Sebastian Funk
sebastian.funk@gmail.com

1 Warmup questions

Do these without looking at the notes and in at most three sentences per question:

1. Write down the rules for introduction and elimination of abstract data types in System F (i.e. *pack* and *let pack*). Explicitly specify the 'client code' that doesn't get to see the abstracted type.
2. What do σ and Σ represent?
3. Write down the typing rules and operational semantics for the imperative additions to STLC.
4. Explain the reason to separate typing logic for pure and impure terms when using monads to track effects and explain how they are linked together.

2 Regular questions

Tips: Try to write proofs in one direction. Avoid writing some steps of the proof next to other ones without numbering them.

1. For each of the following PLC (System F) typing judgements, is there a PLC type A_i that make the judgement provable? (In each case, give a type A_i and typing derivation for, or explain why a typing derivation cannot exist.)
   (a) $\cdot;\cdot \vdash \lambda x : (\forall\alpha\,.\,\alpha)\,.\,(\Lambda\beta\,.\,x\ \beta) : A_1$
   (b) $\cdot;\cdot \vdash \Lambda\alpha\,.\,\lambda x : \alpha\,.\,\Lambda\beta\,.\,x\ \beta : A_2$
   (c) $\cdot;\cdot \vdash \lambda x : A_3\,.\,\Lambda\alpha\,.\,(x\ (\alpha \to \alpha)\ (x\ \alpha)) : A_3 \to \forall\beta\,.\,\beta$
   (d) $\cdot;\cdot \vdash \lambda x : A_4\,.\,\Lambda\alpha\,.\,(x\ (\alpha \to \alpha)\ (x\ \alpha)) : A_4 \to \forall\alpha\,.\,(\alpha \to \alpha)$
   (e) $\cdot;\cdot \vdash \Lambda\alpha\,.\,\lambda x : A_5\,.\,(x\ (\alpha \to \alpha)\ (x\ \alpha)) : \forall\alpha\,.\,(\alpha \to \alpha)$
2. Complete Exercises 2, 3 and 4 from Lecture 5:
   (a) Define a Church encoding for the unit type.
   (b) Define a Church encoding for the empty type.
3. The *Church numerals* are defined as follows:

$$
\begin{aligned}
c_0 &= \Lambda\alpha\,.\,\lambda z : \alpha\,.\,\lambda s : \alpha \to \alpha\,.\,z \\
c_1 &= \Lambda\alpha\,.\,\lambda z : \alpha\,.\,\lambda s : \alpha \to \alpha\,.\,s\ z \\
c_2 &= \Lambda\alpha\,.\,\lambda z : \alpha\,.\,\lambda s : \alpha \to \alpha\,.\,s\ (s\ z) \\
c_3 &= \Lambda\alpha\,.\,\lambda z : \alpha\,.\,\lambda s : \alpha \to \alpha\,.\,s\ (s\ (s\ z)) \\
&\textit{etc.}
\end{aligned}
$$

   The successor function that takes c_n to c_{n+1} is defined in the lectures as

$$suc(n) \stackrel{\text{def}}{=} \Lambda\alpha\,.\,\lambda z : \alpha\,.\,\lambda s : \alpha \to \alpha\,.\,s\ (n\ \alpha\ z\ s)$$

   Using just the variables α, n, s and z, find another way to define the successor function, *suc*, on Church numerals.
4. Complete Exercises 1, 2 and 3 from Lecture 6.

5. Complete Exercise 1 from Lecture 7. Show how Landin's knot works by unrolling fib(3).
6. This question concerns the monadic language given in Lecture 8.
   (a) Given types X and Y, define a term:
   $$\cdot;\cdot \vdash \mathsf{fmap} : (X \to Y) \to (T\,X \to T\,Y)$$
   such that for all terms $f : X \to Y$ and values $v : X$ and $v' : Y$ where $f\,v \rightsquigarrow^* v'$ we have
   $$\langle \sigma, \mathsf{let}\ y = \mathsf{fmap}\ f\ \{\mathsf{return}\ v\};\ \mathsf{return}\ y\rangle \rightsquigarrow^* \langle \sigma, \mathsf{return}\ v'\rangle$$
   (b) For every type X, define terms:
   $$\cdot;\cdot \vdash \eta_X : X \to T\,X$$
   $$\cdot;\cdot \vdash \mu_X : T\,(T\,X) \to T\,X$$
   such that for all values $v : T\,X$ and $v' : X$ where $\langle \sigma, \mathsf{let}\ x = v;\ \mathsf{return}\ x\rangle \rightsquigarrow^* \langle \sigma', \mathsf{return}\ v'\rangle$ we have
   $$\langle \sigma, \mathsf{let}\ y = \mu_X\,(\eta_{(TX)}\,v);\ \mathsf{return}\ y\rangle \rightsquigarrow^* \langle \sigma', \mathsf{return}\ v'\rangle$$
7. 2019 Paper 9 Question 14 part b-e
8. 2021 Paper 9 Question 15 part a-b
9. 2020 Paper 9 Question 15 part b

3 Extension questions (optional)

1. Write a term `and` that takes two Church booleans and returns their conjunction. (How do you change this for `or`?)
2. Write a function `equal` that takes two Church numerals and returns a Church boolean.
3. 2014 Paper 9 Question 13 part b-d
4. Assume encodings of product and existential types in System F (and associated terms, e.g. $\mathsf{fst}\,e$ and $\mathsf{pack}_{\alpha,B}(A,e)$). The signature BOOL and terms yes, no and choose are given by:
   $$\begin{aligned} \mathsf{BOOL} &= (\beta \times \beta) \times \forall\alpha.\,\beta \to \alpha \to \alpha \to \alpha & \beta &\vdash \mathsf{BOOL}\ \mathsf{type} \\ \mathsf{yes} &= \mathsf{fst}\,(\mathsf{fst}\,b) & \beta; b : \mathsf{BOOL} &\vdash \mathsf{yes} : \beta \\ \mathsf{no} &= \mathsf{snd}\,(\mathsf{fst}\,b) & \beta; b : \mathsf{BOOL} &\vdash \mathsf{no} : \beta \\ \mathsf{choose} &= \mathsf{snd}\,b & \beta; b : \mathsf{BOOL} &\vdash \mathsf{choose} : \forall\alpha.\,\beta \to \alpha \to \alpha \to \alpha \end{aligned}$$
   (a) Define a term and such that
   $$\beta; b : \mathsf{BOOL} \vdash \mathsf{and} : \beta \to \beta \to \beta$$
   which returns the conjunction of its two arguments.
   (b) Define a term extend such that
   $$\cdot;\cdot \vdash \mathsf{extend} : (\exists\beta.\,\mathsf{BOOL}) \to (\exists\beta.\,\mathsf{BOOL} \times (\beta \to \beta \to \beta))$$
   which takes an implementation of BOOL, and adds and to it.
   (c) What should the signature NAT of natural numbers be?
5. How many kinds of functions of type $a \to a$ can you write in ML? (Up to renaming variables, print-statements etc.)