Assignment 1
The Untyped λ-Calculus

15-814: Types and Programming Languages
Frank Pfenning

Due Tuesday, September 17, 2019

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-f19/assignments.html.

1 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.

Nomination Deadline Tuesday, September 17, 2019.

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.

2 Calculating in the λ-Calculus

Task 2 (L2.1, 10pts) 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 (L2.2, 10pts) We know we can represent all functions on Booleans returning Booleans once we have exclusive or. But we can also represent the more general conditional *if* with the requirements

$$\begin{array}{lcl} \textit{if true}\ e_1\ e_2 & = & e_1 \\ \textit{if false}\ e_1\ e_2 & = & e_2 \end{array}$$

Give a definition of *if* in the λ-calculus and verify (showing each step) that the equations above are satisfied using β-conversion.

Task 4 (L2.3, 20pts) Recall the specification of the greatest common divisor (*gcd*) from this lecture for natural numbers $a, b > 0$:

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

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 this lecture and *if* from Task 3, but you have to define other functions you may need such as subtraction or arithmetic comparisons.

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