Types and Programming Languages (15-814), Fall 2018
Assignment 5: Polymorphism and Parametricity

Contact: 15-814 Course Staff

Due Tuesday, October 30, 2018, 11:59pm

This assignment is due by 11:59pm 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. As before, problems marked "WB" are subject to the whiteboard policy; all other problems must be done individually.

Task 1 (0 points)**.** How long did you spend on this assignment? Please list the questions that you discussed with classmates using the whiteboard policy.

1 Polymorphic Encoding of Data Types

In this problem we return to functional encodings of data types, but now using explicit polymorphism.

Task 2 (5 points, WB)**.** As a warm-up exercise, find a polymorphic type for self-application $\omega = \lambda x.\, x\, x$. Your type does not need to be most general.

Next we propose the following encoding

$$\alpha\ \mathbf{list} \quad = \quad \forall\gamma.\, \gamma \to (\alpha \to \gamma \to \gamma) \to \gamma$$

Task 3 (5 points, WB)**.** Give definitions for the following functions *nil* and *cons* with the types

$$\begin{array}{lcl} \mathit{nil} & : & \forall\alpha.\, \alpha\ \mathbf{list} \\ \mathit{cons} & : & \forall\alpha.\, \alpha \to \alpha\ \mathbf{list} \to \alpha\ \mathbf{list} \end{array}$$

You may define auxiliary functions as you see fit and show them with their explicit types.

Task 4 (5 points, WB)**.** The function *hd* that returns the first element of a list requires a default argument to handle the case of the empty list. Define a function

$$\mathit{hd} : \forall\alpha.\, \alpha\,\mathbf{list} \to \alpha \to \alpha$$

which satisfies

$$\begin{array}{lcl} \mathit{hd}\ \mathit{nil}\ y & \simeq & y \\ \mathit{hd}\ (\mathit{cons}\ x\ l)\ y & \simeq & x \end{array}$$

where $\simeq$ is the usual Kleene equality.

2 Parametricity

Task 5 (10 points, WB)**.** Prove the following "free theorem" in the polymorphic λ-calculus where all functions are parametric.

Theorem 1. For any types τ, τ', σ, σ' and values $f : \tau \to \tau'$, $g : \sigma \to \sigma'$, $x : \tau$, $y : \sigma$ and $k : \forall\alpha.\forall\beta.\,\alpha \to \beta \to \alpha$ we have

$$f\,(k\,x\,y) \sim k\,(f\,x)\,(g\,y) : \tau'$$

Task 6 (5 points, WB)**.** Can you think of a straightforward consequence of the theorem above, using it for more specific f and g?

3 Data Abstraction

Reconsider the example from Assignment 4 with two different representations of integers: one as a pair $\langle x, y\rangle : \mathit{nat} \otimes \mathit{nat}$ of natural numbers x and y where $a = x - y$, and another one as a sum $(\mathsf{pos} : \mathit{nat}) + (\mathsf{neg} : \mathit{nat})$.

For ease of writing, we use a higher-level syntax also used in Assignment 4 and Lecture 10 and 13. Concretely, we use

```
data Nat = Z | S Nat

data Diff = D (Nat, Nat)
data PosNeg = Pos Nat | Neg Nat
```

and a signature

```
INT = {
  type Int
  zero : Int
  inc : Int -> Int
  dec : Int -> int
}
```

representing the type $\exists\alpha.\, \alpha \otimes (\alpha \to \alpha) \otimes (\alpha \to \alpha)$. In your answers you may define functions by pattern matching, including defining auxiliary functions as needed.

Task 7 (5 points, WB)**.** Define an implementation

```
DIFF : INT = { ... }
```

where the type `Int` $=$ `Diff`. You may adapt your implementation or our sample solution from Assignment 4.

Task 8 (5 points, WB)**.** Define an alternative implementation

```
POSNEG : INT = { ... }
```

where the type `Int` $=$ `PosNeg`.

Task 9 (10 points, WB)**.** Define a relation R : `Diff` $\leftrightarrow$ `PosNeg` to relate values of the two different implementation types. As usual, we will assume it is closed under Kleene equality.

Task 10 (10 points, WB)**.** Prove that `DIFF` $\sim$ `POSNEG` : `INT`, using the relation R from the previous task.

There is a lot of flexibility in how you define the two implementations and how you define the relation between them. The key is simplicity: for complex implementations or relations, the proof may be difficult or impossible. If you notice that matters become too messy, reexamine your implementations and definitions to consider how you might simplify them. Moreover, it is easy to make a mistake in the relation, so you need to be careful with the details of your proof.