# CS704: Assignment 2

Questions 2 and 3 are adapted from TAPL.

## 1 Fixed-point combinators

**A** The most popular encoding of recursion in $\lambda$-calculus is using the Y combinator. Another approach is using the simple U combinator (which is not a fixed-point combinator):

$$U =_{\mathrm{df}} \lambda x.\ x\ x$$

(We use $=_{\mathrm{df}}$ to denote a definition, as opposed to semantic equivalence.) Using the U combinator, we can define the factorial function as follows:

$$\mathit{fact} =_{\mathrm{df}} U(\lambda f.\ \lambda n.\ \ \mathit{if}\ (n = 0)\ \mathit{then}\ 1\ \mathit{else}\ n * ((f f)(n - 1)))$$

(For clarity, we extend pure lambda calculus with conditionals and arithmetic.)
Prove that *fact* satisfies the following $\lambda$-calculus equation:

$$\mathit{fact} = (\lambda n.\ \mathit{if}\ (n = 0)\ \mathit{then}\ 1\ \mathit{else}\ n * (\mathit{fact}(n - 1)))$$

**B** Consider the following theorem for characterizing fixed-point combinators themselves as fixed points:

$$\text{Let } G = \lambda y.\lambda f.f(yf).\text{ Then } M \text{ is a fixed-point combinator if and only if } M = GM. \qquad (1)$$

(Note: Recall that the following $\lambda$-calculus transformation is called the $\eta$-reduction rule:

$$(\lambda x.Mx) \to_\eta M,$$

where $x$ does not occur as one of the free variables of $M$. You are allowed to use $\eta$-reduction in this question.)

**subpart (i)**

Use Theorem 1 to show that $Y =_{\mathrm{df}} \lambda f.((\lambda x.f(xx))(\lambda x.f(xx)))$ is a fixed-point combinator.

**subpart(ii)**

Use Theorem 1 to show that the U combinator ($\lambda x.\ x\ x$) is a not a fixed-point combinator.

**subpart (iii)**

Prove Theorem 1. (Note that the theorem involves an "if and only if"; consequently, your proof should have two parts.)

**subpart (iv)**

The Y combinator allows us to find a $\lambda$-term $g$ that satisfies a single recursive equation over $\lambda$-terms of the form $g = \ldots g \ldots g \ldots$

Suppose that we are presented with a collection of $k$ mutually recursive equations:

$$\begin{array}{rcl} g_1 & = & \ldots g_1 \ldots g_k \ldots \\ & \vdots & \\ g_k & = & \ldots g_1 \ldots g_k \ldots \end{array}$$

Explain how to solve for $g_1, \ldots, g_k$.

## 2 Church Encodings

**A** In lecture, we defined Church numerals and Booleans, along with some operations over them. Define a lambda term that tests equality of two Church numerals, returning $\lambda t.\lambda f.t$ when they are equal, and $\lambda t.\lambda f.f$ when they are not equal.

**B** Suppose you are given a combinator `pred` that computes the predecessor of a Church numeral. Use `pred` to define subtraction in lambda calculus. (Note that, given 0, `pred` returns 0.)

## 3 Typed Lambda Calculus

Is there any context $\Gamma$ and type $\tau$ such that $\Gamma \vdash x\ x : \tau$. If so, provide a $\Gamma$ and $\tau$ and show the typing derivation; if not, prove that there does not exist such a $\Gamma$ and $\tau$.