

Exercise 1 (Constraint Solving)

During type inference we generate type unification constraints between types of the form $\tau_1 \stackrel{?}{=} \kappa_1, \dots, \tau_n \stackrel{?}{=} \kappa_n$. In order to solve these constraints, we want to find a substitution function σ such that $\sigma(\tau_i) = \kappa_i$ for $i \in \{1, \dots, n\}$. For each of the following constraint systems, find such a substitution or justify that no such substitution exists. Additionally, give a λ -term that has the type $\sigma(\tau_0)$.

- a) $\tau_0 \stackrel{?}{=} \tau_1 \rightarrow \tau_2, \tau_1 \stackrel{?}{=} \tau_2$
- b) $\tau_0 \stackrel{?}{=} \tau_1 \rightarrow \tau_2, \tau_2 \stackrel{?}{=} \tau_3 \rightarrow \tau_4, \tau_4 \stackrel{?}{=} \tau_1$
- c) $\tau_0 \stackrel{?}{=} \tau_1 \rightarrow \tau_2, \tau_2 \stackrel{?}{=} \tau_3 \rightarrow \tau_4, \tau_1 \stackrel{?}{=} \tau_5 \rightarrow \tau_4, \tau_1 \stackrel{?}{=} \tau_3 \rightarrow \tau_5$
- d) $\tau_0 \stackrel{?}{=} \tau_1 \rightarrow \tau_2, \tau_1 \stackrel{?}{=} \tau_3 \rightarrow \tau_2, \tau_3 \stackrel{?}{=} \tau_1$

Exercise 2 (Type Inference in Haskell)

In this exercise, we will develop a type inference algorithm for the simply typed λ -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.

- a) Take a look at the template provided on the website. We have provided definitions of terms and types in the simply typed λ -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.
- b) 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 ϕ 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.
- c) Now we want to apply the type inference rules and record the arising type constraints. Function *constraints* of type

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

will achieve this functionality. Given a term t , a type τ , an environment Γ , 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*.

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

Exercise 3 (Every Type is Applicative)

- a) Show that every type is *substitutive*.
- b) Show that every type is *applicative*.

Homework 4 (Types of Church Numerals)

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

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

b) Show that every term $t \in \text{NF}$ with $\Box \vdash t : (\iota \rightarrow \iota) \rightarrow \iota \rightarrow \iota$, t is either `id` or a church numeral. Here ι 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 λ -term has one of the following shapes:

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

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