Types and Programming Languages (15-814)
Fall 2018
Assignment 0: The untyped λ-calculus
(Sample Solutions)

Contact: 15-814 Course Staff
Due Thursday, September 13, 2018

This assignment is due at the beginning of class 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. Questions subject to the whiteboard policy are marked by “WB”.

15-814 Programming Language Award

The Programming Language Award recognizes a programming language that has
had significant influence on computing, as reflected in contributions to academic
research, commercial acceptance, or both. The award is presented each fall at the
15-814 Awards Brunch and is accompanied by a prize of one lecture dedicated to
the chosen language, plus food expenses at the brunch.

Next Deadline Thursday September 13, 2018, 10:30am EDT.

Selection Criteria Nominations will be reviewed for the evidence they provide
of originality, significant conceptual impact, influence on related developments,
beauty, and elegance.

Submissions Nominations for the Programming Language Award should be
submitted on Canvas. Submitted materials should explain the contribution in
terms understandable to a non-specialist. Each nomination involves several
components.

• Name and Andrew ID of the nominator.
• Name of the programming language nominated.

• URL with key information about the language, which could reference a paper, home page, or an implementation.

• Suggested citation if the language is selected. This should be a concise statement (maximum of 25 words) describing the language and its benefits that have had impact. Note that the final wording will be at the discretion of the Award Committee.

• Nomination statement 450 to 500 words in length addressing why the language should receive this award. This should draw particular attention to the contributions that merit the award.

• Brief timeline with key dates, individuals responsible for its creation, publications, websites, implementations, etc.

• Names of at least 3 and not more than 5 researchers or practitioners who might, if asked, write a brief letter of endorsement of the language. Hypothetical endorsers should be chosen to represent a range of perspectives and institutions and could provide additional insights or evidence of the language’s significance. Do not contact hypothetical endorsers!

For questions on the above, please contact us on Piazza. 15-814’s academic integrity guidelines apply to all award nominations.

Task 1 (20 points). Nominate a language of your choice for the Programming Language Award.

Solution 1: Your nomination here.

Programming with the \( \lambda \)-calculus

In class, we saw how to encode the natural numbers using Church numerals. We will explore various other representations of numerals.

Task 2 (5 points, WB). Define functions \( \text{pair} \), \( \text{projl} \), and \( \text{projr} \) such that \( \text{projl} \ (\text{pair} \ x \ y) = x \) and \( \text{projr} \ (\text{pair} \ x \ y) = y \).

Solution 2: This portion was universally correct:

\[
\begin{align*}
\text{pair} &= \lambda x.\lambda y.\lambda z.zxy, \\
\text{projl} &= \lambda p.p(\lambda x.\lambda y.x), \\
\text{projr} &= \lambda p.p(\lambda x.\lambda y.y).
\end{align*}
\]
Using pairings, we can now define numerals:

\[
\begin{align*}
\gamma 0 & = \lambda x.x \\
\gamma (n + 1) & = \text{pair}\ F\ \gamma n
\end{align*}
\]

where

\[
\begin{align*}
T & = \lambda x.\lambda y.x \\
F & = \lambda x.\lambda y.y
\end{align*}
\]

are the booleans from class.

**Task 3** (5 points, WB). Implement the successor, predecessor, and test for zero functions:

\[
\begin{align*}
S\ \gamma n & = \gamma (n + 1), \\
P\ \gamma n + 1 & = \gamma n, \\
Z\ \gamma 0 & = T, \\
Z\ \gamma n + 1 & = F.
\end{align*}
\]

**Solution 3:** The successor function was universally correct: \(S = \text{pair}\ F\).

The predecessor function was also universally correct: \(P = \text{proj}r\). Some added an extra case for \(n = \gamma 0\) to get \(P\gamma 0 = \gamma 0\). An example answer of this form is \(P = \lambda n.\ Zn\gamma 0 (\text{proj}\ n)\).

The zero test function was nearly as successful: \(Z = \text{proj}l\). This works because

\[
\begin{align*}
Z\ \gamma 0 & = \text{proj}l\ \gamma 0 = \gamma 0 (\lambda x.\lambda y.x) = T, \\
Z\ \gamma n + 1 & = \text{proj}l\ \gamma n + 1 = \text{proj}l\ (\text{pair}\ F\ \gamma n) = F.
\end{align*}
\]

This numbering system is equivalent to system of the Church numerals we saw in class, where we defined:

\[
\begin{align*}
\bar{0} & = \lambda z.\lambda s.z \\
\bar{n + 1} & = \lambda z.\lambda s.s(\bar{n}zs)
\end{align*}
\]

The next task asks you to implement a function mapping between our two definitions of natural numbers.

**Task 4** (10 points, WB). Implement functions \(H\) and \(H^{-1}\) such that \(H\pi = \gamma n\) and \(H^{-1}\gamma n = \pi\) for all \(n\). Then implement the predecessor function \(\text{pred}\) for Church numerals, satisfying \(\text{pred}\ \bar{n + 1} = \bar{n}\).

**Solution 4:** This task proved to be a bit harder. Here are the simplest solutions:

\[
\begin{align*}
H & = \lambda n.\ n\gamma 0\ S \\
H^{-1} & = Y (\lambda h.\lambda x.\text{if}\ (Zx)(\bar{0}) (\text{succ}(h(Px)))) \\
\text{pred} & = \lambda n.\ H^{-1}(P(H(n))).
\end{align*}
\]
Many students needlessly complicated this task by not reusing the helper functions they defined in previous tasks. A good approach for this type of problem is to write down the function on paper in pseudocode, and then to translate it into the language, using helper functions as appropriate.

In defining $H$, some students (unsuccessfully) tried to use recursion. This usually did not work because they failed to account for a base case or failed to make the argument decrease. The key idea behind Church numerals is that they act as iterators. The Church numeral $\bar{n}$ takes in a base case $z$ and a function $s$ and iteratively applies $sn$ times to $z$. Consequently, to translate $\bar{n}$ to $\lfloor n \rfloor$, it is sufficient to specify what zero should be ($\lfloor 0 \rfloor$) and what the successor function should be (S) and to apply $\bar{n}$ to these.

Some students struggled with $H^{-1}$. A common mistake was having $H^{-1}$ on both sides of the equation: you needed to use the $Y$ combinator here. Another common mistake was failing to specify a base case, in which case the recursion unfolded forever.

Most students saw that they could straightforwardly get the predecessor function from the parts they had defined so far. 

\[
\Box
\]