1 Monomorphic example

We’ll refer to the four type judgement rules as Const, Var, λ, App.

The goal is to type \( \lambda f. \lambda x. \lambda y. \text{suc}(f(x + y)) \).

Goal: find \( \alpha \) such that \( \vdash \lambda f. \lambda x. \lambda y. \text{suc}(f(x + y)) : \alpha \): apply \( \lambda \), putting \( \alpha = \beta \rightarrow \gamma \), giving subgoal:

\[
\vdash f : \beta \vdash \lambda x. \lambda y. \text{suc}(f(x + y)) : \gamma \quad \text{apply \( \lambda \), putting \( \gamma = \delta \rightarrow \epsilon \), giving subgoal}
\]

\[
\vdash f : \beta, x : \delta, y : \zeta \vdash \text{suc}(f(x + y)) : \eta \quad \text{apply \( \text{App} \), giving two subgoals}
\]

\[
\vdash f : \beta, x : \delta, y : \zeta \vdash \text{suc} : \theta \rightarrow \eta \quad \text{apply \( \text{Const} \), with \( \theta = \text{nat} \rightarrow \text{nat} \)}
\]

\[
\vdash f : \beta, x : \delta, y : \zeta \vdash f : \iota \rightarrow \theta \quad \text{apply \( \text{Var} \), with \( \beta = \iota \rightarrow \theta \)}
\]

\[
\vdash f : \beta, x : \delta, y : \zeta \vdash (x + y) : \iota \quad \text{apply \( \text{App} \), giving two subgoals}
\]

\[
\vdash f : \beta, x : \delta, y : \zeta \vdash + : \kappa \rightarrow \iota \quad \text{apply \( \text{Const} \), with \( \omega = \kappa \rightarrow \iota \)}
\]

\[
\vdash f : \beta, x : \delta, y : \zeta \vdash x : \omega \quad \text{apply \( \text{Var} \), with \( \omega = \delta \)}
\]

\[
\vdash f : \beta, x : \delta, y : \zeta \vdash y : \kappa \quad \text{apply \( \text{Var} \), with \( \kappa = \zeta \)}
\]

Now solve the underlined equations by substitution and matching, giving \( \omega = \kappa = \iota = \zeta = \delta = \theta = \eta = \text{nat} \) and the rest accordingly.

\[\square\]
2 Polymorphic example

The example was \( \text{let id} = \lambda x. x \in \langle \text{id}(1), \text{id}(\text{true}) \rangle \). Here is the inference algorithm:

Goal: find \( \alpha \) such that \( \vdash \text{let id} = \lambda x. x \in \langle \text{id}(1), \text{id}(\text{true}) \rangle : \alpha \). Apply Let, giving two subgoals

**Note:** the use of the meta-variable \( \sigma \) indicates that we will solve the type inference for the first goal completely, before plugging it into the generalization in the second subgoal

\[ \vdash \lambda x. x : \sigma \]: apply \( \lambda \), with \( \sigma = \beta \rightarrow \beta \), giving subgoal

\[ \vdash x : \beta \vdash x : \beta \]: apply Const □

There are no other equations, so the most general type for \( \lambda x. x \) is \( \beta \rightarrow \beta \)

Now the second subgoal of Let is:

\[ \vdash \text{id} : \forall \beta. \beta \rightarrow \beta \vdash \langle \text{id}(1), \text{id}(\text{true}) \rangle : \alpha \]

We calculated \( \sigma = \beta \rightarrow \beta \), so apply the generalization giving

\[ \vdash \text{id} : \forall \beta. \beta \rightarrow \beta \vdash \langle \text{id}(1), \text{id}(\text{true}) \rangle : \alpha \] apply Prod, with \( \alpha = \gamma \times \delta \), giving two subgoals

\[ \vdash \text{id} : \forall \beta, \beta' \rightarrow \beta \vdash \text{id}(1) : \gamma \] apply App, giving two subgoals

\[ \vdash \text{id} : \forall \beta, \beta' \rightarrow \beta \vdash \beta : \gamma \] apply Var, giving two subgoals

Now we instantiate \( \forall \beta, \beta \rightarrow \beta \) with a fresh type variable to \( \beta' \rightarrow \beta' \)

\[ \vdash \text{id} : \forall \beta, \beta' \rightarrow \beta \vdash 1 : \epsilon \] apply Const with \( \epsilon = \text{nat} \)

\[ \vdash \text{id} : \forall \beta, \beta' \rightarrow \beta \vdash \text{id}(\text{true}) : \delta \] apply App, giving two subgoals

Now solve the underlined equations, getting \( \zeta = \beta'' = \delta = \text{bool} \) and \( \epsilon = \beta' = \gamma = \text{nat} \), and thus \( \alpha = \text{nat} \times \text{bool} \) (Note that \( \beta \) does not appear, because it was solved out and generalized in a subproof.)