2.7.1 — Foundations of Proof Systems

Exam

Nov. 30th 2021

1 System F

Consider the following types in system F :

$$\begin{aligned} \text{nat} &\equiv \forall\alpha\,.\,(\alpha \to \alpha) \to \alpha \to \alpha \\ l &\equiv \forall\beta\,.\,\beta \to (\text{nat} \to \beta \to \beta) \to \beta. \end{aligned}$$

Question 1 Give two closed terms of type l. ⋄

Solution.

$$\Lambda\beta.\lambda x : \beta.\lambda f : \text{nat} \to \beta \to \beta.x$$

$$\Lambda\beta.\lambda x : \beta.\lambda f : \text{nat} \to \beta \to \beta.(f\ 0\ x)$$

Question 2 Explain how the closed elements of this type can be viewed as lists of natural numbers. ⋄

Solution. The closed terms are of the form

$$\Lambda\beta.\lambda x : \beta.\lambda f : \text{nat} \to \beta \to \beta.(f\ n_1\ (f\ n_2\ \dots (f\ n_k\ x)\dots))$$

where the n_i are closed terms of type nat, corresponding to the list $[n_1; n_2; \dots; n_k]$. □

Question 3 Construct a function from l to nat which sums all the elements of the list (you can consider the addition function has already been defined). ⋄

Solution. $\lambda m : l.(m\ \ \text{nat}\ \ 0\ \lambda n : \text{nat}\,.\lambda r : \text{nat}\,.n + r)$ □

Question 4 Construct a function returning a when the sum of the elements of a list is 0 and b if not (where a and b are some terms of a given type). ⋄

Solution. $\lambda m : l.(sum\ l\ A\ a\ \lambda x.\lambda y.b)$ □

2 HOL

You are given, in HOL, a type T and a relation over this type :

$$R : T \to T \to o$$

Question 5 Define :

1. The reflexive-transitive closure of R,
2. the transitive closure of R,
3. the proposition "there exists a cycle in R". $\diamond$

$R_p \equiv \lambda a : T \lambda b : T . \forall Q : T \to T \to o . (\forall x : T . \forall y : T . (R\,x\,y) \to (Q\,x\,y)) \to (\forall x : T . \forall y : T . \forall z : T . (Q\,x\,y) \to (R\,y\,z) \to$

$\lambda a : T \lambda b : T . \forall Q : T \to T \to o . (\forall x : T . (Q\,x\,x)) \to (\forall x : T . \forall y : T . \forall z : T . (Q\,x\,y) \to (R\,y\,z) \to (R\,x\,z)) \to (Q\,a\,b)$

$$\forall X : o . (\forall x : T . (R_P\; x\; x) \to X) \to X$$

About partial functions in Type Theory

This part is in Martin-Löf's Type Theory. In what follows I use the notation $A \vee B$ for the sum type $A + B$ in order to avoid confusion with the addition function $a + b$ over natural numbers. I also write $A \wedge B$ for $\Sigma x : A\,.\,B$ when x does not occur in B.

A partial function from A to B is a function which is possibly not defined for some elements of the domain A. Handling such partial functions in type theories is notoriously not trivial, because elements of the type $A \to B$ are total functions.

You are part of a research group which reflects on how to handle functions from a type A to a type B, which are only defined for objects of type A which verify a property P. That is you are given :

$$\begin{array}{rcl} A, B & : & \mathsf{Type} \\ P & : & A \to \mathsf{Type} \end{array}$$

A first colleague, Andrew, sugests two possible types for these partial functions :

$$\begin{array}{rcl} part1 & \equiv & \Pi x : A\,.\,P\,x \to B \\ part2 & \equiv & (\Sigma x : A\,.\,P\,x) \to B \end{array}$$

Question 6 Show that these two types are isomorphic. That is give two functions

$$\begin{array}{rcl} f12 & : & part1 \to part2 \\ f21 & : & part2 \to part1 \end{array}$$

such that you can prove :

$$\begin{array}{rcll} f12c & : & \Pi f : part1\,.\,\Pi x : A\,.\,\Pi p : P\,x\,.\,f\;x\;p = (f21\;(f12\;f)\;x\;p) & (1) \\ f21c & : & \Pi g : part2\,.\,\Pi y : \Sigma x : A\,.\,P\,x\,.\,g\;y = (f12\;(f21\;f)\;y) & (2) \end{array}$$

(you do not have to give the term $f21c$) $\diamond$

Solution.

$$f12 \equiv \lambda f : part1 \,.\, \lambda y : \Sigma x : A \,.\, P\, x \,.\, f\ \pi_1(y)\ \pi_2(y)$$

$$f21 \equiv \lambda g : part2 \,.\, \lambda x : A \,.\, \lambda p : P\, x \,.\, f21\ (x, p)$$

$$(f21\ (f12\ f)\ x\ p) = (f12\ f\ (x, p)) = (f\ pi_1(x, p)\ \pi_2(x, p)) = (f\ x\ p)$$

$$(f12\ (f21\ g)\ y) = (f21\ g\ \pi_1(y)\ \pi_2(y)) = (g\ (\pi_1(y), \pi_2(y)))$$

(the last one is well-typed only if one has a surjective pairing rule, which explains why I did not ask for it) □

A colleague, Beate, sees a problem with Andrew's proposal. She says that the result of a function of type part1 can depend upon the proof p that the argument verifies P. Thus, objects of type part1 (or part2) do not correspond to what a usual mathematician would understand as a partial function.

Question 7 To help Beate make her point, describe a function f of the following type, where q is a given natural number :

$$\Pi n : \text{nat} \,.\, (\Sigma p_1 : nat \,.\, \Sigma p_2 : nat \,.\, q = p_1 * p_2 + n) \rightarrow \text{nat}$$

such that $(f\ 4\ h)$ and $(f\ 4\ h')$ can return different results (for different terms h and h', obviously). ◇

A discussion follows. It is agreed that a possible fix is to ask every partial function to come with a proof that the result of the function does not depend of the proof of $(P\ x)$, but this makes the proof developments quite tedious.

Andrew thus proposes to add a new construct to the type theory. Given A and P, one adds a new type $\{x : A \mid P\}$, called *subset type*, which is very similar to $\Sigma x : A.P$ but where one "forgets" the proof of $(P\ x)$. Here is a first proposal for the new rules :

$$\frac{\Gamma \vdash A : \mathsf{Type} \quad \Gamma(x : A) \vdash P : \mathsf{Type}}{\Gamma \vdash \{x : A \mid P\} : \mathsf{Type}} \qquad \frac{\Gamma \vdash \{x : A \mid P\} : \mathsf{Type} \qquad \Gamma \vdash t : A \qquad \Gamma \vdash p : P[x \setminus t]}{\Gamma \vdash \mathbf{c}_{x.P}(t) : \{x : A \mid P\}}$$

$$\frac{\Gamma \vdash t : \{x : A \mid P\}}{\Gamma \vdash \pi_1(t) : A} \qquad \frac{\Gamma \vdash t : \{x : A \mid P\}}{\Gamma \vdash \mathbf{oracle}_{x.P}(t) : P[\pi_1(t)]}$$

One also adds a reduction rule $\pi_1(\mathbf{c}_{x.P}(t)) \rhd t$.

We admit this system enjoys subject reduction.

Question 8 Explain why type checking in this system is not decidable. ◇

Solution. To type-check $\mathbf{c}_{x.P}(t)$ one needs to find a proof of $P[x \setminus t]$. So type-checking implies deciding whether a possibly arbitrary proposition is true. □

Question 9 Show that, in this system, you can prove, for all A and P :

$$\Pi x : A \,.\, \Pi y : A \,.\, \mathbf{c}_P(x) = \mathbf{c}_P(y) \rightarrow x = y.$$

◇

Solution. $\mathbf{c}_P(x) = \mathbf{c}_P(y)$ implies $\pi_1(\mathbf{c}_P(x)) = \pi_1(\mathbf{c}_P(y))$ which reduces to $x = y$. □

Question 10 In this type theory, construct a closed term of type $A \vee B$ which does not reduce to either $i(u)$ or $j(v)$. ◇

Solution. Take a closed proof $t : 0 = 0$. Then $i(t)$ (or $j(t)$) is of type $0 = 0 \vee 0 = 0$.

Thus, $\mathbf{c}_{\lambda x:nat.0=0}(0) : \{x : nat \mid 0 = 0\}$ and $\mathbf{oracle}(\mathbf{c}_{\lambda x:nat.0=0}(0)) : 0 = 0 \vee 0 = 0$ but it yields no constructive content.

Question 11 Andrew says that his extension allows the encoding of unordered pairs in the style of set theory. Given a and b two objects of type A, one defines :

$$\{a; b\} \equiv \{x : A \mid x = a \vee x = b\}$$

.

Show that one can prove : $\Pi x : \{a; b\} \,.\, \pi_1(x) = a \vee \pi_1(x) = b$. ◇

Solution. The proof is simply $\lambda x : \{a; b\} \,.\, oracle(x)$. □

In the following questions, we assume the type theory is equipped with a primitive type of booleans, with two elements **true** and **false**, and an operator similar to the R of natural numbers. Thus you can decide equality over booleans, prove there are no other booleans than **true** and **false**, *etc...*

Another colleague, Charly, now wonders how far constructivity is broken by these subset types. He comes up with the following relation between $\{a; b\}$ and booleans :

$$R \equiv \lambda x : \{a; b\} \,.\, \lambda y : bool.(y = \mathbf{true} \rightarrow \pi_1(x) = a) \wedge (y = \mathbf{false} \rightarrow \pi_1(x) = b)$$

Question 12 Show that $\Pi x : \{a; b\} \,.\, \Sigma y : bool \,.\, R\ x\ y$. ◇

Solution. Given $x : \{a; b\}$, we use the previous question to obtain $\pi_1(x) = a \vee \pi_1(x) = b$. We reason by case over this disjunction.

— If $\pi_1(x) = a$ we chose $y = \mathbf{true}$ and we have $\pi_1(x) = a$ thus the left branch of the conjunction is true. The second branch is true because $\mathbf{true} \neq \mathbf{false}$.

— If $\pi_1(x) = b$ we chose $y = \mathbf{false}$ and the reasoning is similar. □

Question 13 From there, construct a function

$$f : \{a; b\} \rightarrow \text{bool}$$

such that $\Pi x : \{a; b\} \,.\, R\ x\ (f x)$. ◇

Solution. Let us call p the proof constructed in the previous question. We just take $f \equiv \lambda x.\pi_1(p\ x)$. □

Question 14 Use this to prove $a = b \vee \neg a = b$.

Hint : remember $(f\ x)$ is a boolean, so you can reason by case over its value. ◇

Solution. We know that equality is decidable over the booleans. We look at the possible values of $(f\ \mathbf{c}(a))$ and $(f\ \mathbf{c}(b))$.

If $(f\ \mathbf{c}(a)) \neq (f\ \mathbf{c}(b))$ one can show that $a \neq b$, since $a = b$ implies $(f\ \mathbf{c}(a)) = (f\ \mathbf{c}(b))$.

Take $(f\ \mathbf{c}(a)) = (f\ \mathbf{c}(b))$. For instance $(f\ \mathbf{c}(b)) = \mathbf{true}$; this implies $\pi_1(\mathbf{c}(b)) = a$, that is $a = b$. The case $(f\ \mathbf{c}(a)) = \mathbf{false}$ is similar.

The group thus considers that this version of "naive" subset types is a little too radical. Two other colleagues, Desmond and Molly, come up with another approach, without changing the type theory (thus dropping the addition of subset types).

To make things simpler, you can, from now on, use the following extentionality axiom, for any types A and B :

$$ext : \Pi f : A \to B\ .\ \Pi g : A \to B\ .\ (\Pi x : A\ .\ (f\ x) = (g\ x)) \to f = g.$$

Question 15 Prove that, for any type A, all proofs of $\neg A$ are equal :

$$\Pi x : A \to \bot\ .\ \Pi y : A \to \bot\ .\ x = y.$$

◇

Solution. Using the extentionality axiom, we just have to prove that x and y are pointwise equal. It is trivial to prove that all proofs of $\bot$ are equal. The result follows. □

Desmond and Molly's idea is thus to use $\neg\neg A$ as a weakened version of A, since there is at most one proof of $\neg\neg A$. A partial function from A to B is thus a function of type :

$$(\Sigma x : A\ .\ \neg\neg(P\ x)) \to B$$

.

Question 16 Show you have understood their point. How would you type a function defined only over even natural numbers ?

How would such a function *div2* returning the number divided by two look like ? In particular what can you say about the number of reductions to normalize (*div2* n) ? What is the difference with a function of type :

$$\Pi n : \text{nat}\ .\ (\Sigma p : \text{nat}\ .\ n = p + p) \to \text{nat}\ \ ?$$

◇

Solution. The function could only rely on the value of the argument n to compute the result. The division by 2 would thus take a time proportional to n. This is different to a function of the type given above, which could just return p in constant time. □

Molly says there may be an additional advantage to this approach. She claims that for any types A and B, one can construct terms of the following types :

$$\begin{aligned} p_1 &: \neg\neg(A \vee \neg A) \\ p_2 &: ((A \vee \neg A) \to \neg\neg B) \to \neg\neg B \end{aligned}$$

Question 17 Show Molly is right by constructing the terms p_1 and p_2. ◇

Solution.

$$p_1 \equiv \lambda naa : \neg(A \vee \neg A).naa\ (j(\lambda a : A.(naa\ i(a))))$$

$$p_2 \equiv \lambda f : ((A \vee \neg A) \to \neg\neg B).(p_1\ \lambda ana : (A \vee \neg A).\lambda nb : \neg B.(f\ ana\ nb)$$

Question 18 What does this mean when working with objects of types of the form $\Sigma x : A\ .\ \neg\neg B$? ◇

Solution. This means one can use classical logic prof proving the condition $\neg\neg B$. □