Information Aware Type Systems and Telescopic Constraint Trees [extended mix]

Philippa Cowderoy

22nd November 2021

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?

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

- *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
  - Implementation of Example
- 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

Information Aware Type Systems – Methods

- 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

$$\begin{aligned} \tau = \tau &\quad \text{Type equality} \\ x : \tau \in \Gamma &\quad \text{Binding in context} \\ \Gamma' := \Gamma \,;\, x : \tau &\quad \text{Context extension} \\ \Gamma \multimap \Gamma,\, \Gamma &\quad \text{Context duplication} \end{aligned}$$

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

Information Aware Simply Typed λ-Calculus – *Var*

$$\begin{aligned} \tau = \tau &\quad \text{Type equality} \\ x : \tau \in \Gamma &\quad \text{Binding in context} \\ \Gamma' := \Gamma \,;\, x : \tau &\quad \text{Context extension} \\ \Gamma \multimap \Gamma,\, \Gamma &\quad \text{Context duplication} \end{aligned}$$

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

Information Aware Simply Typed λ-Calculus – *Lam*

$$\tau = \tau \quad \text{Type equality}$$
$$x : \tau \in \Gamma \quad \text{Binding in context}$$
$$\Gamma' := \Gamma \; ; \; x : \tau \quad \text{Context extension}$$
$$\Gamma \multimap \langle \; \Gamma, \; \Gamma \quad \text{Context duplication}$$

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

Information Aware Simply Typed λ-Calculus – *App*

$$\tau = \tau \quad \text{Type equality}$$
$$x : \tau \in \Gamma \quad \text{Binding in context}$$
$$\Gamma' := \Gamma \; ; \; x : \tau \quad \text{Context extension}$$
$$\Gamma \multimap \langle \; \Gamma, \; \Gamma \quad \text{Context duplication}$$

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

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	Checked type
$\Gamma^- \vdash T^+ : \tau^-$	types	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')

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

Information Aware Simply Typed λ-Calculus (moded)

Lam

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

$$\Gamma f^+ := \Gamma^- \; ; \; x^- : \tau p^+$$

$$\Gamma f^- \vdash T^- : \tau r^+$$

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

$$\Gamma^+ \vdash \lambda x^+ . T^+ \; : \; \tau f^- \quad \textit{Lam}$$

Information Aware Simply Typed λ-Calculus (moded)

App

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

$$\Gamma^{-} \prec \Gamma f^{+},\ \Gamma p^{+}$$

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

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

$$\Gamma^{+} \vdash T f^{+}\ T p^{+} : \tau r^{-} \qquad App$$

Information Aware Simply Typed λ-Calculus

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

$$\frac{\begin{array}{c}\Gamma \multimap \langle \Gamma f, \, \Gamma p \\ \Gamma f \vdash Tf : \tau f \qquad \Gamma p \vdash Tp : \tau p \\ \tau p \to \tau r = \tau f\end{array}}{\Gamma \vdash Tf \;\; Tp : \tau r} \qquad App$$

Surprise! Annotations

$$\tau a^- \prec \langle\, \tau ap^+,\ \tau af^+$$

$$\Gamma f^+ := \Gamma\ ;\ x^- : \tau ap^+$$

$$\Gamma f^- \vdash T^- : \tau r^+$$

$$\tau f^+ = \tau af^- \rightarrow^+ \tau r^-$$

$$\frac{}{\Gamma^+ \vdash \lambda x^+ {:} \tau a^+ . T^+ \ :\ \tau f^-}\ ALam$$

- The annotation is (incidentally) redundant in STLC
- Other dataflow options exist
- This lets us reinvent bidirectional typechecking

Implementation

We have a set of syntax-directed typing rules, so:

- Generate constraints
  - By folding the typing rules over the AST
- Then solve the constraints!

Implementation

- We're at the mode $\Gamma^+ \vdash T^+ : \tau^-$
- T is fixed by the fold, leaving Γ^+ and τ^-

```
type Judgment = Context -> ConstraintGen Type
```

- ConstraintGen is an Applicative
- Typing rules take subterm Judgments, local syntax
- . . . and produce new Judgments

Implementation – ConstraintGen

`ConstraintGen` is essentially wrapped state, providing:

- Fresh variable/name generation via `newVar` function
  - Overloaded on the type of the logic variable
  - *n*-ary `newVars` variant
- Constraint generation – keeps a list of constraints
- Binding judgment results to logic variables with `<<-`

Implementation – App rule

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

```
app :: Judgement -> Judgement -> Judgement
app jf jp c = withVars $ \tf tp tr cf cp ->
  tr |- c `ctxtDup` cf & cp                *>
        tf <<- jf cf                       *>
        tp <<- jp cp                       *>
        (tp :-> tr) `eqCon` tf
```

Implementation – Resulting Constraints

```
ghci> :{
| runConstraintGen $
|    check (Lam "x" $ Var "x") Empty
| :}
```

```
(TVar 0,
 GS {constraints =
       [EqCon (TVar 0) (TVar 1 :-> TVar 2),
        EqCon (TVar 2) (TVar 3),
        CtxtIn ("x",TVar 3) (CVar 0),
        CtxtExt (CVar 0) ("x",TVar 1) Empty],
     nextTV = 4, nextCV = 1})
```

Implementation – Result

```
ghci> typecheck (Lam "x" $ Var "x")
TVar 1 :-> TVar 1
```

Implementation – Fusion?

`typecheck` is little more than `solve . generate`

Why not fuse that away and solve as we go?

Implementation – Fusion?

`typecheck` is little more than `solve . generate`

Why not fuse that away and solve as we go?

- All our constraints succeed or fail immediately
- In larger languages, constraints can block
- We don't always have all the information ready!

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': $[\![T]\!]\,\tau$

- 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 this tree:

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

τ acts as a query variable

Building Var

This typing rule:

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

Becomes this TCT rule:

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

Building Lam

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

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

Building App

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

$$[\![Tf \quad Tp]\!]\ \tau r^{-} = \{\exists \tau f,\ \exists \tau p,\ \tau p^{-} \to \tau r^{+} = \tau f^{-}\} \mid [\![Tf]\!]\ \tau f^{+} \mid [\![Tp]\!]\ \tau p^{+}\ (App)$$

An Example TCT

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

$$
\begin{array}{l|r}
\{\}, & \Gamma \\
\quad \{\exists \tau q\}, & \text{Query} \\
\quad\quad \{\exists \tau f,\ \exists \tau p,\ \tau p \to \tau q = \tau f\}, & \mathit{App} \\
\quad\quad\quad |\{\exists \tau p',\ \exists \tau r,\ \tau f = \tau p' \to \tau r\}, \ldots & \lambda x. \\
\quad\quad\quad \ldots \{!x : \tau p'\}, & \\
\quad\quad\quad\quad \{?x : \tau r\} & x \\
\quad\quad\quad |\{\tau p = \text{String}\} & \text{String}
\end{array}
$$

An Example TCT – Some Solving (1)

Solve $!x : \tau p'$:

$$
\begin{array}{l|r}
\{\}, & \Gamma \\
\quad \{\exists \tau q\}, & \text{Query} \\
\qquad \{\exists \tau f,\ \exists \tau p,\ \tau p \to \tau q = \tau f\}, & \mathit{App} \\
\qquad\quad |\{\exists \tau p',\ \exists \tau r,\ \tau f = \tau p' \to \tau r\}, \ldots & \lambda x. \\
\qquad\quad \ldots \{\ !x : \tau p'\ \ x : \tau p'\ \}, & \\
\qquad\qquad \{?x : \tau r\} & x \\
\qquad\quad |\{\tau p = \mathrm{String}\} & \text{String}
\end{array}
$$

An Example TCT – Some Solving (2)

Solve $?x : \tau r$:

$$
\begin{array}{l|r}
\{\}, & \Gamma \\
\quad \{\exists \tau q\}, & \text{Query} \\
\qquad \{\exists \tau f,\ \exists \tau p,\ \tau p \to \tau q = \tau f\}, & App \\
\qquad\quad |\{\exists \tau p',\ \exists \tau r{=}\tau p',\ \ldots & \lambda x. \\
\qquad\quad \ldots \tau f = \tau p' \to \tau p'\},\ \{x : \tau p'\}, & \\
\qquad\qquad \{\ ?x : \tau r\ \} & x \\
\qquad\quad |\{\tau p = \mathrm{String}\} & \text{String}
\end{array}
$$

An Example TCT – Some Solving (3)

Solve $\tau f = \tau p' \to \tau p'$:

$$
\begin{array}{l|r}
\{\}, & \Gamma \\
\quad \{\exists \tau q\}, & \text{Query} \\
\quad\quad \{\exists \tau p',\ \exists \tau f = \tau p' \to \tau p',\ \exists \tau p,\ \ldots & \mathit{App} \\
\quad\quad \ldots \tau p \to \tau q = \tau p' \to \tau p'\}, & \\
\quad\quad\quad |\{\tau f = \tau p' \to \tau p'\},\ \{x : \tau p'\}, & \lambda x. \\
\quad\quad\quad\quad \{\} & x \\
\quad\quad\quad |\{\tau p = \mathrm{String}\} & \text{String}
\end{array}
$$

An Example TCT – Some Solving (4)

Solve $\tau p = \text{String}$

$$
\begin{array}{l|r}
\{\}, & \Gamma \\
\quad \{\exists \tau q\}, & \text{Query} \\
\quad\quad \{\exists \tau p', \ \exists \tau p = \text{String}, \ \ldots & \mathit{App} \\
\quad\quad \ldots \text{String} \to \tau q = \tau p' \to \tau p'\}, & \\
\quad\quad\quad |\{\}, \ \{x : \tau p'\}, & \lambda x. \\
\quad\quad\quad\quad \{\} & x \\
\quad\quad\quad |\{\ \tau p = \text{String}\ \} & \text{String}
\end{array}
$$

An Example TCT – Some Solving (5.1)

Simplify $\text{String} \to \tau q = \tau p' \to \tau p'$ one step:

$$
\begin{array}{l|r}
\{\}, & \Gamma \\
\quad \{\exists \tau q\}, & \text{Query} \\
\quad\quad \{\exists \tau p', \ \text{String} \to \tau q = \tau p' \to \tau p', \ldots & \mathit{App} \\
\quad\quad \ldots \text{String} = \tau p', \ \tau q = \tau p' \}, & \\
\quad\quad\quad |\{\}, \ \{x : \tau p'\}, & \lambda x. \\
\quad\quad\quad\quad \{\} & x \\
\quad\quad\quad |\{\} & \text{String}
\end{array}
$$

An Example TCT – Some Solving (5.2)

Solve $\text{String} = \tau p'$:

$$
\begin{array}{l|r}
\{\}, & \Gamma \\
\quad \{\exists \tau q\}, & \text{Query} \\
\qquad \{\exists \tau p' = \text{String}, \ \ldots & \mathit{App} \\
\qquad \ldots \text{String} = \tau p', \ \tau q = \text{String}\}, & \\
\qquad\quad |\{\}, \ \{x : \tau p'\}, & \lambda x. \\
\qquad\qquad \{\} & x \\
\qquad\quad |\{\} & \text{String}
\end{array}
$$

An Example TCT – Some Solving (5.3)

Solve $\tau q = \text{String}$:

$$
\begin{array}{l|r}
\{\}, & \Gamma \\
\quad \{\exists \tau q{=}\text{String}\}, & \text{Query} \\
\quad\quad \{\tau q = \text{String}\}, & \mathit{App} \\
\quad\quad\quad |\{\},\ \{x : \tau p'\}, & \lambda x. \\
\quad\quad\quad\quad \{\} & x \\
\quad\quad\quad |\{\} & \text{String}
\end{array}
$$

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
- Constraints $\Leftrightarrow$ Source Location for provenance?
- UI for type level debugging?
- All of the above for elaboration problems?

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 from TCTs