Assignment 1
The Untyped λ-Calculus

15-814: Types and Programming Languages
Frank Pfenning

Due Tuesday, September 15, 2020

This assignment is due on the above date and it must be submitted electronically as a PDF file on Canvas. Please use the attached template to typeset your assignment and make sure to include your full name and Andrew ID.

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. Exclusive or "*xor*".
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 *xor* 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* (if possible) and show its correctness.
2. Define the predecessor function *pred* (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.

Task 3 (L3.1, 10 pts) The unary representation of natural numbers requires tedious and error-prone counting to check whether your functions (such a factorial, Fibonacci, or greatest common divisor in the exercises 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 (L3.3, 15 pts) The Fibonacci function is defined by

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

Give two implementations of the Fibonacci function in the λ-calculus (using the LAMBDA implementation).

(i) Exploit the idea behind the encoding of primitive recursion using pairs to give a direct implementation of fib without using the Y combinator.

(ii) Give an implementation of fib using the Y combinator.

You may copy the functions from nat.lam to the beginning of your file `hw01.lam`. Test your implementation on inputs 0, 1, 9, and 11, expecting results 0, 1, 34, and 89. Which of the two is more "efficient" (in the sense of number of β-reductions)?

Task 5 (L3.4, 15 pts) Recall the specification of the greatest common divisor (*gcd*) from this lecture for natural numbers $a, b > 0$:

$$\begin{array}{lcll} \mathit{gcd}\ a\ a & = & a & \\ \mathit{gcd}\ a\ b & = & \mathit{gcd}\ (a-b)\ b & \text{if } a > b \\ \mathit{gcd}\ a\ b & = & \mathit{gcd}\ a\ (b-a) & \text{if } b > a \end{array}$$

We don't care how the function behaves if $a = 0$ or $b = 0$.

Define *gcd* as a closed expression in the λ-calculus over Church numerals. You may use the Y combinator we defined, and any other functions like *succ*, *pred*, etc. from file nat.lam). but you have to define other functions you may need such as subtraction or arithmetic comparisons.

Test your implementation on several examples.

Analyze how your function behaves when one or both of the arguments a and b are $\overline{0}$.