Homework

1. Consider

$$\mathbb{N} = \{0^{\mathbb{N}}, S^{\mathbb{N}\to\mathbb{N}}\}$$

$$\mathbb{B} = \{T^{\mathbb{B}}, F^{\mathbb{B}}\}$$

$R^{\tau}_{\mathbb{N}}$ as in the slides

define a function $eq^{\mathbb{N}\to\mathbb{N}\to\mathbb{B}}$ which computes equality of such two natural numbers as a boolean using R in Gödel's system T^{-}.

A solution (as presented by Bojan iirc):

The idea is to define the function using a non-base-type recursor:

$$eq^{\mathbb{N}\to\mathbb{N}\to\mathbb{B}}\, n\, m = (R^{\mathbb{N}\to\mathbb{B}}_{\mathbb{N}}\, n\, M\, N)\, m$$

with the first part defining an 'equality to n' test, M of type $\mathbb{N}\to\mathbb{B}$ and N of type $\mathbb{N}\to(\mathbb{N}\to\mathbb{B})\to\mathbb{N}\to\mathbb{B}$. In the base case, the equality test is just a test for zero. That is, $M\,0 = T^{\mathbb{B}}$ and $M\,(S\,n) = F^{\mathbb{B}}$, which can be obtained by using a recursor of base type:

$$M\, n = R^{\mathbb{B}}_{\mathbb{N}}\, n\, T^{\mathbb{B}}\, (\lambda\, mb.F^{\mathbb{B}})$$

In the step case, the equality test should fail for the argument being zero and recurse otherwise. The first argument (the 'input') to N is not needed, only the second argument (the recursive call, the 'output'). That is, we set $N\,n = N'$ with $N'\,f\,0 = F^{\mathbb{B}}$ (fail for zero) and $N'\,f\,(S\,m) = f\,m$ (applying an 'equality to $(S\,n)$'-test to $(S\,m)$ is the same as applying an 'equality to n-test, the recursive call f, to m. All this can again be obtained by using a recursor of base type:

$$N\, n\, f\, m = R^{\mathbb{B}}_{\mathbb{N}}\, m\, F^{\mathbb{B}}\, (\lambda\, mb.f\, m)$$

Combining/expanding the above, we obtain

$$eq^{\mathbb{N}\to\mathbb{N}\to\mathbb{B}} = \lambda\, nm.(R^{\mathbb{N}\to\mathbb{B}}_{\mathbb{N}}\, n\, (\lambda\, n.R^{\mathbb{B}}_{\mathbb{N}}\, n\, T^{\mathbb{B}}\, (\lambda\, mb.F^{\mathbb{B}}))\, (\lambda\, n\, f\, m.R^{\mathbb{B}}_{\mathbb{N}}\, m\, F^{\mathbb{B}}\, (\lambda\, mb.f\, m)))\, m$$

This looks rather imposing, but really one shouldn't expand; the same would happen if we were to unfold definitions in any functional program.[1] *Anyway, let's do some computations to verify the above (using the abbreviations introduced).*

$$\begin{aligned}
\underline{eq^{\mathbb{N}\to\mathbb{N}\to\mathbb{B}}\, (S\,0)\, (S\,0)} &= \underline{(R^{\mathbb{N}\to\mathbb{B}}_{\mathbb{N}}\, (S\,0)\, M\, N)}\, (S\,0)\\
&= N\,0\,\underline{(R^{\mathbb{N}\to\mathbb{B}}_{\mathbb{N}}\, 0\, M\, N)}\, (S\,0)\\
&= \underline{N\,0\,M\, (S\,0)}\\
&= \underline{R^{\mathbb{B}}_{\mathbb{N}}\, (S\,0)\, F^{\mathbb{B}}\, (\lambda\, mb.M\, m)}\\
&= \underline{(\lambda\, mb.M\, m)\, 0\, (R^{\mathbb{B}}_{\mathbb{N}}\, 0\, F^{\mathbb{B}}\, (\lambda\, mb.M\, m))}\\
&= \underline{M\, 0}\\
&= \underline{R^{\mathbb{B}}_{\mathbb{N}}\, 0\, T^{\mathbb{B}}\, (\lambda\, mb.F^{\mathbb{B}})}\\
&= T^{\mathbb{B}}
\end{aligned}$$

[1] Alternatively, one can observe that two numbers m,n are equal if and only if both $m - n = 0$ and $n - m = 0$ with $-$ so-called cut-off subtraction (or monus). Since both the cut-off subtraction (e.g. implementing $m - n$ by n times taking the predecessor of m) and a test-for-zero are easily implemented (see above), the result follows.

It is easy to see that replacing the second argument by 0 *in the above, the first four lines above still work, mutatis mutandis, with the computation then ending in the fifth line with the base case of the recursor,* $F^{\mathbb{B}}$*, as result.*

In the other direction, replacing the second argument by $(S\,(S\,0))$ *in the above, the first seven lines above still work, mutatis mutandis, from which moment on, the computation continues as:*

$$\begin{aligned}\underline{R^{\mathbb{B}}_{\mathbb{N}}\,(S\,0)\,T^{\mathbb{B}}\,(\lambda\,mb.F^{\mathbb{B}})} &= \underline{(\lambda\,mb.F^{\mathbb{B}})\,0\,(R^{\mathbb{B}}_{\mathbb{N}}\,0\,T^{\mathbb{B}}\,(\lambda\,mb.F^{\mathbb{B}}))}\\ &= F^{\mathbb{B}}\end{aligned}$$

Bonus Exercise for next time: Can you prove that your (or the above) solution is correct?

2. Given the totality predicate on $\mathbb{N}$

   $$T(0), \forall^{nc}_n(Tn \to T(Sn))$$

   And the weak elimination axiom:

   $$\forall^{nc}_n(Tn \to P0 \to \forall^{nc}_n(Pn \to P(Sn)) \to Pn)$$

   Find a suitable instantiation for P to prove the regular (stronger) elimination axiom:

   $$\forall^{nc}_n(Tn \to Q0 \to \forall^{nc}_n(Tn \to Qn \to Q(Sn)) \to Qn)$$

   An answer:

   The goal here was only to reason informally (since the formal system to reason in was not exactly specified). Such an informal answer is: define $P\,n = T\,n \wedge Q\,n$*. Then, if the assumptions of the regular elimination axiom are satisfied, also those of the weak elimination axiom are (using that the totality predicate is assumed), yielding* $T\,n \wedge Q\,n$*, so certainly* $Q\,n$*.*

   The guest lecturer, Kenji Miyamoto from the Computation with Bounded Resources Group, was kind enough to provide his formalised solution(s) to this exercise (and the others), as developed in the minlog system: `http://www.mathematik.uni-muenchen.de/~logik/minlog/` *You can find the formalisation at:* `http://cl-informatik.uibk.ac.at/teaching/ws16/itp/exercises/ex-20170116.scm`

   Bonus Exercise for next time: Can you actually prove your/the above solution, on paper, in the natural deduction system as presented in the lecture (with, say, intro/elimination for implication, forall, and conjunction)?

3. Implement a function which given an iterator returns a recursor. An Iterator is:

   $$I : \mathbb{N} \to \tau \to (\tau \to \tau) \to \tau$$

   With the reductions

   $$I\,0\,M\,N \Longrightarrow M$$

   $$I\,(S\,n)\,M\,N \Longrightarrow N(I\,n\,M\,N)$$

   An answer:

   Comparing the Iterator to the Recursor

   $$R : \mathbb{N} \to \tau \to (\mathbb{N} \to \tau \to \tau) \to \tau$$

   with reductions

   $$R\,0\,M'\,N' \Longrightarrow M'$$

   $$R\,(S\,n)\,M'\,N' \Longrightarrow N'\,n\,(R\,n\,M'\,N')$$

   we see, just by looking at the types, that they only differ in that the recursor may/must also take the 'input' (natural number) into account in doing the recursive call, whereas the iterator cannot and must do with just the 'output' (the recursive call).

The idea to simulate R by means of I, is to make the input part of the output, that is, to iteratively return input,output pairs instead of just the output, and then finally project the result-pair thus obtained, to its right component.[2] [3] *First observe that, given M', N', we may define*

$$\begin{array}{rcl} M & = & \langle 0, M' \rangle \\ N & = & \lambda\, p.\langle S\,(\pi_1\, p), N'\,(\pi_1\, p)\,(\pi_2\, p)\rangle \end{array}$$

and show the following simulation result between iteration (for M,N thus defined) and recursion (for M',N' given), by induction on n: $I\, n\, M\, N = \langle n, R\, n\, M'\, N'\rangle$.

(base) $I\, 0\, M\, N = M = \langle 0, M'\rangle = \langle 0, R\, 0\, M'\, N'\rangle$.

(step)

$$\begin{array}{rcl} I\,(S\, n)\, M\, N & = & N\,(I\, n\, M\, N) \\ & = & (\lambda\, p.\langle S\,(\pi_1\, p), N'\,(\pi_1\, p)\,(\pi_2\, p)\rangle)\langle n, R\, n\, M'\, N'\rangle \\ & = & \langle S\, n, N'\, n\,(R\, n\, M'\, N')\rangle \\ & = & \langle S\, n, R\,(S\, n)\, M'\, N'\rangle \end{array}$$

Hence, using the above definitions of M,N, we may indeed define

$$R = \lambda\, n M' N'.\pi_2\,(I\, n\, M\, N)$$

[2] This idea goes back to Kleene for his definition of the predecessor function in the λ-calculus for Church numerals $\underline{n} = \lambda\, fx.f^n x$. More specifically, $P\, n = \pi_2\,(n\,(\lambda\, p.\langle S(\pi_1 p), \pi_1 p\rangle)\,\langle \underline{0}, \underline{0}\rangle)$ (predecessor), with $\langle M, N\rangle$ abbreviating $\lambda\, z.z\, M\, N$ (pairing), $\pi_i\, p$ abbreviating $p\,(\lambda\, x_1 x_2.x_i)$ (left projection), and $S\, n$ abbreviating $\lambda fx.n\, f\,(f\, x)$ (successor). That is the predecessor funtion is obtained by means of the second projection, from its graph, the pairs $\langle$ input, output $\rangle$:

$$\langle \underline{0}, \underline{0}\rangle, \langle \underline{1}, \underline{0}\rangle, \langle \underline{2}, \underline{1}\rangle, \langle \underline{3}, \underline{2}\rangle, \ldots$$

where to produce the next pair from the current one, we take the left component of the current pair, and use that twice to construct both components of the next pair, incremented on the left.

[3] This also means that for simulating a recursor for type τ (the output) the type of the iterator becomes a proudct type $\mathbb{N} \times \tau$ (the input,output pairs).