Telescopic [Constraint] Trees

Or: Information-Aware Type Systems In Context

Philippa Cowderoy

August 31, 2018

Terms, Typings and Telescopes

Consider this Simply-Typed Lambda Calculus term:

$(\lambda x.x)$ 42

We might give it this typing
(with a metavariable $\tau = \mathbb{N}$):

$$\dfrac{\begin{array}{c}\dfrac{\dfrac{}{\{x : \tau,\ 42 : \mathbb{N}\} \vdash x : \tau}\ \textsc{Var}}{\{42 : \mathbb{N}\} \vdash (\lambda x.x) : \tau \to \tau}\ \textsc{Lam} \\ \vdots \end{array} \qquad \dfrac{}{\{42 : \mathbb{N}\} \vdash 42 : \mathbb{N}}\ \textsc{Var}}{\{42 : \mathbb{N}\} \vdash (\lambda x.x)\ 42 : \mathbb{N}}\ \textsc{App}$$

Telescopes Grow Branches

If we take that typing:

$$\dfrac{\begin{array}{c}\dfrac{\dfrac{}{\{x:\tau,\ 42:\mathbb{N}\}\vdash x:\tau}\ \textsc{Var}}{\{42:\mathbb{N}\}\vdash(\lambda x.x):\tau\to\tau}\ \textsc{Lam}\\ \vdots\end{array}\qquad\dfrac{}{\{42:\mathbb{N}\}\vdash 42:\mathbb{N}}\ \textsc{Var}}{\{42:\mathbb{N}\}\vdash(\lambda x.x)\ 42:\mathbb{N}}\ \textsc{App}$$

We can "trace" the telescopes found in it to find this corresponding tree (where aligned | shows branches):

$$\begin{array}{ll}\{42:\mathbb{N}\} & |\{x:\tau\}\\ & |\{\}\end{array}$$

More
What is it Good For? Uh...

Given $(\lambda x.x)$ 42 and $\begin{array}{ll} \{42 : \mathbb{N}\} & |\{x : \tau\} \\ & |\{\} \end{array}$

- We can line up the λ with the telescope containing x
- We can line up the application with the branches
- We can see the RHS is a variable
  - It's a leaf with an empty telescope
- From τ we can see the application decides the type of $\lambda x.x$

...Not bad! But it could be better. And what's with τ?
Time for more information!

Information-Aware Simply-Typed λ-Calculus
(Inevitably)

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}\,Var \qquad \frac{\begin{array}{c}\Gamma f := \Gamma \,;\, x : \tau p \\ \Gamma f \vdash T : \tau r \\ \tau f = \tau p \rightarrow \tau r\end{array}}{\Gamma \vdash \lambda x.T \;:\; \tau f}\,Lam$$

$$\frac{\begin{array}{c}\Gamma \prec^{\Gamma L}_{\Gamma R} \\ \Gamma L \vdash Tf : \tau f \qquad \Gamma R \vdash Tp : \tau p \\ \tau p \rightarrow \tau r = \tau f\end{array}}{\Gamma \vdash Tf\;\, Tp : \tau r}\,App$$

Constraints for the Simply-Typed Lambda Calculus

Here's how the IASTLC works now:

$\tau = \tau$	Type equality
$\tau \prec^{\tau l}_{\tau r}$	Type duplication
$x : \tau \in \Gamma$	Binding in context
$\Gamma' := \Gamma \,;\, x : \tau$	Context extension
$\Gamma \prec^{\Gamma L}_{\Gamma R}$	Context duplication

These need turning into something more telescopic

Context Constraints In, um, Context?

Our typing rules use these constraints, which refer to contexts:

$x : \tau \in \Gamma$	Binding in context
$\Gamma' := \Gamma \,;\, x : \tau$	Context extension

We can't put those directly in a telescope - they want to refer to it!

But we can translate them into these:

Old	New	Description
$x : \tau \in \Gamma$	$?x : \tau$	Query/Ask for binding [here]
$\Gamma' := \Gamma \,;\, x : \tau$	$!x : \tau$	Generate/Tell about binding [here]

Freebies and Paperwork

Duplications are free!

- The IASTLC never directly duplicates types
- Duplicating contexts branches them already

All we have left to handle is equality and metavariables:

$\tau = \tau$	Type equality
$\exists\tau$	Bind an unknown τ
$\exists\tau{=}\tau$	Bind τ with current solution

Distinguishing solved forms makes it easier to ask “is this solved?”

Once More, With Information

A sketch typing for $(\lambda x.x)\ 42$ – initial context is $\{42 : \mathbb{N}\}$:

$$\dfrac{\dfrac{\dfrac{}{x}\ \textsc{Var}}{\lambda x.x}\ \textsc{Lam} \qquad \dfrac{}{42}\ \textsc{Var}}{(\lambda x.x)\ 42}\ \textsc{App}$$

A telescopic tree, with constraints:

$$\begin{array}{ll} \{42 : \mathbb{N}\}, \{\exists a\}, \{\exists f, \exists p, p \rightarrow a = f\} \ldots & \\ \ldots & |\{\exists r, \exists \tau, !x : \tau, f = \tau \rightarrow r\}, \{?x : r\} \\ & |\{?42 : p\} \end{array}$$

(Decon)Structural Laws

There's a 'sensible' set of structural laws to be had.
Here's one thing they permit:

$$\begin{array}{ll} \multicolumn{2}{l}{\{\exists a, \exists f, \exists p, \exists r, \exists \tau\}, \{p \to a = f, f = \tau \to r\}, \{42 : \mathbb{N}\} \ldots} \\ \ldots & |\{!x : \tau, ?x : r\} \\ & |\{?42 : p\} \end{array}$$

Let's solve the contextual constraints next.

Context Solving

$$\begin{array}{ll} \{\exists a, \exists f, \exists p, \exists r, \exists \tau\}, \{p \to a = f, f = \tau \to r\}, \{42 : \mathbb{N}\} \ldots & \\ \ldots \quad & |\{x : \tau, r = \tau\} \\ & |\{p = \mathbb{N}\} \end{array}$$

Pull all the remaining constraints towards the head of tree:

$$\begin{array}{lll} \{\exists a, \exists f, \exists p, \exists r, \exists \tau\}, \{p \to a = f, f = \tau \to r, r = \tau, p = \mathbb{N}\} \ldots & & \\ \ldots \quad & \{42 : \mathbb{N}\} \quad & |\{x : \tau\} \\ & & |\{\} \end{array}$$

That's our earlier tree with a typical constraint store at the head.

Information-Aware systems and working 'in context' seem to play well together, at least.

∃s Are Good

A great philosopher once wrote:

$$\frac{\Gamma \vdash e : \sigma \qquad \alpha \notin \mathrm{free}(\Gamma)}{\Gamma \vdash e : \forall \alpha . \sigma}\,\mathit{Gen}$$

∃s Are Good

A great philosopher once wrote:

$$\frac{\Gamma \vdash e : \sigma \qquad \alpha \notin \mathrm{free}(\Gamma)}{\Gamma \vdash e : \forall \alpha . \sigma}\,\textit{Gen}$$

Naughty, naughty, very naughty!

- Takes a creative step to justify and explain
- Can't tell which systems the rule works in
- Was implemented in systems where it was unsound!

$\exists$s Are Good

A great philosopher once wrote:

$$\frac{\Gamma \vdash e : \sigma \qquad \alpha \notin \mathrm{free}(\Gamma)}{\Gamma \vdash e : \forall \alpha . \sigma}\,\textit{Gen}$$

Naughty, naughty, very naughty!

- Takes a creative step to justify and explain
- Can't tell which systems the rule works in
- Was implemented in systems where it was unsound!

Partial solution: $\exists$ can give type variables/metavariables a scope

Back to Front

The Gen rule uses “free in the context before this point” as an alternative to “solved and unconstrained in the context ahead of this point”.

Talking about the ”context before this point” – Γ – is standard. Typing rules had no way to talk about the contexts involved in typing subterms. What about us?

- Let’s add $\fatsemi$ - a separator. (Thanks to Gundry et al!)
- No moving $\exists$ over $\fatsemi$. New structure to our structural rules!
- Our contexts, telescopes and telescopic trees are now broken up into regions

Constraints That Make You Go 'H-M'

We're going to skip the actual typing rules today. Our context will now bind *n*-ary polytypes, with typing rules adjusted accordingly, instantiation on use and generalisation constraints followed by a separator at let.

Here are some constraints we do need:

$\exists\sigma$	Polytype binding
$\exists\sigma{=}\forall\bar{a}.\tau$	(Part-)Solved polytype
$\sigma \geq \tau$	Monotype instantiation
$\sigma = \mathit{Gen}(\tau)$	Generalisation

Solve by Removing Separation

We are allowed to solve separator constraints $\fatsemi$ when:

- There are no context constraints local to the separator
- There are no other separators local to the separator
- There are no monotype equality constraints local to the separator constraint
- There are no generalisation or instantiation constraints local to the separator

These amount to ‘when all constraints in front of the separator are solved’.

A $\mathit{Gen}()$ constraint can be solved under the same conditions – separators create a precongruence defining when we can generalise.

Equality Will Not Be Separated

Separators do not define all solving activity. We can do the following regardless of any separators:

- Solve $=$ constraints and propagate their results
- Solve context constraints (admittedly trivial)
- Propagate information through instantiation constraints – in either direction!

There is no solution until all separators have been removed!

Generalising the Ultimate Answer

Let's build a tree for $\mathrm{let}\ \mathit{id} = \lambda x.x\ \mathrm{in}\ \mathit{id}\ 42$!

$$\begin{array}{l|r}
\{42 : \forall.\mathbb{N}\}, \{\exists\tau a\}\{\exists\sigma\} & \text{Setup\&let} \\
\quad |\{\fatsemi,\ \exists\tau b,\ \sigma = \mathit{Gen}(\tau b)\}\ldots & \text{let (LHS)} \\
\quad \ldots \quad \{\exists\tau p,\ \exists\tau r,\ !x : \forall.\tau p,\ \tau b = \tau p \to \tau r\}\ldots & \text{lambda} \\
\qquad\qquad \ldots \quad \{\exists\sigma r,\ ?x : \sigma r,\ \sigma r \geq \tau r\} & \text{x} \\
\quad |\{!\mathit{id} : \sigma\}\ldots & \text{let (RHS)} \\
\quad \ldots \quad \{\exists\tau f,\ \exists\tau p,\ \tau p \to \tau a = \tau f\} & \text{app} \\
\qquad\qquad\qquad |\{\exists\sigma f,\ ?\mathit{id} : \sigma f,\ \sigma f \geq \tau f\} & \text{id} \\
\qquad\qquad\qquad |\{\exists\sigma p,\ ?42 : \sigma p,\ \sigma p \geq \tau p\} & \text{42}
\end{array}$$

Inference rules left as an exercise for the reader. . .

Drowning in Vars

Variable usage now produces this:

$\{\exists \sigma f,\ ?id : \sigma f,\ \sigma f \geq \tau f\}$

Never let anyone tell you Hindley-Milner is simple again!

- We're going to take a polytype from the context
  - So we need σ to hold it first
- Then we finally get to instantiate it into our return variable τf

Lets Generalise!

The tree fragment for $\mathrm{let}\ id = \ldots\ \mathrm{in}\ \ldots$ looks like this:

$$\begin{array}{l}\{\exists\sigma\}\\ \quad |\{\fatsemi,\ \exists\tau b,\ \sigma = Gen(\tau b)\}\ldots\\ \quad |\{!id : \sigma\}\ldots\end{array}$$

- Create a fresh polytype – it's why we're here
- On the left, sparate, create a monotype to generalise and set up the generalisation
- On the right, bind the LHS with no separator
- Generalisation replaces unconstrained metavariables ($\exists\tau$, not $\exists\tau{=}\tau\prime$) with universally-quantified ones, stopping at the first $\fatsemi$ global of the $Gen()$ constraint.

Regional Concerns

- Separators create a regioning discipline akin to Tofte-Talpin
- Regions branch in (syntactic) space rather than time
- $\exists$ quantifiers are confined to their region until the barrier of separation is solved and removed
- Parallel regions can only communicate via more global regions
- Parallel regions are thus separate as in separation logic

A Moving Existence?

Sometimes $=$ has to unify variables from different regions - one more global than the other with $\fatsemi$s between them.

- Direct case is easy – $\exists \mathit{local} = \mathit{global}$
- Tricky if *local* is part-solved – we have to 'move' variables in it just globally of *global*

- We 'really' create new quantifiers just global of *global*. . .
- Point the more local variables to new more global ones. . .
- And substitute the local variables away

You can abstract out that pattern and even optimise it so long as it has the same semantics! But this approach guarantees soundness.

A Moving Example - 1
The Problem

Let's start with this subtree:

$$\begin{array}{l} \{⨟,\ \exists\tau b,\ \sigma = Gen(\tau b)\}\ldots \\ \ldots\quad \{\exists\tau p,\ \exists\tau r,\ !x : \forall.\tau p,\ \tau b = \tau p \to \tau r\}\ldots \\ \qquad\ldots\quad \{\exists\sigma r,\ ?x : \sigma r,\ \sigma r \geq \tau r\} \end{array}$$

After some solving, we get this situation:

$$\begin{array}{l} \{⨟,\ \exists\tau b,\ \sigma = Gen(\tau b)\}\ldots \\ \ldots\quad \{\exists\tau p,\ x : \forall.\tau p,\ \tau b = \tau p \to \tau p\}\ldots \\ \qquad\ldots\quad \{\exists\sigma r = \forall.\tau p\} \end{array}$$

When we solve $\tau b = \tau p \to \tau p$, τp is too local to appear in τb.

A Moving Example - 2

The Actual Move

Continuing with: $\begin{array}{l}\{\fatsemi,\ \exists\tau b,\ \sigma = \mathit{Gen}(\tau b)\}\ldots \\ \ldots\quad \{\exists\tau p,\ x:\forall.\tau p,\ \tau b = \tau p \to \tau p\}\end{array}$

Add a quantifier for $\tau p'$ immediately global of τb, and $\tau p = \tau p'$:

$$\begin{array}{l}\{\fatsemi,\ \exists\tau p',\ \exists\tau b,\ \sigma = \mathit{Gen}(\tau b)\}\ldots \\ \ldots\quad \{\exists\tau p,\ x:\forall.\tau p,\ \tau b = \tau p \to \tau p,\ \tau p = \tau p'\}\end{array}$$

Solve $\tau p = \tau p'$ and substitute:

$$\begin{array}{l}\{\fatsemi,\ \exists\tau p',\ \exists\tau b,\ \sigma = \mathit{Gen}(\tau b)\}\ldots \\ \ldots\quad \{\exists\tau p{=}\tau p',\ x:\forall.p',\ \tau b = \tau p' \to \tau p'\}\end{array}$$

Now τp's quantifier is redundant. We can remove it and solve $\tau b = \tau p' \to \tau p'$.

A Moving Example - 3

Using The Result

With τp gone: $\{\fatsemi,\ \exists \tau p',\ \exists \tau b,\ \sigma = \mathit{Gen}(\tau b)\} \ldots$
$\ldots \quad \{x : \forall . p',\ \tau b = \tau p' \to \tau p'\}$

Solve for τb:

$\{\fatsemi,\ \exists \tau p',\ \exists \tau b{=}\tau p' \to \tau p',\ \sigma = \mathit{Gen}(\tau b)\} \ldots$
$\ldots \quad \{x : \forall . p'\}$

Zooming out a level, we then substitute τb away:

$\{\exists \sigma\}\{\fatsemi,\ \exists \tau p',\ \sigma = \mathit{Gen}(\tau p' \to \tau p')\}$

We can now generalise then remove the separator, concluding:

$\{\exists \sigma{=}\forall t . t \to t\}$

So What Is It?

So we now have:

- Typing semantics in the same shape as our abstract syntax
  - You can zip the AST to the telescopic tree
  - You can safely put metavariables in the AST
- Trees derivable from Information-Aware typing rules
- Cutting edge tech of 20 years ago available to all!...
  - Contextualising metavariables, conveniently-shaped data
- A general syntax for discussing these structures
  - Even if it's not exactly what we implemented
  - Useful for type-level debugging?

Extra: A General Mistake

Suppose the separator was to the right instead of the left of generalisation:

$$\{\exists \tau p',\ \exists \tau b{=}\tau p' \to \tau p',\ \sigma = \mathit{Gen}(\tau b),\ \fatsemi\}\ldots$$
$$\ldots \quad \{x : \forall . \tau p'\}$$

Substitute out τb:

$$\{\exists \tau p',\ \sigma = \mathit{Gen}(\tau p' \to \tau p'),\ \fatsemi\}\ldots$$
$$\ldots \quad \{x : \forall . \tau p'\}$$

Either we don't remove the separator and we're stuck, or we do and there's no longer a good scope for generalisation - we need to generalise $\tau p'$!