Telescopic Constraint Trees

Philippa Cowderoy

Idris Developers Meeting 2021

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

- 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

- 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

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

What is a telescope?

- A telescope looks a lot like a context: {x:\approx1, y:\approx2}
- 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:\appr1, y:\appr2}, {x:\appr3, z:\appr4}

• As we trace a path through a term, we accumulate more context

• We can see this as composing telescopes, one per AST node

 ${\scriptstyle \bullet}$ Context \approx sequence of telescopes

• 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

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



A More Compact Telescopic Tree

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

- Simple enough so far take existing idea, add branches
- Not informative enough
 Given {}, {} |{}, {x : String} |{}, {x : String} , |{} where does String even come from?...

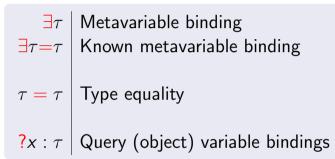
• No problem declared, no computational content!

- 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



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

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

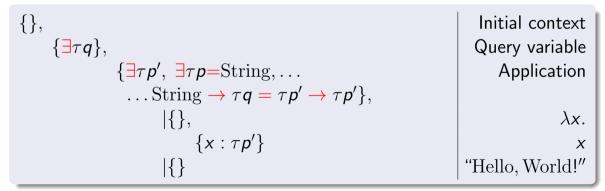
э

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

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

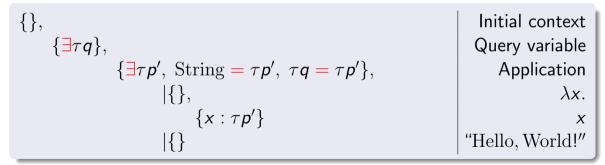
An Example TCT – Yet More Solving

Eliminate τf Solve $\tau p = \text{String}$ and propagate:



э

Eliminate τp Simplify String $\rightarrow \tau q = \tau p' \rightarrow \tau p'$ one step:



Philippa Cowderov

э

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

An Example TCT – Supsiciously Familiar Solution!

э

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 an ordinary typechecker does in time

Telescopic constraint trees do in space

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

$$\begin{array}{c|c} \tau = \tau & \text{Type equality} \\ x : \tau \in \Gamma & \text{Binding in context} \\ \Gamma := \Gamma ; x : \tau & \text{Context extension} \\ \Gamma - \!\! \langle \Gamma \!\! \langle \Gamma \!\! \rangle & \text{Context duplication} \end{array}$$

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

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

• TCTs could use another context constraint!

'Information Aware'	ТСТ	Description
<i>x</i> : <i>τ</i> ∈ Γ		Query/Ask for binding [here]
$\Gamma' := \Gamma; x : \tau$	x: au	Generate/Tell about binding [here]

• We'll exploit this later

We build a TCT by traversing the AST

- A 'TCT semantics': $\llbracket T \rrbracket \tau$
 - Translate T into a TCT
 - \bullet Have τ become the result type

Retaining all information from the typing rules!

```
Suppose we want to synthesise a type:

\Gamma^+ \vdash T^+ : \tau^-
```

We use this rule to build the corresponding tree:

$$\Gamma^+, \{ \exists \tau^- \}, \ \llbracket T^+ \rrbracket \ \tau^+$$

au acts as a query variable

This typing rule: $x : \tau \in \Gamma$

 $\Gamma \vdash x : \tau$ Var

Becomes this TCT rule:

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

3

・ 同 ト ・ ヨ ト ・ ヨ ト

Building Lam

 $\Gamma f := \Gamma ; x : \tau p^+$ $\Gamma f \vdash T : \tau r^+$ $\tau f = \tau p^- \to \tau r^-$

 $\Gamma \vdash \lambda x.T$: τf Lam

$$\begin{bmatrix} \lambda x. T \end{bmatrix} \tau f = \\ \{ \exists \tau p, \ \exists \tau r, \ \tau f = \tau p^- \to \tau r^-, \ !x : \tau p^+ \}, \llbracket T \rrbracket \tau r^+$$
(Lam)

3

く 周 ト く ヨ ト く ヨ ト

 $\begin{array}{l} \Gamma - \langle \{ \Gamma f, \ \Gamma p \} \\ \Gamma f \vdash Tf : \tau f \qquad \Gamma p \vdash Tp : \tau p \\ \tau p \rightarrow \tau r = \tau f \end{array}$

 $\Gamma \vdash Tf \ Tp : \tau r \qquad App$

$$\begin{bmatrix} Tf & Tp \end{bmatrix} \tau r = \\ \{ \exists \tau f, \exists \tau p, \tau p \to \tau r = \tau f \} \mid \llbracket Tf \rrbracket \tau f \\ \mid \llbracket Tp \rrbracket \tau p \ (App) \end{bmatrix}$$

イロト 不得 トイヨト イヨト 二日

• 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

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

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

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

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

From 'Type Inference In Context' by Gundry, McBride and McKinna: • Two constraints – $\langle Gen \rangle$ and $\langle /Gen (\tau \ge \tau) \rangle$

- A matched pair delimiting unbound metavariables
- < Gen> prevents making metavariables too global without cause
- </Gen (τ ≥ τ)> fires when it has no unsolved local constraints
 <Gen> can then remove itself

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

$$\{ \}, \{ \exists \tau q \} \{ \exists \sigma \}$$
 Setup&let

$$\{ \{ < Gen >, \exists \tau b, < / Gen (\sigma \ge \tau b) > \},$$
 let (LHS)

$$\{ \exists \tau p, \exists \tau r, !x : \forall .\tau p, \tau b = \tau p \rightarrow \tau r \},$$
 lambda

$$\{ \exists \sigma r, ?x : \sigma r, \sigma r \ge \tau r \}$$
 k

$$\{ !id : \sigma \},$$
 let (RHS)

$$\{ \exists \tau f, \exists \tau p, \tau p \rightarrow \tau q = \tau f \}$$
 let (RHS)

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

$$\{ \tau p = String \}$$
 "Hello, World!"

• Originally § instead of *<Gen>* – the HTML-like notation is my fault

§ 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

- 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

What satisfies $\Gamma - \langle \{\Gamma I, \Gamma r\} \rangle$?

When $\Gamma \stackrel{struct}{=} \Gamma I, \Gamma r$

This works for any combination of the usual structural rules

'Additive' Duplication	'Multiplicative' Duplication	
$\Gamma - \langle \Gamma_r $ or $\Gamma - \langle \{\overline{\Gamma}\}$	$\varkappa_{\Gamma_r}^{\Gamma_l}$ or $\varkappa_{\overline{\Gamma}}$	
Split Γ between Γ / and Γ r	Duplicate Γ into Γ / and Γ r	
$\Gamma \stackrel{struct}{=} \Gamma I, \Gamma r$	$\Gamma = \Gamma I = \Gamma r$	

In systems with all the structural laws, these coincide

Phili		

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

Perhaps:

$$\Gamma \not\prec_{1 \ \Gamma term}^{0 \ \Gamma type} \text{ or } \Gamma \not\prec_{1}^{\prime} \{0 \cdot \Gamma type, 1 \cdot \Gamma term\}$$

Multiply $\Gamma type$'s contents by 0 Multiply $\Gamma term$'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