Exercises – Introduction to Type Theory

Exercise 1 – Types and terms

1. Find a term t which has type $\mathbb{N} \to (\mathbb{N} \to \mathbb{N})$.

   NB: when we say that a term t has type A, we mean that t has type A *in the empty context* unless specified otherwise – that is, the judgment $\vdash t : A$ holds.

2. Find a term t which has type $A \to (A \to A)$ for any type A.

3. Write a derivation tree which shows that $\lambda\,(t : A \times B)\,.\,\langle\,\pi_2(t)\,,\pi_1(t)\,\rangle$ has type $A \times B \to B \times A$ for any two types A, B.

4. ($\star$) Is it possible to find a type A such that the judgment $\vdash \lambda\,t\,.\,t\ t : A$ holds?

Exercise 2 – The empty type

We extend our simple type theory with an *empty type* $\bot$, which has no inhabitants. For intuition, you might want to think of the empty set in math, or a type of exceptions in functional programming. The empty type has a single typing rule:

$$\frac{\Gamma \vdash t : \bot}{\Gamma \vdash \text{raise}(t) : A}\ \textsc{Empty-Elim}$$

and does not need any computation rule.

1. Find a term t which has type $A \times (A \to \bot) \to B$ for any two types A, B.

2. Find a term t which has type $A \to ((A \to \bot) \to \bot)$ for any type A.

3. ($\star\star$) Is it possible to find a type t which has type $((A \to \bot) \to \bot) \to A$?

Exercise 3 – Sum types

We extend our simple type theory with *sum types*. The sum type $A + B$ contains a copy of all the elements of A, and a copy of all the elements of B. For intuition, you might want to think about disjoint unions in mathematics, or algebraic datatypes in functional programming.

The rules for sum types are given in the document on simple type theory (*stt.pdf*).

1. Find a term t which has type $A + B \to B + A$ for any two types A, B.

2. Find a term t which has type $(A + B \to C) \to (A \to C) \times (B \to C)$ for any types A, B, C.

3. ($\star\star$) Find a term t which has type $((A + (A \to \bot)) \to \bot) \to \bot$ for any type A.