

# Assignment 1

## The Untyped $\lambda$ -Calculus

15-814: Types and Programming Languages  
Jan Hoffmann & C. B. Aberlé

Due Tuesday, September 10, 2024  
75 pts

This assignment is due on the above date and it must be submitted electronically on Gradescope <https://www.gradescope.com/courses/838981>. Please use the attached template to typeset your assignment and make sure to include your full name and Andrew ID. For the written problems, you may also submit handwritten answers that have been scanned and are **easily legible**.

Please carefully read the [policies on collaboration and credit](https://www.cs.cmu.edu/~janh/courses/814/24/) on the course web pages at <https://www.cs.cmu.edu/~janh/courses/814/24/>.

You should upload the following files to Gradescope:

- `hw01.pdf` with your written solutions to the questions.
- `hw01.lam` with the code, where the solutions to the problems are clearly marked and auxiliary code (either from lecture or your own) is included so it passes the LAMBDA checker.

In the following, tasks marked with **Lambda** involves programming in LAMBDA. They should be completed in the aforementioned `*.lam` file. Please *do not* present the solution on paper.

## 1 Calculating in the $\lambda$ -Calculus

**Task 1 (Lambda 5 pts)** Implement the following boolean functions.

1. The “*and*” operator such that

$$\begin{aligned} \text{and true true} &=_{\beta} \text{true} \\ \text{and true false} &=_{\beta} \text{false} \\ \text{and false true} &=_{\beta} \text{false} \\ \text{and false false} &=_{\beta} \text{false} \end{aligned}$$

2. The “*or*” operator such that

$$\begin{aligned} \text{or true true} &=_{\beta} \text{true} \\ \text{or true false} &=_{\beta} \text{true} \\ \text{or false true} &=_{\beta} \text{true} \\ \text{or false false} &=_{\beta} \text{false} \end{aligned}$$

3. The “nor” operator such that

$$\begin{aligned} \text{nor true true} &=_{\beta} \text{false} \\ \text{nor true false} &=_{\beta} \text{false} \\ \text{nor false true} &=_{\beta} \text{false} \\ \text{nor false false} &=_{\beta} \text{true} \end{aligned}$$

**Task 2 (35 pts)** A prominent subset of the computable functions on the natural numbers are the *primitive recursive* functions. They include functions like addition, multiplication, predecessor, and (iterated) exponentiation. More generally, they can be characterized as the set of total computable functions that does not “grow incredibly fast” (like Ackermann’s function).

The set of primitive recursive functions  $\mathbf{Pr}$  is inductively generated by the constant functions, successor, projection, composition, and primitive recursion. For a natural number  $n \in \mathbb{N}$ , we write  $\mathbb{N}^n \rightarrow \mathbb{N}$  for the set of  $n$ -ary functions whose domain is a  $n$ -tuple of natural numbers and codomain is the natural numbers. Define  $\mathbf{Pr}$  as the smallest subset of  $\mathbb{N}^n \rightarrow \mathbb{N}$  such that the following holds:

1. The function  $\text{const}_{n,k} : \mathbb{N}^n \rightarrow \mathbb{N}$  defined by  $(x_1, \dots, x_n) \mapsto k$  is in  $\mathbf{Pr}$  for all  $n, k \in \mathbb{N}$ .
2. The function  $\text{succ} : \mathbb{N} \rightarrow \mathbb{N}$  defined by  $n \mapsto n + 1$  is in  $\mathbf{Pr}$ .
3. The function  $\text{proj}_{n,k} : \mathbb{N}^n \rightarrow \mathbb{N}$  defined by  $(x_1, \dots, x_n) \mapsto x_k$  is in  $\mathbf{Pr}$  for all  $n > 0$  and  $1 \leq k \leq n$ .
4. Given  $f : \mathbb{N}^n \rightarrow \mathbb{N} \in \mathbf{Pr}$  and a sequence of  $k$ -ary functions  $g_1, \dots, g_n : \mathbb{N}^k \rightarrow \mathbb{N}$ , each of which is in  $\mathbf{Pr}$ , the function  $\text{comp}(f, g_1, \dots, g_n) : \mathbb{N}^k \rightarrow \mathbb{N}$  defined by  $x \mapsto f(g_1(x), \dots, g_n(x))$  is in  $\mathbf{Pr}$ .
5. Given  $f : \mathbb{N}^n \rightarrow \mathbb{N} \in \mathbf{Pr}$  and  $g : \mathbb{N}^{n+2} \rightarrow \mathbb{N} \in \mathbf{Pr}$ , let  $\text{rec}(f, g) : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$  be the unique function satisfying the following:

$$\begin{aligned} \text{rec}(f, g)(0, x) &= f(x) \\ \text{rec}(f, g)(n + 1, x) &= g(n + 1, \text{rec}(f, g)(n, x), x) \end{aligned}$$

Then  $\text{rec}(f, g)$  is in  $\mathbf{Pr}$ .<sup>12</sup>

Consider the following representation of natural numbers in the lambda calculus:

$$\begin{aligned} \bar{0} &= \lambda s. \lambda z. z \\ \overline{n + 1} &= \lambda s. \lambda z. s(\bar{n} s z) \end{aligned}$$

We say that a function  $f : \mathbb{N}^n \rightarrow \mathbb{N}$  is *realized* by a lambda calculus expression  $\bar{f}$  when

$$(((\bar{f} \bar{x}_1) \bar{x}_2) \dots) \bar{x}_n) =_{\beta} \overline{f(x_1, x_2, \dots, x_n)}$$

<sup>1</sup>Note that  $x \in \mathbb{N}^n$  is a tuple/vector, so notational convention dictates that  $f(x)$  is  $f(x_1, \dots, x_n)$  and  $\text{rec}(f, g)(0, x)$  is  $\text{rec}(f, g)(0, x_1, \dots, x_n)$ . Feel free to expand or compress vector notation in your proof to make it easier to understand.

<sup>2</sup>To simplify the encoding, we use the recursion scheme of *weak primitive recursion*. It has been shown that it is equivalent to the scheme of primitive recursion in an [article by Fischer, Fischer, and Beigel](#).

for all natural numbers  $x_1, x_2, \dots, x_n$ . In the above  $\bar{f}$  is also called the *realizer*, and when a function  $f$  has a realizer, we say that  $f$  is *realizable*. Prove that any  $f \in \mathbf{Pr}$  is realizable.

*Hint.* Your proof should proceed by induction on the construction of a given function  $f \in \mathbf{Pr}$ . The cases for the constant functions and projections are given as examples.

**Case:**  $f$  is  $\text{const}_{n,k}$  for some  $n, k$ . Define  $\overline{\text{const}_{n,k}} \triangleq \lambda y_1, \dots, y_n. \bar{k}$ . Let  $x_1, \dots, x_n \in \mathbb{N}$ . Compute:

$$\begin{aligned} (((\overline{\text{const}_{n,k}} \bar{x}_1) \cdots) \bar{x}_n) &= (((\lambda y_1, \dots, y_n. \bar{k}) \bar{x}_1) \cdots) \bar{x}_n \\ &=_{\beta} \bar{k} \\ &= \overline{\text{const}_{n,k}(x_1, \dots, x_n)} \end{aligned}$$

**Case:**  $f$  is  $\text{proj}_{n,k}$  for some  $n, k$  such that  $n > 0$  and  $1 \leq k \leq n$ . Define  $\overline{\text{proj}_{n,k}} \triangleq \lambda y_1, \dots, \lambda y_n. y_k$ . Observe that this expression is well-scoped since  $1 \leq k \leq n$ . Let  $x_1, \dots, x_n \in \mathbb{N}$ . Compute:

$$\begin{aligned} (((\overline{\text{proj}_{n,k}} \bar{x}_1) \cdots) \bar{x}_n) &= (((\lambda y_1, \dots, y_n. y_k) \bar{x}_1) \cdots) \bar{x}_n \\ &=_{\beta} \bar{x}_k \\ &= \overline{\text{proj}_{n,k}(x_1, \dots, x_n)} \end{aligned}$$

*Note.* You may assume that  $\beta$ -equivalence is a congruence for the lambda calculus, that is, you can use the following two lemmas without proof:

1. If  $e =_{\beta} e'$  then  $(\lambda x. e) =_{\beta} (\lambda x. e')$ .
2. If  $e_1 =_{\beta} e'_1$  and  $e_2 =_{\beta} e'_2$  then  $(e_1 e_2) =_{\beta} (e'_1 e'_2)$ .

Now fill in the remaining cases for the induction, given below:

**Case:**  $f$  is  $\text{succ}$ ...

*Hint.* To show that a function  $f$  is realizable, you should construct a  $\lambda$ -term  $\bar{f}$  and then argue that  $\bar{f}$  is a realizer for  $f$ .

**Case:**  $f$  is  $\text{comp}(h, g_1, \dots, g_n)$  for some primitive recursive functions  $h : \mathbb{N}^n \rightarrow \mathbb{N}$  and  $g_1, \dots, g_n : \mathbb{N}^k \rightarrow \mathbb{N}$ ...

*Hint.* Make use of the induction hypothesis, by which you may assume that  $h, g_1, \dots, g_n$  are all realizable.

**Case:**  $f$  is  $\text{rec}(h, g)$  for some primitive recursive functions  $h : \mathbb{N}^n \rightarrow \mathbb{N}$  and  $g : \mathbb{N}^{n+2} \rightarrow \mathbb{N}$ ...

*Hint.* There are (at least!) two valid ways to go about constructing a  $\lambda$ -term that realizes  $\text{rec}(h, g)$ , both of which we sketch for you:

1. Use the  $Y$ -combinator to define  $\text{rec}(h, g)$  recursively. In this case, you may freely use the combinator  $\text{pred}$  defined in the lectures, and assume that it realizes the predecessor function. You may also wish to make use of the  $\text{if0}$  combinator you define in Task 4 below.

2. Let  $\text{aux}(h, g) : \mathbb{N}^{n+1} \rightarrow \mathbb{N}^2$  be the function given by

$$\text{aux}(h, g)(m, x_1, \dots, x_n) = (k, \text{rec}(h, g)(m, x_1, \dots, x_n))$$

Show that there is a  $\lambda$ -term corresponding to  $\text{aux}(h, g)$ , and then define  $\text{rec}(h, g)$  as  $\text{proj}_{2,2} \circ \text{aux}(h, g)$ . In this case, you may need the following helper functions for pairs:

- $\text{pair} = \lambda x. \lambda y. \lambda p. p \ x \ y$
- $\text{fst} = \lambda x. \lambda y. x$
- $\text{snd} = \lambda x. \lambda y. y$

In both cases, you will need to argue by induction on  $m \in \mathbb{N}$  that for all  $x \in \mathbb{N}^n$  your realizer  $\bar{f}$  has the property that  $\bar{f} \ \bar{m} \ \bar{x} =_{\beta} \overline{\text{rec}(h, g)(m, x)}$ .

**Task 3 (0pts)** The unary representation of natural numbers requires tedious and error-prone counting to check whether your functions (such as the Lucas function in the exercise below) behave correctly on some inputs with large answers. Fortunately, you can exploit that the LAMBDA implementation counts the number of reduction steps for you and prints it in decimal form!

Given the representation of natural numbers in [nat.lam](#), the number of reductions steps used when normalizing  $\bar{n} \ \text{succ} \ \text{zero}$  is  $3n + 2$ . You may find this useful for reading a normalized numeral. For instance, the second line in the following code should produce 8 upon execution:

```
norm n = lucas zero
norm _ = n succ zero
```

**Task 4 (Lambda 20 pts)** Implement the following functions. You may use all the functions in [nat.lam](#) as helper functions.

(i)  $\text{if0}$  (definition by cases) satisfying the specification

$$\begin{aligned} \text{if0 } \bar{0} \ x \ y &=_{\beta} x \\ \text{if0 } \overline{k+1} \ x \ y &=_{\beta} y \end{aligned}$$

(ii)  $\text{even}$  satisfying the specification

$$\begin{aligned} \text{even } \overline{2k} &=_{\beta} \text{true} \\ \text{even } \overline{2k+1} &=_{\beta} \text{false} \end{aligned}$$

(iii)  $\text{half}$  satisfying the specification

$$\begin{aligned} \text{half } \overline{2k} &=_{\beta} \overline{k} \\ \text{half } \overline{2k+1} &=_{\beta} \overline{k} \end{aligned}$$

**Task 5 (Lambda 15 pts)** The Lucas function (a variant on the Fibonacci function) is defined as follows:

$$\begin{aligned} L \ 0 &= 2 \\ L \ 1 &= 1 \\ L \ (n+2) &= L \ n + L \ (n+1) \end{aligned}$$

Implement a function `lucas` that realizes  $L$ . You may use the functions from [nat.lam](#) as helper functions, as well as those from Task 4. Test your implementation on inputs 0, 1, 9, and 11, expecting results 2, 1, 76, and 199. Include these tests in your code submission `hw01.lam`.