Assignment 1
The Untyped λ-Calculus

15-814: Types and Programming Languages
Frank Pfenning

Due Thursday, September 9, 2021
60 pts

This assignment is due on the above date and it must be submitted electronically on Gradescope. 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 on the course web pages at `http://www.cs.cmu.edu/~fp/courses/15814-f20/assignments.html`.

You should hand in two files

- `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.

1 Calculating in the λ-Calculus

Task 1 (L1.1, 5 pts) Define the following functions on Booleans.

1. The "*nor*" operator, which yields *true* iff both inputs are *false*.
2. The conditional "*if*" such that
$$\begin{array}{lcl} \mathit{if}\ \mathit{true}\ e_1\ e_2 & =_\beta & e_1 \\ \mathit{if}\ \mathit{false}\ e_1\ e_2 & =_\beta & e_2 \end{array}$$
3. In the solution file `hw01.lam` include the necessary definitions of *nor* and *if* and also sufficient test cases to certify their correctness.

Task 2 (L2.3, 15 pts) One approach to representing functions defined by the schema of primitive recursion is to change the representation so that $\overline{n}$ is not an iterator but a *primitive recursor*.

$$\begin{array}{lcl} \overline{0} & = & \lambda s.\, \lambda z.\, z \\ \overline{n+1} & = & \lambda s.\, \lambda z.\, s\, \overline{n}\, (\overline{n}\, s\, z) \end{array}$$

1. Define the successor function *succ* on this new representation (if possible) and show its correctness.

2. Define the predecessor function *pred* on this new representation (if possible) and show its correctness.
3. Explore if it is possible to directly represent any function f specified by a schema of primitive recursion, ideally without constructing and destructing pairs. Write what you find.

Task 3 (L2.4, 10 pts) 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 or reduction steps for you and prints it in decimal form!

(i) We have

$$\overline{n}\ \mathit{succ}\ \mathit{zero} \longrightarrow_\beta^* \overline{n}$$

because $\overline{n}$ iterates the successor function n times on 0. Run some experiments in LAMBDA and conjecture how many leftmost-outermost reduction steps are required as a function of n. Note that only β-reductions are counted, and *not* replacing a definition (for example, *zero* by $\lambda s.\, \lambda z.\, z$). We justify this because we think of the definitions as taking place at the metalevel, in our mathematical domain of discourse.

(ii) Prove your conjecture from part (i), using induction on n. It may be helpful to use the mathematical notation $f^k c$ to describe a λ-expression generated by $f^0\, c = c$ and $f^{k+1}\, c = f\, (f^k\, c)$ where f and c are λ-expressions. For example, $\overline{n} = \lambda s.\, \lambda z.\, s^n\ z$ or $\mathit{succ}^3\ \mathit{zero} = \mathit{succ}\ (\mathit{succ}\ (\mathit{succ}\ \mathit{zero}))$.

Task 4 (L2.5, 15 pts) Define the following functions in the λ-calculus using the LAMBDA implementation. Here we take "$=$" to mean $=_\beta$, that is, β-conversion.

You may use all the functions in nat.lam as helper functions. Your functions should evidently reflect iteration, primitive recursion and pairs. In particular, you should avoid the use of the Y combinator which will be introduced in Lecture 3.

Provide at least 3 test cases for each function and include them, together with your function definitions, in the file `hw01.lam`.

(i) if0 (definition by cases) satisfying the specification

$$\begin{array}{lcl} \mathsf{if0}\ \overline{0}\ x\ y & = & x \\ \mathsf{if0}\ \overline{k+1}\ x\ y & = & y \end{array}$$

(ii) even satisfying the specification

$$\begin{array}{lcl} \mathsf{even}\ \overline{2k} & = & \mathsf{true} \\ \mathsf{even}\ \overline{2k+1} & = & \mathsf{false} \end{array}$$

(iii) half satisfying the specification

$$\begin{array}{lcl} \mathsf{half}\ \overline{2k} & = & \overline{k} \\ \mathsf{half}\ \overline{2k+1} & = & \overline{k} \end{array}$$

Task 5 (L2.6, 15 pts) The Lucas function (a variant on the Fibonacci function) is defined mathematically by

$$\begin{array}{lcl} \mathsf{lucas}\ 0 & = & 2 \\ \mathsf{lucas}\ 1 & = & 1 \\ \mathsf{lucas}\ (n+2) & = & \mathsf{lucas}\ n + \mathsf{lucas}\ (n+1) \end{array}$$

Give an implementation of the Lucas function in the λ-calculus via the LAMBDA implementation.

You may use the functions from nat.lam as helper functions, as well as those from Task 4. Your functions should evidently reflect iteration, primitive recursion and pairs. In particular, you should avoid the use of the Y combinator which will be introduced in Lecture 3.

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`, and record the number of β-reductions used by your function in your written submission.