq inductive type for binary positive numbers:

```
Inductive positive : Set :=
| xH : positive
| xO : positive -> positive
| xI : positive -> positive.
```

In this representation of binary numbers, `xH` stands for the number 1, `xO` stands for the function $\lambda n.2n$ (which adds a zero at the end of the number), and `xI` stands for the functin $\lambda n.2n + 1$ (which adds a one).

(a) Give a Coq term that represents the binary form of the decimal number 18.

18 has binary representation 10010 and therefore the term is

```
xO (xI (xO (xO xH)))
```

(b) Give the type of the dependent induction principle `positive_rect` for this inductive type. (You can write this induction principle using Coq syntax or using PTS syntax, whatever is your preference.)

```
forall P : positive -> Type,
  P xH ->
  (forall n : positive, P n -> P (xO n)) ->
  (forall n : positive, P n -> P (xI n)) ->
  forall n : positive, P n
```

(c) Give the type of the corresponding non-dependent induction principle `positive_rect_nondep` for this inductive type.

```
forall A : Type,
  A ->
  (positive -> A -> A) ->
  (positive -> A -> A) ->
  positive -> A
```

(d) Write a function `succ : positive -> positive` that adds one to its argument using a combination of `Fixpoint` and `match`.
For instance

`succ (xI xH) = xO (xO xH)`

should hold, because succ(3) = 4.

```
Fixpoint succ (n : positive) {struct n} : positive :=
  match n with
  | xH => xO xH
  | xO n' => xI n'
  | xI n' => xO (succ n')
  end.
```

(e) Now also write the `succ` function using the non-dependent induction priciple used as a primitive recursor.

```
positive_rect_nondep positive
  (xO xH)
  (fun n' : positive => fun _ : positive => xI n')
  (fun _ : positive => fun succ' : positive => xO succ')
```

7. We define the extended (untyped) lambda terms and the subclass called *values* by:

$$t ::= x \mid t_1 t_2 \mid v$$
$$v ::= \lambda x.t \mid [\tilde{x}\, v_1 \ldots v_n]$$

On these terms we define weak reduction by the three rules:

$$(\lambda x.t)v \to_v t[x := v] \qquad (\beta_v)$$
$$[\tilde{x}\, v_1 \ldots v_n]v \to_v [\tilde{x}\, v_1 \ldots v_n\, v] \qquad (\beta_s)$$
$$E_v(t) \to_v E_v(t') \quad \text{if } t \to_v t' \qquad (\text{context}_v)$$

where the one step contexts are defined as:

$$E_v(\Box) ::= \Box\, v \mid t\, \Box$$

We write $\mathcal{V}(t)$ for the unique normal form of t under $\to_v$ reduction (if it exists), the function $\mathcal{V}$ is a *partial* function. Using this we define functions $\mathcal{R}(t)$ and $\mathcal{N}(v)$ recursively by:

$$\mathcal{N}(t) = \mathcal{R}(\mathcal{V}(t)) \tag{1}$$

$$\mathcal{R}(\lambda x.t) = \lambda y.\, \mathcal{N}((\lambda x.t)\, [\tilde{y}]) \quad (y \text{ fresh}) \tag{2}$$

$$\mathcal{R}([\tilde{x}\, v_1 \dots v_n]) = x\, \mathcal{R}(v_1) \dots \mathcal{R}(v_n) \tag{3}$$

Finally we define the term t_7 by

$$t_7 := (\lambda x.x)(\lambda y.\, (\lambda z.z)\, y\, (\lambda w.w))$$

(a) Does there exist an extended lambda term that has two different one step $\to_v$ reductions? If so, give an example.

No, weak reduction $\to_v$ is deterministic.

First note that a value v will never reduce, because all the left hand sides of the reduction rules are an application, and values are never an application.

Now there are four rules:

$$\begin{aligned} (\lambda x.t)v &\to_v \dots \\ [\tilde{x}\, v_1 \dots v_n]v &\to_v \dots \\ uv &\to_v \dots \\ tu &\to_v \dots \quad \dots \end{aligned}$$

where u is a term that reduces. Clearly there is no overlap between these left hand sides. This means that by following the rules we will find at most one redex, which can be reduced in only one way.

(b) Is there an extended lambda term t for which $\mathcal{V}(t)$ does not exist (because the reduction does not terminate)? If so, give an example.

Yes, take $t = \Omega = \omega\omega = (\lambda x.xx)(\lambda x.xx)$. We have $\Omega \to_v \Omega$ by rule β_v because ω is a value, so clearly Ω does not have a normal form under $\to_v$.

(c) Is there an extended lambda term t that is typable in Curry-style simple type theory (and therefore does not contain subterms of the shape $[\tilde{x}\, v_1 \ldots v_n]$) for which $\mathcal{V}(t)$ does not exist? If so, give an example.

No, such at term does not exist. Weak reduction is a restriction of normal beta reduction, and simple type theory is strongly normalizing under beta reduction.

(d) Show how the value $\mathcal{V}(t_7)$ is calculated, and give the result. Write down all the $\to_v$ reduction steps that are used in this calculation.

We have the reduction

$$(\lambda x.x)(\lambda y.\,(\lambda z.z)\,y\,(\lambda w.w)) \to_v \lambda y.\,(\lambda z.z)\,y\,(\lambda w.w) \qquad (\beta_v)$$

using rule β_v, and the latter term is a value and does not reduce. Therefore we have:

$$\mathcal{V}(t_7) = \lambda y.\,(\lambda z.z)\,y\,(\lambda w.w)$$

(e) Show how the value $\mathcal{N}(t_7)$ is calculated, and give the result. Whenever in this calculation you calculate a weak normal form $\mathcal{V}(t)$, write down all the $\to_v$ reduction steps like in part (d).

We have

$$\begin{aligned}
\mathcal{V}((\lambda x.x)(\lambda y.(\lambda z.z)\,y\,(\lambda w.w))) &= \lambda y.(\lambda z.z)\,y\,(\lambda w.w) \\
\mathcal{V}((\lambda y.(\lambda z.z)\,y\,(\lambda w.w))[\tilde{y}]) &= [\tilde{y}\,(\lambda w.w)] \\
\mathcal{V}((\lambda w.w)[\tilde{w}]) &= [\tilde{w}]
\end{aligned}$$

because of the following reduction paths:

$$(\lambda x.x)(\lambda y.\,(\lambda z.z)\,y\,(\lambda w.w)) \to_v \lambda y.(\lambda z.z)\,y\,(\lambda w.w) \qquad (\beta_v)$$

$$(\lambda z.z)[\tilde{y}] \to_v [\tilde{y}] \qquad (\beta_v)$$

$$\begin{aligned}
(\lambda y.(\lambda z.z)\,y\,(\lambda w.w))[\tilde{y}] &\to_v (\lambda z.z)[\tilde{y}](\lambda w.w) && (\beta_v) \\
&\to_v [\tilde{y}](\lambda w.w) && (\text{context}_v) \\
&\to_v [\tilde{y}\,(\lambda w.w)] && (\beta_s)
\end{aligned}$$

$$(\lambda w.w)[\tilde{w}] \to_v [\tilde{w}] \qquad (\beta_v)$$

Using these equalities we then calculate:

$$\begin{aligned}
\mathcal{N}(t_7) &= \mathcal{R}(\mathcal{V}(t_7)) \\
&= \mathcal{R}(\lambda y.(\lambda z.z)\, y\, (\lambda w.w)) \\
&= \lambda y.\mathcal{N}((\lambda y.(\lambda z.z)\, y\, (\lambda w.w))[\tilde{y}]) \\
&= \lambda y.\mathcal{R}(\mathcal{V}((\lambda y.(\lambda z.z)\, y\, (\lambda w.w))[\tilde{y}])) \\
&= \lambda y.\mathcal{R}([\tilde{y}\, (\lambda w.w)]) \\
&= \lambda y.y\, \mathcal{R}(\lambda w.w) \\
&= \lambda y.y\, (\lambda w.\mathcal{N}((\lambda w.w)[\tilde{w}])) \\
&= \lambda y.y\, (\lambda w.\mathcal{R}(\mathcal{V}((\lambda w.w)[\tilde{w}]))) \\
&= \lambda y.y\, (\lambda w.\mathcal{R}([\tilde{w}])) \\
&= \lambda y.y\, (\lambda w.w)
\end{aligned}$$