Type Inference

needs

Revolution

Conor McBride

Milner's Coincidence

syntax	terms	types
visibility	seen	unseen
agency	human	machine
code-gen	retained	erased
quantification	$\sigma \to \tau$	$\forall \alpha . \sigma$
	vacuous	dependent

Unpicking two aspects of type inference

let

$$\frac{\Gamma, \vec{\alpha} \vdash s : \sigma \qquad \Gamma, x : \forall \vec{\alpha}.\sigma \vdash t : \tau}{\Gamma \vdash \mathrm{let}\ x = s\ \mathrm{in}\ t : \tau}$$

versus

var

$$\frac{\Gamma, x : \forall \vec{\alpha}.\sigma, \Delta \vdash \tau_i\ \mathsf{TYPE}}{\Gamma, x : \forall \vec{\alpha}.\sigma, \Delta \vdash x : \sigma[\vec{\tau}/\vec{\alpha}]}$$

where do these come from?

What makes type synthesis tricky?

$$\frac{\Gamma \vdash f : (x : \sigma) \to \tau \qquad \Gamma \vdash s : \sigma}{\Gamma \vdash f\ s : \tau[s/x]}$$

But the wheel was already a-wobbling...

- show • read

- data Tm α = Lam (Tm (Maybe α)) | ...

  rename f (Lam b) =
    Lam (rename (fmap f) b)

  ...

- the obstinate disembodied SPJ

Quantifiers in Haskell (speculative fiction)

	machine	← agency →	human
erased	$\forall x :: \sigma .\ \tau$		$\forall x :: \sigma \to \tau$
↕ codegen			
retained (dependent)	$\Pi x :: \sigma .\ \tau$	dependent	$\Pi x :: \sigma \to \tau$
retained (vacuous)	$\phi \Rightarrow \tau$	vacuous	$\sigma \to \tau$

Use-site versus definition; run-time relevance

$$\mathit{downfall} : \mathrm{Vec}\ \mathbb{N}\ n$$
$$\mathit{downfall}_{0} = []$$
$$\mathit{downfall}_{\mathrm{suc}\ n} = n, \mathit{downfall}$$

After Milner's Coincidence

visibility
code-gen

syntax - gone

quantification - dependent

agency

human | machine

var-rule stuff

type as spec

seen | unseen

details details

erased
retained

Bidirectionalism illustrated in four rules

lam

brain-fade

$$\frac{\Gamma, x : S \vdash T \ni t}{\Gamma \vdash (x : S) \to T \ni \lambda x . t}$$

cod

$$\frac{\Gamma \vdash e \in S \qquad \Gamma \vdash S \le T}{\Gamma \vdash T \ni \underline{e}}$$

ditto

may your gods go with you

var

$$\frac{}{\Gamma, x : S, \Delta \vdash x \in S}$$

app

$$\frac{\Gamma \vdash f \in (x : S) \to T \qquad \Gamma \vdash S \ni s}{\Gamma \vdash f\, s \in T[\underbrace{s : S}_{\text{what?}} / x]}$$

Bidirectionalism with small-step computation

doc

$$\frac{\Gamma \vdash \mathsf{Type} \ni T \qquad \Gamma \vdash T \ni t}{\Gamma \vdash t : T \in T}$$

β $\quad (\lambda x.\, t : (x : S) \to T)\; s \leadsto t[s:S/x] : T[s:S/x]$

γ $\quad \underline{t : T} \leadsto t$

pre cook

$$\frac{\blacksquare\; T \leadsto^{*} R \qquad \Gamma \vdash R \ni t}{\Gamma \vdash T \ni t}$$

postcook

$$\frac{\Gamma \vdash e \in S \qquad S \leadsto^{*} R}{\Gamma \vdash e \in R}$$

Digression - Effects

- either "see what happens"

$$\frac{\Gamma \vdash f : \sigma \xrightarrow[\varepsilon_3]{} \tau,\ \varepsilon_1 \qquad \Gamma \vdash s : \sigma,\ \varepsilon_2}{\Gamma \vdash f\ s : \tau,\ \varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3}$$

accumulate

- or check "what's permitted"

confirm

$$\frac{\Gamma ; \varepsilon \vdash f : \sigma \xrightarrow[\varepsilon']{} \tau \qquad \Gamma ; \varepsilon \vdash s : \sigma \qquad \varepsilon' \subseteq \varepsilon}{\Gamma ; \varepsilon \vdash f\ s : \tau}$$

distribute

Types vs Type schemes

don't put them in your kernel

they're too hard to guess well

$$
\left.
\begin{aligned}
\mathrm{SCH} ::= \;& \mathrm{TYPE} \\
\mid\; & (x : \mathrm{SCH}) \to \mathrm{SCH} \\
\mid\; & \underbrace{\{?x : \mathrm{TYPE}\}}_{x \text{ to be guessed}} \to \mathrm{SCH}
\end{aligned}
\right\} \text{from declarations}
$$

Unification or Bust?

- types are not a first-order theory
  Miller → solve the equations which look like definitions

- programs get stuck
  constraints go yes, no and maybe

- but there are other ways to fill a hole

Type-Directed Construction

- explore the typed but unfinished
- down with random ascii

what about type errors?

Design Errors are the new Type Errors

BANZAI!

is not enough – we need

BANJAXED?

Turn the world around!

Types are the inputs to
"type inference",
a term which no longer makes sense.