Telescopic Constraint Trees

Philippa Cowderoy

Idris Developers Meeting 2021

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

Overview

- What is a Telescopic Constraint Tree? (*aka TCT*)
- Example: Rules for Building TCTs for STLC
- Generalisation with TCTs
- TCTs for Substructural Systems and QTT
- Odds and Ends

What is a TCT? – Big Picture

- A complete representation of a typechecking problem
  - Maybe in progress or even finished
- A scope-aware constraint store
- A way to talk about checker behaviour
- A way to implement a checker

What is a TCT? – Not For Today

- A recurring idea I put a name to
- Derivable from ‘Information Aware’ typing rules
- A tool for type-level debugging?
- An operating system?...

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:
  $\{x{:}\tau 1,\ y{:}\tau 2\} \circ \{x{:}\tau 3,\ z{:}\tau 4\} = \{x{:}\tau 1,\ y{:}\tau 2,\ x{:}\tau 3,\ z{:}\tau 4\}$
- But I'll compose with commas like so:
  $\{x{:}\tau 1,\ y{:}\tau 2\}, \{x{:}\tau 3,\ z{:}\tau 4\}$

De Bruijn Telescopes – Tracing AST Paths

- As we trace a path through a term, we accumulate more context
- We can see this as composing telescopes, one per AST node
- Context $\approx$ sequence of telescopes

Telescopic Trees

- We have a tree, the AST
- We can already trace paths through it...
- ...But we could build a second tree and cover all paths!
- A *Telescopic Tree*

A Telescopic Tree

Example term: $(\lambda x.\ x)$ "Hello, World!"

$\{\},$	Initial context
$\quad \{\}$	Application
$\quad\quad \vert\{\},$	$\lambda x.$
$\quad\quad\quad \{x : \text{String}\}$	x
$\quad\quad \vert\{\}$	"Hello, World!"

A More Compact Telescopic Tree

$$
\begin{array}{l}
\{\}, \\
\qquad \{\} \\
\qquad\qquad |\{\}, \\
\qquad\qquad\qquad \{x : \text{String}\} \\
\qquad\qquad |\{\}
\end{array}
$$

Initial context
Application
$\lambda x.$
x
"Hello, World!"

(Overly?) compact version: $\{\}, \{\} \quad |\{\}, \{x : \text{String}\}$
$|\{\}$

So Far So Cute?

- Simple enough so far – take existing idea, add branches
- Not informative enough
  - Given $\begin{array}{ll} \{\}, \{\} & | \{\}, \{x : \mathrm{String}\} \\ & | \{\} \end{array}$, where does String even come from?...
- No problem declared, no computational content!

Telescopic *Constraint* Trees

- We want to talk about how context and connectives interact
- Contents of typing rules reflected in tree!

- We want to see the types of subterms
  - Perhaps not emphasised as such, but definitely present

- Constraint solving problems can do all this!

Telescopic Constraints

$\exists\tau$	Metavariable binding
$\exists\tau{=}\tau$	Known metavariable binding
$\tau = \tau$	Type equality
$?x : \tau$	Query (object) variable bindings

- $=$ covers simple metavariable assignments and unification
- $?x : \tau$ is *situated* in the TCT – it must respect scope

An Example TCT

Example term: $(\lambda x.\ x)$ "Hello, World!"

$\{\},$	Initial context
$\{\exists \tau q\},$	Query variable
$\{\exists \tau f,\ \exists \tau p,\ \tau p \to \tau q = \tau f\},$	Application
$\vert\{\exists \tau p',\ \exists \tau r,\ \tau f = \tau p' \to \tau r\},$	$\lambda x.$
$\{x : \tau p',\ ?x : \tau r\}$	x
$\vert\{\tau p = \text{String}\}$	"Hello, World!"

An Example TCT – Some Solving

Solve the constraint $?x : \tau r$ and propagate:

$$
\begin{array}{l|r}
\{\}, & \text{Initial context} \\
\quad \{\exists \tau q\}, & \text{Query variable} \\
\qquad \{\exists \tau f,\ \exists \tau p,\ \tau p \to \tau q = \tau f\}, & \text{Application} \\
\qquad\quad |\{\exists \tau p',\ \exists \tau r{=}\tau p',\ \tau f = \tau p' \to \tau p'\}, & \lambda x. \\
\qquad\qquad \{x : \tau p'\} & x \\
\qquad\quad |\{\tau p = \mathrm{String}\} & \text{“Hello, World!”}
\end{array}
$$

An Example TCT – More Solving

Eliminate τr (no remaining occurrences due to propagation)
Solve $\tau f = \tau p' \rightarrow \tau p'$ and propagate:

$\{\},$	Initial context
$\{\exists \tau q\},$	Query variable
$\{\exists \tau p', \; \exists \tau f {=} \tau p' \rightarrow \tau p', \; \exists \tau p, \ldots$	Application
$\ldots \tau p \rightarrow \tau q = \tau p' \rightarrow \tau p'\},$	
$\|\{\},$	$\lambda x.$
$\{x : \tau p'\}$	x
$\|\{\tau p = \mathrm{String}\}$	"Hello, World!"

An Example TCT – Yet More Solving

Eliminate τf

Solve $\tau p = \mathrm{String}$ and propagate:

$\{\},$	Initial context
$\{\exists \tau q\},$	Query variable
$\{\exists \tau p',\ \exists \tau p {=} \mathrm{String}, \ldots$	Application
$\ldots \mathrm{String} \rightarrow \tau q = \tau p' \rightarrow \tau p'\},$	
$\vert\{\},$	$\lambda x.$
$\{x : \tau p'\}$	x
$\vert\{\}$	"Hello, World!"

An Example TCT – Solving = Slowly (1)

Eliminate τp

Simplify $\mathrm{String} \rightarrow \tau q = \tau p' \rightarrow \tau p'$ one step:

$\{\},$	Initial context
$\quad \{\exists \tau q\},$	Query variable
$\qquad \{\exists \tau p',\ \mathrm{String} = \tau p',\ \tau q = \tau p'\},$	Application
$\qquad\quad \vert\{\},$	$\lambda x.$
$\qquad\qquad \{x : \tau p'\}$	x
$\qquad\quad \vert\{\}$	"Hello, World!"

An Example TCT – Solving = Slowly (2)

Solve $\mathrm{String} = \tau p'$ and $\tau q = \tau p'$, propagate:
$\tau p'$ can now be eliminated – and we have solved for τq!

$\{\},$	Initial context
$\quad\{\exists \tau p' {=} \mathrm{String}, \exists \tau q {=} \mathrm{String}\},$	Query variable
$\qquad\{\},$	Application
$\qquad\quad|\{\},$	$\lambda x.$
$\qquad\qquad\{x : \mathrm{String}\}$	x
$\qquad\quad|\{\}$	"Hello, World!"

An Example TCT – Supsiciously Familiar Solution!

$\{\},$	Initial context
$\quad \{\exists \tau q = \mathrm{String}\},$	Query variable
$\qquad \{\},$	Application
$\qquad\quad |\{\},$	$\lambda x.$
$\qquad\qquad \{x : \mathrm{String}\}$	x
$\qquad\quad |\{\}$	"Hello, World!"

$$\{\}, \{\exists \tau q = \mathrm{String}\} \; |\{\}, \{x : \mathrm{String}\}$$
$$|\{\}$$

A Note on Showing Working

At each step we eliminated bindings and solved constraints.

This is much easier for us to follow, but destroys any audit trail!

Alternatives:

- Mark constraints solved
- Mark metavariables unused
- Mark solved metavariables with the constraint that found them
- Mark generated constraints with their parent constraint

What Does a Telescopic Constraint Tree *Do*?

What an ordinary typechecker does in time

Telescopic constraint trees do in space

Building TCTs

Next up:

- Rules for building TCTs like the example

- Based on an ‘Information Aware’ presentation of STLC
  - Constraint-based presentation
  - One ‘unusual’ constraint
  - Structural laws reified as context constraints

Constraints for the Simply Typed Lambda Calculus

$$
\begin{array}{r|l}
\tau = \tau & \text{Type equality} \\
x : \tau \in \Gamma & \text{Binding in context} \\
\Gamma := \Gamma\, ;\, x : \tau & \text{Context extension} \\
\Gamma \prec \begin{smallmatrix}\Gamma \\ \Gamma\end{smallmatrix},\ \Gamma \prec \{\overline{\Gamma}\} & \text{Context duplication}
\end{array}
$$

- Context constraints are ‘opinionated’ but normal enough
- Duplication (possibly also ‘merge’ or even ‘split’) is unusual

One Extra Quirk for TCTs

We could skip this at this point, but...

- TCTs could use another context constraint!

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

- We'll exploit this later

Building Trees

We build a TCT by traversing the AST

A 'TCT semantics': $[\![T]\!]\ \tau$

- Translate T into a TCT
- Have τ become the result type

Retaining all information from the typing rules!

Starting the Build

Suppose we want to synthesise a type:
$\Gamma^+ \vdash T^+ : \tau^-$

We use this rule to build the corresponding tree:

$$\Gamma^+, \{\exists\tau^-\}, \llbracket T^+ \rrbracket\ \tau^+ \qquad (\mathit{Start})$$

τ acts as a query variable

Building Var

This typing rule:

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \; Var$$

Becomes this TCT rule:

$$[\![x^+]\!]\ \tau^- \;= \{?x^- : \tau^+\} \qquad (Var)$$

Building Lam

$$\dfrac{\Gamma f := \Gamma \,;\, x : \tau p^{+} \qquad \Gamma f \vdash T : \tau r^{+} \qquad \tau f = \tau p^{-} \rightarrow \tau r^{-}}{\Gamma \vdash \lambda x.T \;:\; \tau f} \; Lam$$

$$[\![\lambda x.T]\!]\ \tau f \;= \\ \{\exists \tau p,\ \exists \tau r,\ \tau f = \tau p^{-} \rightarrow \tau r^{-},\ !x : \tau p^{+}\}, [\![T]\!]\ \tau r^{+} \qquad (Lam)$$

Building App

$$\frac{\Gamma \prec \{\Gamma f,\ \Gamma p\} \qquad \Gamma f \vdash Tf : \tau f \qquad \Gamma p \vdash Tp : \tau p \qquad \tau p \to \tau r = \tau f}{\Gamma \vdash Tf\ Tp : \tau r}\ App$$

$$\begin{aligned}
[\![Tf \quad Tp]\!]\ \tau r \ = & \\
\{\exists \tau f,\ \exists \tau p,\ \tau p \to \tau r = \tau f\} &\mid [\![Tf]\!]\ \tau f \\
&\mid [\![Tp]\!]\ \tau p\ \ (App)
\end{aligned}$$

Deriving TCT Rules

- The rules for TCTs are mechanically derived!

- This gives us (informal) completeness of TCTs
  - Complete regarding type system rules
  - Anything a checker (justifiably) does, TCTs can do
  - TCTs can be used to discuss all possible checkers!

  - More ‘operational’ details with no abstraction lost

‘Advanced’ Topics

We’ve already covered:

- How TCTs work
- How to build them for a specific system
- How generic TCTs are

Now for:

- Generalisation
  - Constraining when constraints are solved
- TCTs for substructural systems and QTT

Generalisation Is Hard?

- Generalisation rules are normally written in terms of the context where generalisation happens:

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

- This is confusing (and ultimately error-prone)...
- ...because we really want to generalise over *unconstrained* variables
  - After all local constraints have been solved!

Generalisation 'In Context' Is Easy!

We try to could use a constraint like $\sigma = \mathit{Gen}_{\Gamma}(\tau)$, but that hurts

From 'Type Inference In Context' by Gundry, McBride and McKinna:

- Two constraints – <*Gen*> and </*Gen* $(\tau \geq \tau)$>
- A matched pair delimiting unbound metavariables
- <*Gen*> prevents making metavariables too global without cause
- </*Gen* $(\tau \geq \tau)$> fires when it has no unsolved local constraints
    - <*Gen*> can then remove itself

Generalisation 'In Context' Example

Example in Hindley-Milner: let $id = \lambda x.x$ in id "Hello, World!"

$\{\}, \{\exists \tau q\}\{\exists \sigma\}$	Setup&let
$\quad \mid \{<Gen>,\ \exists \tau b,\ </Gen\ (\sigma \geq \tau b)>\},$	let (LHS)
$\quad\quad \{\exists \tau p,\ \exists \tau r,\ !x : \forall.\tau p,\ \tau b = \tau p \to \tau r\},$	lambda
$\quad\quad\quad \{\exists \sigma r,\ ?x : \sigma r,\ \sigma r \geq \tau r\}$	x
$\quad \mid \{!id : \sigma\},$	let (RHS)
$\quad\quad \{\exists \tau f,\ \exists \tau p,\ \tau p \to \tau q = \tau f\}$	app
$\quad\quad\quad \mid \{\exists \sigma f,\ ?id : \sigma f,\ \sigma f \geq \tau f\}$	id
$\quad\quad\quad \mid \{\tau p = \mathrm{String}\}$	"Hello, World!"

Generalisation ‘In Context’ – Historical Notes

- Originally $\fatsemi$ instead of $<\mathit{Gen}>$ – the HTML-like notation is my fault
- $\fatsemi$ can be seen as ‘residue’ from dissecting an abstract machine
  - It marks where the machine went down the LHS of a let
- TCTs as a strategy are old! I just figured out how to derive them

Generalisation 'In Context' – Technical Notes

- I normally write typing rules with 'linear' variables
- Duplication constraints and TCT branches mark separation
  - As in separation logic!
- Generalisation delimits 'regions' (think Tofte-Talpin)
- As much sequencing of constraint solving as we need, no more

'Duplication' for Substructural Systems

What satisfies $\Gamma \prec \{\Gamma l,\ \Gamma r\}$?

When $\Gamma \overset{struct}{=} \Gamma l, \Gamma r$

This works for any combination of the usual structural rules

'Duplication' for Linear Systems – Two Fragments

'Additive' Duplication	'Multiplicative' Duplication
$\Gamma \dashv\!\!\langle^{\Gamma l}_{\Gamma r}$ or $\Gamma \dashv\!\!\langle\{\overline{\Gamma}\}$	$\times\!\!\langle^{\Gamma l}_{\Gamma r}$ or $\times\!\!\langle\{\overline{\Gamma}\}$
Split Γ between Γl and Γr	Duplicate Γ into Γl and Γr
$\Gamma \overset{struct}{=} \Gamma l, \Gamma r$	$\Gamma = \Gamma l = \Gamma r$

In systems with all the structural laws, these coincide

QTT Duplication

How could I copy 0 times into one context and 1 into another?

Perhaps:

$\Gamma \rtimes\!\langle^{0}_{1} \begin{smallmatrix} \Gamma \mathit{type} \\ \Gamma \mathit{term} \end{smallmatrix}$ or $\Gamma \rtimes\!\langle \{0 \cdot \Gamma \mathit{type}, 1 \cdot \Gamma \mathit{term}\}$

Multiply Γ*type*'s contents by 0
Multiply Γ*term*'s contents by 1
(using the resource rig's multiplication!)

Wild Speculation! Also previous material...

- Interactive editing based on TCTs?
- 'Information Awareness' is about information flow and preservation
  - Could we have reversible TCT-based elaboration etc?

- MSFP2020 Extended Abstract: https://msfp-workshop.github.io/msfp2020/cowderoy.pdf
- MSFP2020 Slides: https://msfp-workshop.github.io/msfp2020/slides/cowderoy.pdf
- MSFP2020 Talk – https://www.youtube.com/watch?v=JzfdjMgEKzs