# Assignment 4
# Defining Types

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

Due Thursday, September 30, 2021
60 points

This assignment is due on the above date at 11:59 pm and it must be submitted electronically on Gradescope. 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

- `hw04.pdf` with your written solutions to the questions
- `hw04.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 Evaluation and Reduction

**Task 1 (L7.1, 10 pts)** For each of the following statements, provide either a proof or counter example. Consider only the case for $e$ in the untyped $\lambda$-calculus from earlier in the course, and recall that a closed expression is one with no free variables.

(i) For closed $e$, if $e$ *value* then $e$ *normal*.

(ii) For closed $e$, if $e$ *normal* then $e$ *value*.

## 2 Structure of Types

**Task 2 (L7.2, 15 pts)** An alternative form of binary tree to the one given in Lecture 7.3 is one where the natural numbers are stored in the leaves and not in the nodes. Let's call such a tree a *shrub*.

(i) Give the types for shrub constructors.

(ii) Give the construction of a shrub containing the numbers 1, 2, and 3.

(iii) Give the polymorphic definition of the type *shrub*, assuming it is represented by its own iterator.

(iv) Provide the definitions of the shrub constructors.

(v) Write a function *sumup* to sum the elements of a shrub.

(vi) Write a function *mirror* that returns the mirror image of a given tree, reflected about a vertical line down from the root.

Include your definitions in the file `hw04.poly` together with a few test cases.

**Task 3 (L7.3 10 pts)** We say two types $\tau$ and $\sigma$ are *isomorphic* (written $\tau \cong \sigma$) if there are two functions *forth* : $\tau \to \sigma$ and *back* : $\sigma \to \tau$ such that they compose to the identity in both directions, that is, $\lambda x.\, \mathit{back}\ (\mathit{forth}\ x))$ is equal to $\lambda x.\, x$ and $\lambda y.\, \mathit{forth}\ (\mathit{back}\ y)$ is equal to $\lambda y.\, y$.

Consider the two types

$$\begin{array}{rcl} \mathit{nat} & = & \forall\alpha.\, (\alpha \to \alpha) \to \alpha \to \alpha \\ \mathit{tan} & = & \forall\alpha.\, \alpha \to (\alpha \to \alpha) \to \alpha \end{array}$$

(i) Provide functions *forth* : *nat* $\to$ *tan* and *back* : *tan* $\to$ *nat* that, intuitively, should witness the isomorphism between *nat* and *tan*.

(ii) Compute the normal forms of the two function compositions. You may recruit the help of the LAMBDA implementation for this purpose.

(iii) Are the two function compositions $\beta$-equal to the identity? If yes, you are done. If not, can you see a sense under which they would be considered equal, either by changing your two functions or be defining a suitably justified notion of equality?

Include your functions *forth* and *back* as well as their compositions in the file `hw04.poly`.

## 3 Lazy Pairs

**Task 4 (L8.6, 20 points)** *Lazy pairs*, constructed as $\langle\!|e_1, e_2|\!\rangle$, are an alternative to the eager pairs $\langle e_1, e_2\rangle$. Lazy pairs are typically available in "lazy" languages such as Haskell. The key differences are that a lazy pair $\langle\!|e_1, e_2|\!\rangle$ is always a value, whether its components are or not. In that way, it is like a $\lambda$-expression, since $\lambda x.\, e$ is always a value. The second difference is that its destructors are $\mathsf{fst}\ e$ and $\mathsf{snd}\ e$ rather than a new form of case expression.

We write the type of lazy pairs as $\tau_1 \,\&\, \tau_2$. In this exercise you are asked to design the rules for lazy pairs and check their correctness.

1. Write out the new rule(s) for $e$ *value*.
2. State the typing rules for new expressions $\langle\!|e_1, e_2|\!\rangle$, $\mathsf{fst}\ e$, and $\mathsf{snd}\ e$.
3. Give evaluation rules for the new forms of expressions.

Instead of giving the complete set of new proof cases for the additional constructs, we only ask you to explicate a few items. Nevertheless, you need to make sure that the progress and preservation continue to hold.

4. State the new clause in the canonical forms theorem.
5. Show one case in the proof of the preservation theorem where a destructor is applied to a constructor.

6. Show the case in the proof of the progress theorem analyzing the typing rule for $\mathsf{fst}\, e$.

**Task 5 (L8.8, 5 points)** It is often stated that lazy pairs are not necessary in an eager language, because we can already define $\tau_1 \mathbin{\&} \tau_2$ and the corresponding constructors and destructors. Fill in this table.

| | | |
|---|---|---|
| $\tau_1 \mathbin{\&} \tau_2$ | $\triangleq$ | $(1 \to \tau_1) \times (1 \to \tau_2)$ |
| $\langle e_1, e_2 \rangle$ | $\triangleq$ | |
| $\mathsf{fst}\, e$ | $\triangleq$ | |
| $\mathsf{snd}\, e$ | $\triangleq$ | |