Information-Aware Type Systems

Philippa Cowderoy

SPLS March 2019

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

Symbol Games and Hidden Information

Our standard notation hides things from us.

$$\frac{\Gamma \vdash Tp : \tau p \qquad \Gamma \vdash Tf : \tau p \rightarrow \tau r}{\Gamma \vdash Tf\ Tp : \tau r}\ App1$$

$$\frac{\Gamma \vdash Tp : \tau p \qquad \Gamma \vdash Tf : \tau f \qquad \tau p \rightarrow \tau r = \tau f}{\Gamma \vdash Tf\ Tp : \tau r}\ App2$$

- While we are used to $App1$, $App2$ is easier for beginners to understand – an implicit constraint is made explicit
- Generating that constraint is new information
- Can we make the possibilities in our systems clearer?

What are Information-Aware Type Systems?

An Information-Aware Type System is a type system where:

- It is clear where information is introduced and eliminated
- It is clear (or at least clearer) how information flows within the type system

This is achieved by using *information effects* to track where information is created and destroyed - or if you prefer, where the system violates *conservation of information*. We hope inferences tell us something new!

How To Make A System Information-Aware

This is just one recipe, but it's pretty reliable:

- Linear logic variables: one +ve occurrence, one -ve
- Constraints:
  - Constraint generation is an information effect
  - Constraints give us an abstraction tool
  - Constraints help avoid *overconstraining* data flow
- Duplication effects: track dataflow branches and merges
- Mode analysis: keep track of which way data flows, which forms of constraints we can solve

Constraints for the Simply Typed Lambda Calculus

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

Convention: $=$ is always written as if 'assigning' to the LHS

Note that the context constraints encode the structural rules. An alternative interpretation could give us a minimal linear calculus.

A modified $\prec$ might be suitable for Quantitative Type Theory!

Information-Aware Simply Typed λ-Calculus – *Var*

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

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

Information-Aware Simply Typed λ-Calculus – *Lam*

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

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

Information-Aware Simply Typed λ-Calculus – *App*

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

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

Information-Aware Simply Typed λ-Calculus

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

$$\frac{\begin{array}{c}\Gamma \mathrel{-\!\!\!\langle}\substack{\Gamma L \\ \Gamma R} \\ \Gamma L \vdash Tf : \tau f \qquad \Gamma R \vdash Tp : \tau p \\ \tau p \to \tau r = \tau f\end{array}}{\Gamma \vdash Tf\ Tp : \tau r}\ \mathit{App}$$

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	

- Systems that only support checking modes may not be *algorithms*, but they're typecheckers and not type systems.
- I'm not aiming to actively *support* program synthesis. Without syntax direction, it's search as usual.

$\rightarrow$ - The Other Information Effect

- The function arrow $\rightarrow$ doesn't appear in the source language, but it does appear in our types.
  - Not simply isomorphic to something in the term
  - Part of our (abstract) *interpretation* of a term
- Information we *generate* 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
  - Construction vs pattern-matching

Modes for $\rightarrow$ - 1

$\tau 1^{+} = (\tau 2^{-} \rightarrow^{+} \tau 3^{-})$

- $\rightarrow$ behaves as a *constructor* assigned to $\tau 1$
- $\tau 2$ and $\tau 3$ have -ve mode
  - They are being consumed to construct something to match against

Modes for $\rightarrow$ – 2

$(\tau 1^{-} \rightarrow^{-} \tau 2^{+}) = \tau 3^{-}$

- $\rightarrow$ behaves as a *pattern* matched against $\tau 3$
- $\tau 2$ is $+$ve, act as a variable pattern
  - Produce something to use elsewhere
- $\tau 1$ is -ve, acts as a constructor pattern
  - Matched against, but generates no new local information
  - May still contain other unknowns

Modes for $\to$ – 3

During solving:

- $\to^+$ creates or introduces information
- $\to^-$ destroys or eliminates information

Why mention introduction and elimination?

- $\to^+$ appears in the *Lam* rule, aka $\to I$
- $\to^-$ in *App*, aka $\to E$
- The modes are telling us about introducing and eliminating connectives!

Contextual Behaviour

Context extension and binding constraints also have a relationship.

Read one way:

- $\Gamma' := \Gamma \,;\, x : \tau$ introduces the need for a binding
- $x : \tau \in \Gamma$ makes use of - or especially in linear and affine systems eliminates a binding

This can also be read in reverse:

- Using a variable requires it to be bound
- Providing a binding meets that requirement!

Likewise, $\Gamma \prec^{\Gamma L}_{\Gamma R}$ can be read as merging ΓL and ΓR.

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')

$$\frac{\begin{array}{c} \Gamma f^{+} := \Gamma^{-} \; ; \; x^{-} : \tau p^{+} \\ \Gamma f^{-} \vdash T^{-} : \tau r^{+} \\ \tau f^{+} = \tau p^{-} \rightarrow^{+} \tau r^{-} \end{array}}{\Gamma^{+} \vdash \lambda x^{+}.T^{+} \; : \; \tau f^{-}} \; Lam$$

Information-Aware Simply Typed λ-Calculus (moded)
App

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

$$
\dfrac{
\begin{array}{c}
\Gamma^- \prec \begin{smallmatrix}\Gamma f^+ \\ \Gamma p^+\end{smallmatrix} \\[1ex]
\Gamma f^- \vdash Tf^- : \tau f^+ \qquad \Gamma p^- \vdash Tp^- : \tau p^+ \\[1ex]
\tau p^- \to^- \tau r^+ = \tau f^-
\end{array}
}{
\Gamma^+ \vdash Tf^+ \; Tp^+ : \tau r^-
}\; \mathit{App}
$$

Information Omitted Due To Constraints
(Further Work)

- *Telescopic Trees* – mapping Informaton-Aware systems to a structure representing a typechecking problem in progress
  - Notation matches techniques in use from LEGO to Idris
  - Permits a 'Type Inference in Context' style
  - Reuses ideas about dataflow from Bastiaan Heeren's work
- Information-Aware Elaboration
  - Desugaring also requires information effects!
- Re-examining constraints
  - Possibly with session types?
    (suggestion due to Conor McBride)

Optional: Annotations, Duplication & Bidirectionality

Let's support annotations!

- We are forced to duplicate a type
- We could duplicate the function type to check then return
- Better: send the annotation both 'in' and 'out'

$$
\frac{\begin{array}{c} \tau a \multimap\!\langle^{\tau ap}_{\tau af} \\ \Gamma f := \Gamma \,;\, x : \tau ap \\ \Gamma f \vdash T : \tau r \\ \tau f = \tau af \rightarrow \tau r \end{array}}{\Gamma \vdash \lambda x{:}\tau a.T \;:\; \tau f} \; ALam
$$

Extra: Free Join-the-Dots slide! *Connect +ve to −ve*

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

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

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