Information Aware Type Systems and Telescopic Constraint Trees

Philippa Cowderoy

MSFP 2020

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

What I'm Talking About

- Motivating 'information awareness'
- Tools for information awareness
- Information Aware Type Systems

What I'm Talking About

- Motivating 'information awareness'
- Tools for information awareness
- Information Aware Type Systems
- Telescopic Constraint Trees

Information Awareness?

Some people find *AppEq* easier to read than *AppTrad*. Why?

$$\frac{\Gamma \vdash Tp : \tau p \qquad \Gamma \vdash Tf : \tau p \rightarrow \tau r}{\Gamma \vdash Tf\ Tp : \tau r}\ AppTrad \qquad \frac{\Gamma \vdash Tp : \tau p \qquad \Gamma \vdash Tf : \tau f \qquad \tau f = \tau p \rightarrow \tau r}{\Gamma \vdash Tf\ Tp : \tau r}\ AppEq$$

- *AppEq* shows us explicitly the information we infer
- e.g. *Tf* is a function that can take *Tp* as a parameter

Information Aware Variable Use?

What does this τ say?

Information Aware Variable Use?

What does this τ say?

Using schematic variables can mean different things:

- Does it use known information?
- Is it a pattern?
- Is it part of a non-linear pattern?
- Can it be reduced first? (Should it?)

Part Way There

- We want an ‘information aware’ approach
- Let us be explicit about this!

Part Way There

- We want an 'information aware' approach
- Let us be explicit about this!

- Constraints can help us do better...
- ...But they can't stop us doing worse

What I'm Talking About Now

- Motivating 'information awareness' – DONE
- Tools for information awareness
  - Information Effects
  - Constraints
- Information Aware Type Systems
- Telescopic Constraint Trees

Information Effects

- From *Information Effects* by James & Sabry
- Mostly-reversible programming

Information Effects

- From *Information Effects* by James & Sabry
- Mostly-reversible programming
- Seen via *isomorphic* programming

Information Effects

- From *Information Effects* by James & Sabry
- Mostly-reversible programming
- Seen via *isomorphic* programming

- I can still use the idea in just one direction!

Information Effects Are...

Breaches of conservation of information:

- When information is created
- When information is destroyed

Information Effects Are...

Breaches of conservation of information:

- When information is created
- When information is destroyed
- When information is duplicated

Information Effects Are...

Breaches of conservation of information:

- When information is created
- When information is destroyed
- When information is duplicated

- *Inference* creates new information
- *Implied* by the information we have

Contraints as Information

Syntax

Static Semantics

Dynamic Semantics

Contraints as Information

Syntax

- Just what we wrote
- Information as in bits
- Bits are a bad unit for precise accounting!

Static Semantics

Dynamic Semantics

Contraints as Information

Syntax

Static Semantics

- What holds if the constraint is satisfied?
- *Satisfaction predicate*
- What shapes of problem are solveable?

Dynamic Semantics

Contraints as Information

Syntax

Static Semantics

Dynamic Semantics

- Part of a *constraint problem*
- Part of the *process* of solving that problem

Contraints as Information

Syntax

Static Semantics

Dynamic Semantics

- Part of the *process* of solving a problem
- A process that can block on unknown information
- . . . Or provide information
- Unknowns represented by
  solver variables or *metavariables*

Contraints as Information

Syntax

Static Semantics

Dynamic Semantics

Insight

Constraints are a unit of information!

What I'm Talking About Now

- Motivating 'information awareness' – DONE
- Tools for information awareness – DONE
- Information Aware Type Systems
  - Aim
  - Methods
  - Example – Simply Typed Lambda Calculus
  - Modes
- Telescopic Constraint Trees

Information Aware Type Systems – Aim

- Clear introduction and elimination of information
- Clear data flow
- Clear choices about flow
- In design and implementation

- Linear logic variables: one +ve source, one -ve sink
- Constraints
  - Generation is an information effect
  - Keep dataflow general
  - Flexible abstraction
- Explicit duplication

Constraints for the Simply Typed Lambda Calculus

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

- Convention: write $L = R$ as if 'assigning' to L
- Context constraints $\Rightarrow$ structural rules

Information Aware Simply Typed λ-Calculus – *Var*

$$\tau = \tau \quad \text{Type equality}$$

$$x : \tau \in \Gamma \quad \text{Binding in context}$$

$$\Gamma' := \Gamma \,;\, x : \tau \quad \text{Context extension}$$

$$\Gamma \prec^{\Gamma}_{\Gamma} \quad \text{Context duplication}$$

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

Information Aware Simply Typed λ-Calculus – *Lam*

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

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

Information Aware Simply Typed λ-Calculus – *App*

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

$$
\frac{\Gamma \prec^{\Gamma f}_{\Gamma p} \qquad \Gamma f \vdash Tf : \tau f \qquad \Gamma p \vdash Tp : \tau p \qquad \tau p \rightarrow \tau r = \tau f}{\Gamma \vdash Tf \ Tp : \tau r} \; App
$$

Modes

- Which way is data flowing?
- When do we know enough to solve a constraint?

Modes

- Which way is data flowing?
- When do we know enough to solve a constraint?

- Data flows +ve to -ve
- Constraints can ‘fire’ when -ve variables known

Different Modes of a Type System

Mode	Unidirectional	Bidirectional
$\Gamma^{+} \vdash T^{+} : \tau^{+}$	Type checking	Checking
$\Gamma^{+} \vdash T^{+} : \tau^{-}$		Synthesis
$\Gamma^{-} \vdash T^{+} : \tau^{+}$	Free variable types	Checked type
$\Gamma^{-} \vdash T^{+} : \tau^{-}$		Synthesised type
$\Gamma^{+} \vdash T^{-} : \tau^{+}$	Proof search	
	Program synthesis	

Each mode can have its own implementation ('procedure')

$\rightarrow$ - The Other Information Effect

- The function arrow $\rightarrow$ doesn't appear in source
- It does appear in our types

- Information we *infer* from or *create* about terms

Moding $\rightarrow$

I assign two different modes to $\rightarrow$:

- Based on how the solver handles $=$ constraints
- LHS of $=$ is being 'assigned to' in some form
- $+$ve construction vs -ve pattern-matching

Information Aware Simply Typed λ-Calculus (moded)
Var

Mode: $\Gamma^+ \vdash T^+ : \tau^-$ (Synthesis or 'typechecking')

$$\frac{x^- : \tau^+ \in \Gamma^-}{\Gamma^+ \vdash x^+ : \tau^- \quad \mathit{Var}}$$

Information Aware Simply Typed λ-Calculus (moded) *Lam*

Mode: $\Gamma^+ \vdash T^+ : \tau^-$ (Synthesis or 'typechecking')

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

Information Aware Simply Typed λ-Calculus (moded)
App

Mode: $\Gamma^{+} \vdash T^{+} : \tau^{-}$ (Synthesis or 'typechecking')

$$\Gamma^{-} \prec \begin{matrix}\Gamma f^{+} \\ \Gamma p^{+}\end{matrix}$$

$$\Gamma f^{-} \vdash Tf^{-} : \tau f^{+} \qquad \Gamma p^{-} \vdash Tp^{-} : \tau p^{+}$$

$$\tau p^{-} \rightarrow^{-} \tau r^{+} = \tau f^{-}$$

$$\frac{}{\Gamma^{+} \vdash Tf^{+}\ Tp^{+} : \tau r^{-}} \quad \textit{App}$$

Information Aware Simply Typed λ-Calculus

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

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

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

What I'm Talking About Now

- Motivating 'information awareness' – DONE
- Tools for information awareness – DONE
- Information Aware Type Systems – DONE
- Telescopic Constraint Trees
  - What they are and do
  - How to build them
  - Things to do with them!

What's a Telescopic Constraint Tree?

A Telescopic Constraint Tree (or TCT) is:

- A Tree – a refined AST
- Containing Constraints

- Telescopic

What's a Telescopic Constraint Tree?

A Telescopic Constraint Tree (or TCT) is:

- A Tree – a refined AST
- Containing Constraints
  - Also metavariable binders
- Telescopic

What's a Telescopic Constraint Tree?

A Telescopic Constraint Tree (or TCT) is:

- A Tree – a refined AST
- Containing Constraints
  - Also metavariable binders
- Telescopic
  - Built from composable contexts

What Else is a Telescopic Constraint Tree?

- A typechecking problem in progress
- Generic
- Derivable from Information Aware Type Systems
- A scope-aware constraint store

What Else is a Telescopic Constraint Tree?

- A typechecking problem in progress
- Generic
- Derivable from Information Aware Type Systems
- A scope-aware constraint store
  - Scoping not needed for STLC
  - Useful for HM, dependent types and more!

What Does a Telescopic Constraint Tree *Do*?

What an ordinary typechecker does

What Does a Telescopic Constraint Tree *Do*?

What an ordinary typechecker does in time

Telescopic constraint trees do in space

Context Constraints In, um, Context?

We 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!

Situated Constraints

Old	New
$x : \tau \in \Gamma$	$?x : \tau$
$\Gamma' := \Gamma\ ;\ x : \tau$	$!x : \tau$

The new constraints are *situated*.
Their meaning depends on their position in the telescope:

	Description
$?x : \tau$	Query/Ask for current binding [here]
$!x : \tau$	Generate/Tell about binding [here]

Binding and Equality

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

We can ask "is this solved?"

The Easy One

Context duplications are tree branches!

Building Trees

We build a TCT by traversing the AST

A 'TCT semantics': $\llbracket T \rrbracket\ \tau$

- 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 this tree:

$$\Gamma^+, \{\exists\tau^-\}, \; [\![T^+]\!] \; \tau^+ \qquad (Start)$$

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

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

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

Building App

$$\Gamma \prec^{\Gamma f}_{\Gamma p}$$

$$\Gamma f \vdash Tf : \tau f \qquad \Gamma p \vdash Tp : \tau p$$

$$\tau p \rightarrow \tau r = \tau f$$

$$\frac{}{\Gamma \vdash Tf \; Tp : \tau r} \quad App$$

$$\begin{aligned} [\![Tf \quad Tp]\!] \; \tau r \; = \\ \{\exists \tau f, \; \exists \tau p, \; \tau p \rightarrow \tau r = \tau f\} \; | \; [\![Tf]\!] \; \tau f \\ | \; [\![Tp]\!] \; \tau p \; (App) \end{aligned}$$

What can I do with them?

We can, of course, implement typecheckers.

We can also instrument those checkers:

- Changes in the tree show the process, including scope
- Those changes preserve the AST structure
  - Mostly – labels are useful extra info
- UI for type level debugging?

What else can I do with them?

We can usually fuse TCTs away:

Assuming we have:

- A *greedySolver* for the TCT's constraints
- A *treeGenerator*, implementing $[\![T]\!] \; \tau$

We can pick *treeGenerator*'s traversal strategy

And fuse *greedySolver* $\circ$ *treeGenerator*

Exceptions are interesting! Greed doesn't always work.

What else can I do with them?

We can find and explain optimisations:

For HM:
(not my trick! – MLton?)

Generalisation depth – generalisation constraints from root

- Annotate every $\exists\tau$ with its generalisation depth
- Constraints generalise variables at their depth
- Generalisation is now linear in type size

If you remember one thing...

Telescopic Constraint Trees do in space
what conventional checkers do in time

Some extras:

- Information effects track information well
- Constraints are a unit of information
- Information Aware Type Systems
- Telescopic Constraint Trees can be derived
- We can calculate, trade off and optimise from TCTs