TYPE INFERENCE

Problem: Given a term e, what is its type?

An issue here is that the answer is not necessarily unique:

$$\text{recfun } f\ x = x \ ::\ \text{Int} \to \text{Int},\ \forall a.\ a \to a$$

The second type is more general than the first.

Ex. 1 What are the most-general types of

1. $\text{recfun } f\ x = ((\text{fst } x) + 1)$ $\quad \forall a.\ (\text{Int} \times a) \to \text{Int}$

2. $\text{InL}\ (\text{InR True})$ $\quad \Rightarrow \quad \forall a.b.\ ((b + \text{Bool}) + a)$

   $\forall b.\ b + \text{Bool}$

3. $\text{recfun } f\ x = \text{if } (\text{fst } x) \text{ then } (\text{snd } x) \text{ else } f\ x$

   $$\forall a.(\text{Bool} \times a) \to a$$

4. $\text{recfun } f\ x = \text{if } (\text{fst } x) \text{ then } (\text{InL } x) \text{ else } (\text{InR } x)$

   $x : \forall a.\ (\text{Bool} \times a)$

   $\forall a.b.\ (\text{Bool} \times a) + b$ $\qquad$ $\forall a.b.$ $\ b + (\text{Bool} \times a)$

   $\Rightarrow\ b := \text{Bool} \times a$

) m.g.t. is $\forall a.\ (\text{Bool} \times a) \to ((\text{Bool} \times a) + (\text{Bool} \times a))$

Ideal: have an algorithm to do this for us.

Attempt 1: use typing rules directly to derive an algo.

What breaks?

①
②
③

Attempt 2. use unification variables for any presently unknown types during derivations. Write $F\,\alpha$ for a flexible type α.

Unify: a substitution for unification variables that makes two types equal:

$$\begin{cases} (\alpha \times \beta) \to \alpha \;\sim\; (\mathrm{Int} \times \mathrm{Bool}) \to \gamma \\ \text{unifier is } [\alpha = \mathrm{Int},\ \beta = \mathrm{Bool},\ \gamma = \mathrm{Int}] \end{cases}$$

Add this to a typing context

What we want often is the most general unifier of 2 types.

Typing judgments look like

$$\underbrace{\Gamma}_{\text{Context}} \vdash \underbrace{e}_{\text{Expr.}} \Rightarrow \underbrace{\tau}_{\text{Determined type}} \dashv \underbrace{\Gamma'}_{\text{updated context}}$$

inputs / outputs of the algo.

The updated context is minimal in the sense that it adds as little info as possible to infer the type (i.e. unifies)

EXISTENTIAL TYPES.

Goal: add abstract data types (stacks, trees, ...)

We do this via existential types:

$\exists a.\tau$ = Produce picks the type a,
Consume cannot speculate about what a is
(This is the opposite of how $\forall a.\tau$ works!)

Think of this like how when #including a C header file, you can't assume impl. details, have to work with just the interface

Ex. 3 Suggest non-$\exists$ alts. to these types:

① $\exists t.\ t \times (t \to \text{String})$ String

② $\exists t.\ t \times (\text{Int} \times t \to t)$ ⇝ 1.

③ $\exists t.\ (t \to t \times t) \times (t \times t \to t)$ ⇝ 1.