Information-Aware Type Systems

Philippa Cowderoy

SPLS March 2019

e-mail: flippa@flippac.org
twitter: @flippacpub
Symbol Games and Hidden Information

Our standard notation hides things from us.

\[ \Gamma \vdash Tp : \tau p \]
\[ \Gamma \vdash Tf : \tau f \]
\[ \Gamma \vdash Tf : \tau p \rightarrow \tau r \]
\[ \tau p \rightarrow \tau r = \tau f \]
\[ \Gamma \vdash Tf \ Tp : \tau r \quad \text{App1} \]
\[ \Gamma \vdash Tf \ Tp : \tau r \quad \text{App2} \]

▶ 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

\[ \tau = \tau \quad \text{Type equality} \]
\[ \tau \langle \tau_l \rangle \langle \tau_r \rangle \quad \text{Type duplication} \]

\[ x : \tau \in \Gamma \quad \text{Binding in context} \]
\[ \Gamma' \coloneqq \Gamma ; x : \tau \quad \text{Context extension} \]
\[ \Gamma \langle \Gamma_L \rangle \langle \Gamma_R \rangle \quad \text{Context duplication} \]

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 \[ \langle \\rangle \] might be suitable for Quantitative Type Theory!
Information-Aware Simply Typed \( \lambda \)-Calculus – \( \text{Var} \)

\[
\begin{align*}
x : \tau &\in \Gamma \\
\hline
\Gamma \vdash x : \tau &\quad \text{Var}
\end{align*}
\]

\[
\begin{align*}
\tau = \tau &\quad \text{Type equality} \\
\tau \mathrel{\Delta}^{\triangleright}_{\triangleright} &\quad \text{Type duplication} \\
x : \tau &\in \Gamma \quad \text{Binding in context} \\
\Gamma' &:= \Gamma ; x : \tau \quad \text{Context extension} \\
\Gamma \mathrel{\Delta}^{\ll}_{\ll} &\quad \text{Context duplication}
\end{align*}
\]
Information-Aware Simply Typed $\lambda$-Calculus – \textit{Lam}

$$
\Gamma f := \Gamma ; x : \tau p
\quad \Gamma f \vdash T : \tau r
\quad \tau f = \tau p \rightarrow \tau r
\quad \frac{\vdash \lambda x. T : \tau f}{\Gamma \vdash \lambda x. T : \tau f \text{ Lam}}
$$

\begin{align*}
\tau &= \tau & \text{Type equality} \\
\tau &\xrightarrow{\tau l / \tau r} & \text{Type duplication} \\
\quad x : \tau &\in \Gamma & \text{Binding in context} \\
\Gamma' &:= \Gamma ; x : \tau & \text{Context extension} \\
\quad \Gamma &\xrightarrow{\Gamma L / \Gamma R} & \text{Context duplication}
\end{align*}
Information-Aware Simply Typed λ-Calculus – App

\[
\Gamma \vdash_{\Gamma L}^{\Gamma R}
\]

\[
\Gamma L \vdash Tf : \tau_f \quad \Gamma R \vdash Tp : \tau_p
\]
\[
\tau p \rightarrow \tau r = \tau f
\]

\[
\Gamma \vdash Tf \ Tp : \tau r \quad \text{App}
\]

\[
\tau = \tau \quad \text{Type equality}
\]

\[
\tau \vdash_{\tau}^{\tau/\tau_r} \quad \text{Type duplication}
\]

\[
\mathbf{x} : \tau \in \Gamma \quad \text{Binding in context}
\]

\[
\Gamma' := \Gamma ; \ x : \tau \quad \text{Context extension}
\]

\[
\Gamma \vdash_{\Gamma L}^{\Gamma R} \quad \text{Context duplication}
\]
Information-Aware Simply Typed λ-Calculus

\[ \Gamma f := \Gamma ; x : \tau p \]
\[ \Gamma f \vdash T : \tau r \]
\[ x : \tau \in \Gamma \]
\[ \Gamma \vdash x : \tau \ Var \]
\[ \tau f = \tau p \rightarrow \tau r \]
\[ \Gamma \vdash \lambda x. T : \tau f \ Lam \]

\[ \Gamma \leftarrow \Gamma L \rightarrow \Gamma R \]
\[ \Gamma L \vdash Tf : \tau f \quad \Gamma R \vdashTp : \tau p \]
\[ \tau p \rightarrow \tau r = \tau f \]
\[ \Gamma \vdash Tf \ Tp : \tau r \quad App \]
## Different Modes of a Type System

<table>
<thead>
<tr>
<th>Mode</th>
<th>Unidirectional</th>
<th>Bidirectional</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Gamma^+ \vdash T^+ : \tau^+$</td>
<td>Type Checking</td>
<td>Checking</td>
</tr>
<tr>
<td>$\Gamma^+ \vdash T^+ : \tau^-$</td>
<td>Synthesis</td>
<td>Synthesis</td>
</tr>
<tr>
<td>$\Gamma^- \vdash T^+ : \tau^+$</td>
<td>Free Variable Types</td>
<td>Checked type</td>
</tr>
<tr>
<td>$\Gamma^- \vdash T^+ : \tau^-$</td>
<td>Synthesised Type</td>
<td>Synthesised Type</td>
</tr>
<tr>
<td>$\Gamma^+ \vdash T^- : \tau^+$</td>
<td>Proof search</td>
<td>Program Synthesis</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 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.
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 $\rightarrow - 2$

$$(\tau_1^- \rightarrow^\rightarrow \tau_2^+) = \tau_3^-$$

- $\rightarrow$ 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 $\rightarrow - 3$

During solving:

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

Why mention introduction and elimination?

- $\rightarrow^+$ appears in the Lam rule, aka $\rightarrow I$
- $\rightarrow^-$ in App, aka $\rightarrow E$
- 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, $\Gamma \mid_{\Gamma L, \Gamma R}$ can be read as merging $\Gamma L$ and $\Gamma R$. 
Information-Aware Simply Typed $\lambda$-Calculus (moded) \\
Var

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

\[ x^- : \tau^+ \in \Gamma^- \]
\[ \Gamma^+ \vdash x^+ : \tau^- \quad Var \]
Mode: $\Gamma^+ \vdash T^+ : \tau^-$ (Synthesis or ‘typechecking’)

$$
\begin{align*}
\Gamma f^+ & := \Gamma^- \ ; \ x^- : \tau p^+ \\
\Gamma f^- & \vdash T^- : \tau r^+ \\
\tau f^+ & = \tau p^- \rightarrow^+ \tau r^-
\end{align*}
$$

$$
\Gamma^+ \vdash \lambda x^+. T^+ : \tau f^- \ \text{Lam}
$$
Information-Aware Simply Typed $\lambda$-Calculus (moded)

**App**

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

$$
\Gamma^+ \vdash \frac{\Gamma^+ \vdash T^+ : \tau^-}{\Gamma^+ \vdash T^+ T^+ : \tau^+_r} \quad \text{App}
$$

$$
\Gamma^+ \vdash \frac{\Gamma f^- \vdash Tf^- : \tau f^+ \quad \Gamma p^- \vdash Tp^- : \tau p^+}{\tau p^- \rightarrow^\tau r^+ = \tau f^-}
$$
Information Omitted Due To Constraints

*Further Work*

- **Telescopic Trees** – mapping Information-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’

\[
\begin{align*}
\tau a & \xrightarrow{(\tau ap)_{\tau af}} \\
\Gamma f & := \Gamma ; x : \tau ap \\
\Gamma f & \vdash T : \tau r \\
\tau f & = \tau af \rightarrow \tau r \\
\hline \\
\Gamma & \vdash \lambda x: \tau a. T : \tau f & \text{ALam}
\end{align*}
\]
Extra: Free Join-the-Dots slide!  \textit{Connect }+ve\textit{ to }−ve

\begin{align*}
\Gamma f^+ & := \Gamma^- ; x^- : \tau p^+ \\
\Gamma f^- & \vdash T^- : \tau r^+
\end{align*}

\begin{align*}
x^- : \tau^+ & \in \Gamma^- \\
\Gamma^+ & \vdash x^+ : \tau^- \text{ Var} \\
\tau f^+ & = \tau p^- \rightarrow^+ \tau r^-
\end{align*}

\begin{align*}
\Gamma^+ & \vdash \lambda x^+. T^+ : \tau f^- \text{ Lam}
\end{align*}

\begin{align*}
\Gamma^- & \xrightarrow{\Gamma^+} \Gamma f^+
\Gamma f^- & \vdash Tf^- : \tau f^+ \\
\Gamma p^- & \vdash Tp^- : \tau p^+
\end{align*}

\begin{align*}
\tau p^- & \rightarrow^- \tau r^+ = \tau f^-
\end{align*}

\begin{align*}
\Gamma^+ & \vdash Tf^+ Tp^+ : \tau r^- \text{ App}
\end{align*}