

### Exercise 1 (Type Inference in Haskell)

In this exercise, we will develop a type inference algorithm for the simply typed  $\lambda$ -calculus in Haskell. The general idea of the algorithm is to apply the type inference rules in a backward manner and to record equality constraints between types on the way. These constraints are then solved to obtain the result type.

- Take a look at the template provided on the website. We have provided definitions of terms and types in the simply typed  $\lambda$ -calculus, together with syntax sugar for input and printing. Moreover, you can find the type of substitutions and utility functions to work with substitutions, types and terms.
- The first component of the algorithm is *unification* on types. Given a list of equality constraints between types of the form  $u_1 \stackrel{?}{=} t_1, \dots, u_n \stackrel{?}{=} t_n$ , we want to produce a suitable substitution  $\phi$  such that  $\phi(u_i) = t_i$  for all  $1 \leq i \leq n$  or report that the given constraints do not have a solution. Fill in the remaining cases of the function *solve* that achieves this functionality.
- Now we want to apply the type inference rules and record the arising type constraints. Function *constraints* of type

$$Term \rightarrow Type \rightarrow Env \rightarrow (Int, [(Type, Type)]) \rightarrow Maybe (Int, [(Type, Type)])$$

will achieve this functionality. Given a term  $t$ , a type  $\tau$ , an environment  $\Gamma$ , and a pair  $(n, C)$ , it will try to justify  $\Gamma \vdash t : \tau$ , adding the arising type constraints to  $C$ . The natural number  $n$  is used to keep track of the least variable index that is currently unused. This allows to easily generate fresh variable names. Complete the definition of *constraints*.

- Define the function *infer* that infers the type of a term by combining *solve* and *constraints* and try it on a few examples.

### Exercise 2 (Every Type is Applicative)

- Show that every type is *substitutive*.
- Show that every type is *applicative*.

### Exercise 3 (Alternative Proof of Lemma 3.2.3)

- Show that for every  $s \in T$ ,  $s x \in T$  if  $x$  is fresh with respect to  $s$ .
- Show that every substitutive type is *applicative*.

## Homework 4 (Types of Church Numerals)

a) Let  $\tau$  be any type. Show that for the  $n$ -th Church numeral  $\underline{n}$ , we have

$$[] \vdash \underline{n} : (\tau \rightarrow \tau) \rightarrow \tau \rightarrow \tau$$

b) Show that every term  $t \in \text{NF}$  with  $[] \vdash t : (\iota \rightarrow \iota) \rightarrow \iota \rightarrow \iota$ ,  $t$  is either `id` or a church numeral. Here  $\iota$  is any *elementary* type.

## Homework 5 (Completeness of $T$ )

In this exercise, you will show the converse of Lemma 3.2.2, i.e.

$$\Downarrow t \implies t \in T$$

a) Show that every  $\lambda$ -term has one of the following shapes:

- $x r_1 \dots r_n$
- $\lambda x.r$
- $(\lambda x.r) s_1 \dots s_n$

Note that this gives rise to an alternative inductive definition for  $\lambda$ -terms and to a corresponding rule induction on  $\lambda$ -terms.

b)  $\Downarrow$  gives rise to a wellfounded induction principle. To show

$$\forall t. \Downarrow t \implies P(t),$$

it suffices to prove:

$$\forall t. (\forall t'. t \rightarrow_\beta t' \implies P(t')) \implies P(t).$$

Use this to prove:

$$\Downarrow t \implies t \in T$$

*Hint:* Use (a) for an inner induction on terms.