Telescopic Constraint Trees

Philippa Cowderoy

SPLS October 2019

email: *flippa@flippac.org*
twitter: *@flippacpub*

De Bruijn Telescopes – Composing Contexts

- What is a telescope?
  - A telescope looks a lot like a context: $\{x{:}\tau 1, y{:}\tau 2\}$
  - Telescopes can be composed: $\{x{:}\tau 1, y{:}\tau 2\} \circ \{x{:}\tau 3, z{:}\tau 4\} = \{x{:}\tau 1, y{:}\tau, x{:}\tau 3, z{:}\tau 4\}$
  - But I'll compose with commas like so: $\{x{:}\tau 1, y{:}\tau 2\}, \{x{:}\tau 3, z{:}\tau 4\}$
- Is there another useful form of composition?

Terms, Typings and Telescopes

Consider this Simply Typed Lambda Calculus term:

$(\lambda x.x)\ 42$

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

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

Telescopes Grow Branches

We can "trace" the telescopes found in this typing:

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

To find this corresponding tree
(where aligned $|$ shows branches):

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

Branches Go Zip

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
- Using 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

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \; Var \qquad \frac{\Gamma f := \Gamma \,;\, x : \tau p \quad \Gamma f \vdash T : \tau r \quad \tau f = \tau p \to \tau r}{\Gamma \vdash \lambda x.T \;:\; \tau f} \; Lam$$

$$\frac{\Gamma \prec^{\Gamma L}_{\Gamma R} \quad \Gamma L \vdash Tf : \tau f \quad \Gamma R \vdash Tp : \tau p \quad \tau p \to \tau r = \tau f}{\Gamma \vdash Tf \;\, Tp : \tau r} \; App$$

Constraints for the Simply Typed Lambda Calculus

Here's how the IASTLC works now:

$$\begin{array}{rl} \tau = \tau & \text{Type equality} \\ \tau \prec^{\tau l}_{\tau r} & \text{Type duplication} \\ & \\ x : \tau \in \Gamma & \text{Binding in context} \\ \Gamma' := \Gamma ; x : \tau & \text{Context extension} \\ \Gamma \prec^{\Gamma L}_{\Gamma R} & \text{Context duplication} \end{array}$$

These need turning into something more telescopic

Context Constraints In, um, Context?

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

$$
\begin{array}{rl}
x : \tau \in \Gamma & \text{Binding in context} \\
\Gamma' := \Gamma \,;\, x : \tau & \text{Context extension}
\end{array}
$$

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 current 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:

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

(Decon)Structural Laws

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

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

- Moving $\exists$ is easy for unsolved variables
- Moving $=$ is often valid...
- Still needs to respect metavariable scope
  - Especially during solving

Context Solving

- We'll do this next because constraint contexts are *situated*
- Their exact location in the data structure matters

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

$\Rightarrow$

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

Constraint Lifting

Now pull all the remaining constraints towards the head/global end of the tree:

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

- 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 so far!

∃s Are Good

A great philosopher once wrote:

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

$\exists$s Are Good

A great philosopher once wrote:

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

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!

∃s Are Good

A great philosopher once wrote:

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

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: ∃ can give type variables/metavariables a scope

Back to Front

- The Gen rule talks about “free in the context [at this point]”
- This is another way to talk about “solved and unconstrained in the context local to this point”.

- Talking about the ”context before/at this point” (Γ) is standard
- Typing rules had no way to talk about the local contexts involved in typing subterms
- What about us?

Telescopes and Generalisation

Normally I would write this:

- $\sigma = \mathit{Gen}(\tau)$ for generalisation
- $\sigma \geq \tau$ for instantiation
- Milner's rules as their satisfaction predicates

That doesn't fix anything though.

Let's look at the scope in $\sigma = \mathit{Gen}(\tau)$ carefully.
Do σ and τ belong in the same scope?

Reinvention

$\{\exists\sigma, \exists\tau, \sigma = \mathit{Gen}(\tau)\}$ for $\sigma = \mathit{Gen}(\tau)$ is a little unsatisfying.

Instead, $\sigma = \mathit{Gen}(\tau)$ can become:

$\{\exists\sigma, <\mathit{Gen}>, \exists\tau, </\mathit{Gen}\ (\sigma \geq \tau)>\}$

- Splits generalisation into two constraints
- Delimits a scope for existential quantifiers

(People who got here another way write $<\mathit{Gen}>$ as $⨟$!)

New Structure

- We may move existentials global of a $</Gen>$:
  $\{</Gen>, \exists\tau\} \Rightarrow \{\exists\tau, </Gen>\}$
- Nothing moves past a $<Gen>$ – it separates until solved
- $</Gen>$ solves when all its local existentials are known and there are no local constraints
  - Unconstrained existentials need moving 'inside' it
- $<Gen>$ solves when there is no local constraint or unknown existential
  - Disappears once the local $</Gen>$ is solved
  - Acts as a separator

Constraints That Make You Go ‘H-M’

We’re going to skip the actual typing rules today

- Context will now bind *n*-ary polytypes, $\sigma = \forall \bar{a}.\tau$
- Instantiation on variable use and generalisation at let

Constraints we’ll use include:

$\exists\sigma$	Polytype binding
$\exists\sigma{=}\forall\bar{a}.\tau$	(Part-)Solved polytype
$\sigma \geq \tau$	Instantiation
$\sigma = \mathit{Gen}(\tau)$	Generalisation – typing rules
$<\!\mathit{Gen}\!>$	Opening generalisation – tree
$<\!/\mathit{Gen}\ (\sigma \geq \tau)\!>$	Closing generalisation – tree

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 $\text{let } id = \lambda x.x \text{ in } id\ 42$!

$\{42 : \forall.\mathbb{N}\}, \{\exists \tau a\}\{\exists \sigma\}$	Setup&let
$\quad \vert\{<Gen>,\ \exists \tau b,\ </Gen\ (\sigma \geq \tau b)>\} \ldots$	let (LHS)
$\quad \ldots \quad \{\exists \tau p,\ \exists \tau r,\ !x : \forall.\tau p,\ \tau b = \tau p \to \tau r\} \ldots$	lambda
$\qquad \ldots \quad \{\exists \sigma r,\ ?x : \sigma r,\ \sigma r \geq \tau r\}$	x
$\quad \vert\{!id : \sigma\} \ldots$	let (RHS)
$\quad \ldots \quad \{\exists \tau f,\ \exists \tau p,\ \tau p \to \tau a = \tau f\}$	app
$\qquad \vert\{\exists \sigma f,\ ?id : \sigma f,\ \sigma f \geq \tau f\}$	id
$\qquad \vert\{\exists \sigma p,\ ?42 : \sigma p,\ \sigma p \geq \tau p\}$	42

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 $\text{let } \mathit{id} = \ldots \text{ in } \ldots$ looks like this:

$$
\begin{aligned}
&\{\exists\sigma\}\\
&\quad|\{<\!\mathit{Gen}\!>,\ \exists\tau b,\ <\!/\,\mathit{Gen}\ (\sigma \geq \tau b)\!>\}\ldots\\
&\quad|\{!\mathit{id} : \sigma\}\ldots
\end{aligned}
$$

- Create a fresh polytype – it's why we're here
- On the left, separate, 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 $<\!\mathit{Gen}\!>$

Regional Concerns

- Separators like $<$*Gen*$>$ 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 $<$*Gen*$>$s between them.

- Direct case is easy – $\exists local = global$
- 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 optimise it so long as it has the same semantics! This approach guarantees soundness.

Wins

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?