Type and Translation Rules for Arrow Notation in GHC

Ross Paterson        Simon Peyton Jones

September 16, 2004

GHC is full.
SIMON MARLOW (shortly before arrow notation was added)

Simple type rules

Let’s assume that ordinary Haskell type judgements have the form \( \Gamma \vdash e :: \tau \), ignoring type constraints for simplicity. The rules below define a new judgement of the form

\[
\Gamma \mid \Delta \vdash_\rightarrow a \cdot c :: \tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow \tau
\]

The components of this judgement are as follows:

\( \Gamma \) an ordinary Haskell environment, for variables introduced outside the current proc-expression.

\( \Delta \) typings of variables introduced in the current proc-expression.

\( a \) the arrow type of the command being constructed.

\( c \) a command: a new syntactic category. Many of the forms resemble expression forms, but have a different semantics.

\( \tau_i \) the types of a stack of additional, anonymous inputs to the command. \( \tau_1 \) is the type of the top element.

\( \tau \) the type of values returned by the command.

Note that \( \rightarrow \) is not a type constructor; \( \tau \) and \( \tau_i \) are ordinary Haskell types, but \( \tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow \tau \) is something else.
Typing and translation rules for arrow notation

These rules take a judgement

\[ \Gamma \vdash a \ c :: \tau \]

and produce an expression \( c^* \) such that

\[ \Gamma \vdash c^* :: a \ ((\tau_\Delta), \tau_1), \ldots, \tau_n) \tau \]

where \((\tau_\Delta)\) means a tuple of the types of \(\Delta\).
\[
p :: \tau \Rightarrow \Delta \\
\Gamma \vdash a :: \tau' \\
\Gamma \vdash \text{proc } p \rightarrow c :: a \tau \tau' \\
\Gamma \vdash f :: a (\tau, \bar{\tau}) \tau' \\
\Gamma \vdash e :: \tau \\
\]
Do notation

\[
\begin{align*}
\Gamma \vdash \Delta \vdash c :: \tau & \quad \mapsto c^* \\
\Gamma | \Delta \vdash a \{ c \} :: \tau & \quad \mapsto c^* \\
\Delta, \text{binds} \Rightarrow \Delta' & \quad \mapsto k = \lambda (\Delta) \to \text{let binds in } (\Delta') \\
\Gamma | \Delta' \vdash a \{ s \} :: \tau & \quad \mapsto s^* \\
\Gamma | \Delta' \vdash a \{ \text{let binds}; s \} :: \tau & \quad \mapsto \text{arr } k \gg s^*
\end{align*}
\]

\[
\begin{align*}
\Delta \subseteq \Delta_1 \cup \Delta_2 & \quad \mapsto k = \lambda (\Delta) \to ((\Delta_1), (\Delta_2)) \\
\Gamma | \Delta_1 \vdash a c :: \tau & \quad \mapsto c^* \\
\Gamma | \Delta_2 \vdash a s :: \tau' & \quad \mapsto s^* \\
\Gamma | \Delta \vdash a \{ c; s \} :: \tau' & \quad \mapsto \text{arr } k \gg \text{first } c^* \gg \text{arr snd } s^* \\
\Delta \subseteq \Delta_1 \cup \Delta_2 & \quad \mapsto k = \lambda (\Delta) \to ((\Delta_1), (\Delta_2)) \\
\Gamma | \Delta_1 \vdash a c :: \tau & \quad \mapsto c^* \\
p :: \tau, \Delta_2 \Rightarrow \Delta' & \quad \mapsto k' = \lambda (p, (\Delta_2)) \to (\Delta') \\
\Gamma | \Delta' \vdash a s :: \tau' & \quad \mapsto s^* \\
\Gamma | \Delta \vdash a \{ p \leftarrow c; s \} :: \tau' & \quad \mapsto \text{arr } k \gg \text{first } c^* \gg \text{arr } k' \gg s^*
\end{align*}
\]