Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Lukas Stevens

Lambda Calculus
Winter Term 2021/22
Solutions to Exercise Sheet 11

Exercise 1 (Church Numerals in System F)

Encode the natural numbers in System F with Church numerals. Use the construction for recursive types from the lecture.

Solution

We start from the reursive definition

$$\mathsf{nat} = \mathsf{S}\ \mathsf{nat} \mid \mathsf{Z}$$

where the constructor C_1 is S and C_2 is Z. We use the construction from the lecture to deduce the type of nat:

$$\begin{array}{ll} \tau_1 = \mathsf{nat} \rightarrow \mathsf{nat} & \tau_2 = \mathsf{nat} \\ \sigma_1 = \gamma \rightarrow \gamma & \sigma_2 = \gamma \end{array}$$

Thus $\mathsf{nat} = \forall\gamma.\ \sigma_1 \rightarrow \sigma_2 \rightarrow \gamma = \forall\gamma.\ (\gamma \rightarrow \gamma) \rightarrow \gamma \rightarrow \gamma$. Now, we derive the the terms for the constructors:

$$\mathsf{Z} = \lambda\,\gamma.\ \lambda\,f_1 : \gamma \rightarrow \gamma.\ \lambda f_2 : \gamma.\ f_2$$

$$\mathsf{S} = \lambda\,n : \mathsf{nat}.\ \lambda\,\gamma.\ \lambda\,f_1 : \gamma \rightarrow \gamma.\ \lambda f_2 : \gamma.\ f_1\ (n\ \gamma\ f_1\ f_2)$$

Exercise 2 (Programming in System F)

System F allows us to define functions that go far beyond what was possible in the simply typed λ-calculus. In particular, we can also define some non-primitively recursive functions in System F. As a prominent example, consider the Ackermann function:

$$\begin{aligned} \mathsf{ack}\ 0\ n &= n + 1 \\ \mathsf{ack}\ (m+1)\ 0 &= \mathsf{ack}\ m\ 1 \\ \mathsf{ack}\ (m+1)\ (n+1) &= \mathsf{ack}\ m\ (\mathsf{ack}\ (m+1)\ n) \end{aligned}$$

Define the Ackermann function in System F based on the encoding of natural numbers from the last exercise. *Hint*: First define a function g such that $g\ f\ n = f^{n+1}\ \underline{1}$

Solution

To understand why we need the function g, it is useful to consider ack as a function that is recursive in its first argument. Using the definition of the primitive recursor from the lecture, we can define ack in terms of the recursor on Church numerals:

$$\begin{aligned} \mathsf{rec}\ (\mathsf{S}\ n)\ \gamma\ f_1\ f_2 &= f_1\ (\mathsf{rec}\ n\ \sigma\ f_1\ f_2) \\ \mathsf{rec}\ \mathsf{Z}\ \gamma\ f_1\ f_2 &= f_2 \end{aligned}$$

This means that we need functions g, h such that

$$\begin{aligned} \mathsf{ack}\ \underline{m+1} &= g\ (\mathsf{ack}\ \underline{m}), \\ \mathsf{ack}\ \underline{0} &= h. \end{aligned}$$

Finding h is easy as $\mathsf{ack}\ \underline{0}\ n = \mathsf{S}\ n$ should hold which implies that $h = \mathsf{S}$. For finding g it helps to unfold the definition of ack on $\mathsf{ack}\ (m+1)\ n$ until $n = 0$:

$$\begin{aligned} \mathsf{ack}\ (m+1)\ n &= \mathsf{ack}\ m\ (\mathsf{ack}\ (m+1)\ (n-1)) \\ &= \mathsf{ack}\ m\ (\mathsf{ack}\ m\ (\mathsf{ack}\ (m+1)\ (n-2))) \\ &= \dots \\ &= \mathsf{ack}\ m\ (\mathsf{ack}\ m\ (\dots\ (\mathsf{ack}\ (m+1)\ 0)\ \dots)) \\ &= \mathsf{ack}\ m\ (\mathsf{ack}\ m\ (\dots\ (\mathsf{ack}\ m\ 1)\ \dots)) \\ &= (\mathsf{ack}\ m)^{n+1}\ 1 \\ &= g\ (\mathsf{ack}\ m)\ n \end{aligned}$$

Where the last equation follows from the hint. Now, the only thing left is to define g and plug g and S into the primitive recursor of nat which is just the type itself according to the lecture.

$$\begin{aligned} g = &\ \lambda f\colon \mathsf{nat} \to \mathsf{nat}.\ \lambda n\colon \mathsf{nat}.\ f\ (n\ \mathsf{nat}\ f\ \underline{1}) \\ \mathsf{ack} = &\ \lambda m\colon \mathsf{nat}.\ m\ (\mathsf{nat} \to \mathsf{nat})\ g\ \mathsf{S} \end{aligned}$$

Finally, we check that our definition satisifies the equations of the Ackermann function:

$$\mathsf{ack}\ \underline{0}\ n =_\beta\ \mathsf{S}\ n$$

$$\begin{aligned} \mathsf{ack}\ \underline{m+1}\ n &=_\beta\ \mathsf{S}\ \underline{m}\ (\mathsf{nat} \to \mathsf{nat})\ g\ \mathsf{S}\ n \\ &=_\beta\ (\lambda n\colon \mathsf{nat}.\ \lambda\gamma.\ \lambda f_1\colon \gamma \to \gamma.\ \lambda f_2\colon \gamma.\ f_1\ (n\ \gamma\ f_1\ f_2))\ \underline{m}\ (\mathsf{nat} \to \mathsf{nat})\ g\ \mathsf{S}\ n \\ &=_\beta\ g\ (\underline{m}\ (\mathsf{nat} \to \mathsf{nat})\ g\ \mathsf{S})\ n \\ &=_\beta\ g\ (\mathsf{ack}\ \underline{m})\ n \end{aligned}$$

Exercise 3 (Existential Quantification in System F)

System F can also be defined with additional existential types of the form $\exists\alpha.\ \tau$. To make use of these types, we add the following constructs to our term language

- pack τ with t as τ',

- $\mathsf{open}\ t\ \mathsf{as}\ \tau\ \mathsf{with}\ m\ \mathsf{in}\ t'$,

together with the reduction rule:

$$\mathsf{open}\ (\mathsf{pack}\ \tau\ \mathsf{with}\ t\ \mathsf{as}\ \exists\alpha.\ \tau')\ \mathsf{as}\ \alpha\ \mathsf{with}\ m\ \mathsf{in}\ t' \to t'[\tau/\alpha][t/m]$$

a) Specify the typing rules for $\exists$.

b) Show how $\exists$ can be used to specify an abstract module of sets that supports the empty set, insertion, and membership testing.

c) Show how to implement this module with lists.

d) How do these concepts relate to the SML (or OCaml) concepts of signatures, structures, and functors?

Solution

a)

$$\frac{\Gamma \vdash t\colon \tau'[\tau/\alpha]}{\Gamma \vdash \mathsf{pack}\ \tau\ \mathsf{with}\ t\ \mathsf{as}\ \exists\alpha.\ \tau' \colon \exists\alpha.\ \tau'}$$

$$\frac{\Gamma \vdash t\colon \exists\alpha.\ \tau' \qquad \Gamma, m\colon \tau' \vdash t' \colon \tau'' \qquad \alpha \text{ not free in } \Gamma, \tau''}{\Gamma \vdash \mathsf{open}\ t\ \mathsf{as}\ \alpha\ \mathsf{with}\ m\ \mathsf{in}\ t' \colon \tau''}$$

b)

$$\mathsf{setsig} = \exists\,\mathsf{set}.\ \langle \mathsf{set}, \mathsf{nat} \to \mathsf{set} \to \mathsf{set}, \mathsf{nat} \to \mathsf{set} \to \mathsf{bool}\rangle$$

c)

$$\mathsf{packed} = \mathsf{pack}\ \mathsf{list}\ \mathsf{nat}\ \mathsf{with}\ \ \mathsf{as}\ \langle \mathsf{nil}, \mathsf{cons}\,\mathsf{nat}, \ldots\rangle \mathsf{setsig}$$

$$\mathsf{open}\ \mathsf{packed}\ \mathsf{as}\ \mathsf{set}\ \mathsf{with}\ m\ \mathsf{in}\ (\lambda\, empty\, insert\, mem.\ mem\ \underline{1}\ (insert\ \underline{0}\ empty))$$
$$(\mathsf{fst}\ m)\ (\mathsf{snd}\ m)\ (\mathsf{third}\ m)$$

d)

- Signatures: existential types
- Structures: values of existential type
- Functors: functions with arguments of existential type

Homework 4 (Finger Exercises on Typing in System F)

a) Give a type τ such that

$$\vdash \lambda m\colon \mathsf{nat}.\ \ \lambda n\colon \mathsf{nat}.\ \lambda\,\alpha.\ (n\ (\alpha \to \alpha))\ (m\ \alpha)\colon \tau$$

is typeable in System F and prove the typing judgement. Recall that

$$\mathsf{nat} = \forall\,\alpha.\ (\alpha \to \alpha) \to \alpha \to \alpha\ .$$

b) Is there any typeable term t (in System F) such that if we remove all type annotations and type abstractions from t we get $(\lambda\,x.\ x\ x)\ (\lambda\,x.\ x\ x)$?

Homework 5 (Programming in System F)

Define (in System F) a function zero of type $\mathsf{nat} \to \mathsf{bool}$ that checks whether a given Church numeral is zero. Use the encoding that was introduced in the tutorial.

Homework 6 (Disjunction in System F)

Prove $\vee_{I_1}$ and $\vee_E$ from

$$A \vee B = \forall C.\ (A \to C) \to (B \to C) \to C$$

in System F. Use pure logic without lambda-terms.

Homework 7 (Progress and Preservation)

We have proved the properties of *progress* (see Exercise 7.1) and *preservation* (see Homework 7.4) for the simply typed λ-calculus. Extend our previous proofs to show that these properties also hold for System F.