Assignment 3
Polymorphism

15-814: Types and Programming Languages
David M Kahn & Frank Pfenning

Due Thursday September 23, 2021
75 points

This assignment is due on the above date and it must be submitted electronically as a PDF file on Canvas. You may use the distributed lecture notes and assignment sources to typeset your assignment and make sure to include your full name and Andrew ID on the written part.

You should hand in two files

- `hw03.pdf` with your written solutions to the questions
- `hw03.poly` 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 Proof by Rule Induction

Task 1 (L5.2, 40 points) Define a new single-step relation $e \mapsto e'$ which means that e reduces to e' by *leftmost-outermost reduction*, using a collection of inference rules. Intuitively, a single step of leftmost-outermost reduction operates by attempting to apply the leftmost lambda possible in the term, such that functions are applied before their body or arguments are reduced. This strategy is *sound* (it only performs β-reductions) and *complete for normalization* (if e has a normal form, we can reach it by performing only leftmost-outermost reductions).

Prove the following two statements about your reduction judgment. For reference, we provide rules for general β-reduction $\longrightarrow$ here.

(i) If $e \mapsto e'$ then $e \longrightarrow e'$

(ii) $\mapsto$ is *small-step deterministic*, that is, if $e \mapsto e_1$ and $e \mapsto e_2$ then $e_1 = e_2$.

You should interpret $=$ as α-equality, that is, the two terms differ only in the names of their bound variables (which we always take for granted). For each of the following statements, either indicate that they are true (without proof) or provide a counterexample.

(iii) For all e, either $e \mapsto e'$ for some e' or e *normal*.

(iv) There does not exist an e such that $e \mapsto e'$ for some e' and e *normal*.

(v) If $e \longrightarrow e'$ then $e \mapsto e'$.

(vi) $\longrightarrow$ is *small-step deterministic*.

(vii) $\longrightarrow$ is *big-step deterministic*, that is, if $e \longrightarrow^* e_1$ and $e \longrightarrow^* e_2$ where e_1 *normal* and e_2 *normal*, then $e_1 = e_2$.

(viii) For arbitrary e and *normal* e', $e \longrightarrow^* e'$ iff $e \mapsto^* e'$.

2 Polymorphic Functions

Task 2 (L3.2, 10 points) Fill in the blanks in the following judgments so that it holds, or indicate there is no way to do so. You do not need to justify your answer or supply a typing derivation, and the types do not need to be "most general" in any sense. As always, feel free to use LAMBDA to check your answers.

(i) $\boxed{} \vdash \forall\alpha.\, \alpha \to \beta$ *type*

(ii) $\boxed{} \vdash\ \Lambda\alpha.\, x\,[\alpha \to \alpha]\, y\,[\beta] : \forall\alpha.\, \beta$

(iii) $\cdot \vdash \lambda x.\, x\,[\boxed{}]\, x\, x : \boxed{}$

(iv) α *type* $\vdash \boxed{} : \forall\beta.\, \alpha \to \beta$

(v) $x : \forall\alpha.\, (\forall\beta.\, \beta \to \beta) \to \alpha, \gamma$ *type* $\vdash \boxed{} : (\gamma \to \gamma) \to \gamma$

Task 3 (L6.1, 10 pts)

(i) In lecture we defined *plus* : *nat* $\to$ *nat* $\to$ *nat* using the polymorphic λ-calculus with

$$\lambda n.\, \lambda k.\, n\,[\mathit{nat}]\, \mathit{succ}\, k$$

In doing so, we instantiated $\mathit{nat} = \forall\alpha.(\alpha \to \alpha) \to \alpha \to \alpha$ with *nat* itself. Such an approach would not work using the simply-typed definition of *nat*, $(\alpha \to \alpha) \to \alpha \to \alpha$, which has no way of instantiating its type variables. For this task, find a definition of *plus* : *nat* $\to$ *nat* $\to$ *nat* using the polymorphic defition of *nat* that *would* work in the simply-typed λ-calculus, in the sense that *nat* is only ever instantiated with a type variable.

(ii) Give a simply-typed definition (in the sense of part (i)) for *times* or give a conjecture that none exists.

Include your definition(s) with least 3 test cases each in the file `hw03.poly`, and in your `hw03.pdf` state either conjecture and reasoning, or that you have found a simply-typed definition.

Task 4 (L6.3, 15 pts) While we didn't quite get to it in lecture, the self-application $\lambda x.\, x\,[u]\, x$ can be typed polymorphically with $u \to u$ by setting $u = \forall\alpha.\, \alpha \to \alpha$. It turns out that this is just one instance of a whole family of types for self-application. Consider a (mathematical) function[1] F from

[1] Note that such a function not expressible in the polymorphic λ-calculus but requires system F^ω

types to types. A more general family of types and terms for self-application can be parameterized by F, given by u_F and ω_F below:

$$\begin{array}{lcl} u_F & = & \forall\alpha.\,\alpha \to F(\alpha) \\ \omega_F & : & u_F \to F(u_F) \\ \omega_F & = & \lambda x.\,x\ [u_F]\ x \end{array}$$

We can find the original type in this family where $F = \Lambda\alpha.\,\alpha$. You may want to verify the general typing derivation in preparation for the following questions, but you do not need to show it.

(i) Consider $F = \Lambda\alpha.\,\alpha \to \alpha$. In this case $u_F = \mathit{bool}$. Calculate the type and characterize the behavior of ω_F as a function on Booleans.

(ii) Consider $F = \Lambda\alpha.\,(\alpha \to \alpha) \to \alpha$. Calculate u_F, the type of ω_F, and characterize the the behavior of ω_F. Can you relate u_F and ω_F to the types and functions we have considered in the course so far?

For both parts, include your functions with their type in the file `hw03.poly` together with a few test cases that demonstrate your interpretation of how ω_F behaves in those two instances. And of course, include the written answers in `hw03.pdf`.