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

Philippa Cowderoy

22nd November 2021

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

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

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

• Telescopic Constraint Trees

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

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

What does this τ say?

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?)

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

- 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

- Motivating 'information awareness' DONE
- Tools for information awareness
 - Information Effects
 - Constraints

- Information Aware Type Systems
- Telescopic Constraint Trees

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

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

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

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

Breaches of conservation of information:

- When information is created
- When information is destroyed

Breaches of conservation of information:

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

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

Syntax

Static Semantics

Syntax

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

Static Semantics

Syntax

Static Semantics

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

Syntax

Static Semantics

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

Syntax

Static 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

Syntax **Static Semantics Dynamic Semantics** Insight Constraints are a unit of information!

Image: A = 1 = 1

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

• 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

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

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

Information Aware Simply Typed λ -Calculus – Var

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

$$x: \tau \in \Gamma$$

$$\overline{\Gamma \vdash x: \tau} Var$$

Information Aware Simply Typed λ -Calculus – Lam

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

 $\Gamma f := \Gamma ; x : \tau p$ $\Gamma f \vdash T : \tau r$ $\tau f = \tau p \rightarrow \tau r$

 $\Gamma \vdash \lambda x.T$: τf Lam

Philippa Cowderoy

Information Aware Simply Typed λ -Calculus – App

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

 $\Gamma - \langle \Gamma f, \Gamma p$ $\Gamma f \vdash Tf : \tau f \qquad \Gamma p \vdash Tp : \tau p$ $\tau p \rightarrow \tau r = \tau f$

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

Philippa Cowderoy

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

- 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
$\Gamma^+ \vdash T^+ : au^-$		Synthesis
$\Gamma^- \vdash T^+ : \tau^+$	Free variable	Checked type
$\Gamma^- \vdash T^+: au^-$	types	Synthesised type
$\Gamma^+ \vdash T^-$: $ au^+$	Proof search	
	Program	
	synthesis	

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

Philippa Cowderoy

18 / 55

The function arrow → doesn't appear in source
It does appear in our types

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

- 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

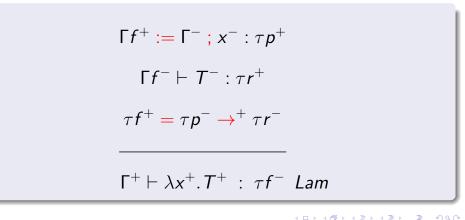
Mode: $\Gamma^+ \vdash T^+$: τ^- (Synthesis or 'typechecking')

$$x^{-}:\tau^{+}\in\Gamma^{-}$$

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

Information Aware Simply Typed λ -Calculus (moded) Lam

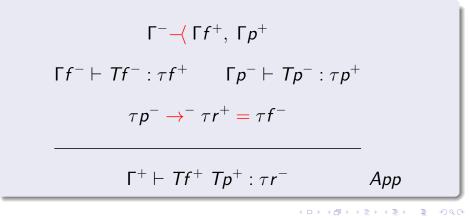
Mode: $\Gamma^+ \vdash T^+$: τ^- (Synthesis or 'typechecking')



Philippa Cowderoy

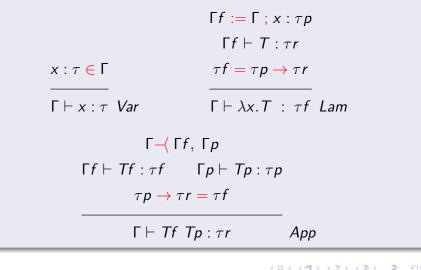
Information Aware Simply Typed λ -Calculus (moded) App

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



Philippa Cowderoy

Information Aware Simply Typed λ -Calculus



Surprise! Annotations

$$\tau a^{-} \not\leftarrow \tau a p^{+}, \ \tau a f^{+}$$

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

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

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

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

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

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

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

- We're at the mode $\Gamma^+ \vdash T^+$: au^-
- ${\cal T}$ 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

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

$$\Gamma - \langle \Gamma f, \Gamma p$$

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

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

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

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

```
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)
```

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

3

typecheck is little more than solve . generate

Why not fuse that away and solve as we go?

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!

- 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!

- A Telescopic Constraint Tree (or TCT) is:
 - A Tree a refined AST
 - Containing Constraints
 - Telescopic

- A Telescopic Constraint Tree (or TCT) is:
 - A Tree a refined AST
 - Containing Constraints
 - Also metavariable binders
 - Telescopic

- A Telescopic Constraint Tree (or TCT) is:
 - A Tree a refined AST
 - Containing Constraints
 - Also metavariable binders
 - Telescopic
 - Built from composable contexts

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

- 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 an ordinary typechecker does

What an ordinary typechecker does in time

Telescopic constraint trees do in space

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!

Old	New
<i>x</i> : <i>τ</i> ∈ Γ	
$\Gamma' := \Gamma$; $x : \tau$	\mathbf{x} : $ au$

The new constraints are *situated*.

Their meaning depends on their position in the telescope:

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

We can ask "is this solved?"

Context duplications are tree branches!

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!

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

We use this rule to build this tree:

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

 τ acts as a query variable

(Start)

This typing rule: $x^-: \tau^+ \in \Gamma$

$$I \vdash x' : \tau$$
 Var

Becomes this TCT rule:

$$\llbracket x^+ \rrbracket \tau^- = \{?x^- : \tau^+\}$$

Building Lam

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

$$\Gamma f \vdash T : \tau r^{+}$$

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

$$\overline{\Gamma \vdash \lambda x.T} : \tau f^{-} Lam$$

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

æ

$$\Gamma - \langle \Gamma f, \Gamma p$$

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

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

 $\Gamma \vdash Tf \ Tp : \tau r^{-} \qquad App$

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

∃ ► < ∃ ►</p>

æ

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

An Example TCT – Some Solving (1)

Solve $x: \tau p'$:

An Example TCT – Some Solving (2)

Solve $?x : \tau r$:

An Example TCT – Some Solving (3)

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

An Example TCT – Some Solving (4)

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

< □ > < 同 >

æ

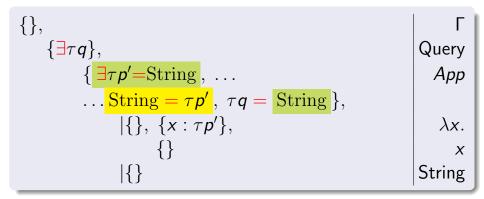
An Example TCT – Some Solving (5.1)

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

Image: A matrix

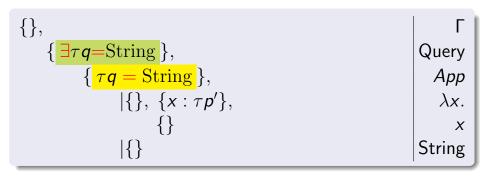
An Example TCT – Some Solving (5.2)

Solve String = $\tau p'$:



An Example TCT – Some Solving (5.3)

Solve $\tau q =$ String:



We can, of course, implement typecheckers.

We can also instrument those checkers:

- Changes in the tree show the process, including scopeThose changes preserve the AST structure
 - Mostly labels are useful extra info
- Constraints ⇔ Source Location for provenance?
- UI for type level debugging?
- All of the above for elaboration problems?

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

55 / 55