A Foundation for Dependently Typed Haskell

Stephanie Weirich
University of Pennsylvania
What if Haskell was based, not on the Hindley-Milner type system, but on a different ML type system?

An alternate history…
Proposal

• Base Haskell on a core dependently-typed language with ★ : ★

\[
\text{terms, types }\quad a, b, A, B \quad ::= \quad \star \mid x \mid \lambda x : A. b \mid a \ b \\
\mid \Pi x : A. B
\]

• Full-spectrum dependently typed language with a single sort
• Proposed by Martin Löf (1971 draft paper)
• Not logically consistent
• Not good for proof checking \(...\text{but neither is Haskell}\)
• Type checking is undecidable
• Type sound (Cardelli 1985) and \textit{supremely} uniform
• Acknowledgements: Conor, Adam Gundry, Richard
• Subsumes higher-order polymorphism, type families, kind polymorphism, etc.

\[
\begin{align*}
\Gamma, x : A & \vdash B : \star \quad \Gamma \vdash A : \star \\
\frac{\Gamma \vdash \Pi x : A. B : \star}{\Gamma \vdash B_{\text{PI}}} \\
\Gamma \vdash b : \Pi x : A. B \\
\Gamma \vdash a : A \\
\frac{\Gamma \vdash b \ a : B \{a/x\}}{\Gamma \vdash B_{\text{APP}}} \\
\end{align*}
\]
Caveats

• This is work in progress
• I'm not going to say anything about type inference
• It gets complicated…
• And, it is easy to make mistakes when working with these systems

• However, all theorems have been mechanically verified by Coq
  – LaTeX rules generated from same source
  – I'm overly impressed by my own "trivial" proofs
+ Coercion abstraction

- GADTs require \textit{propositional equality}: the ability of the type system to assume equality

\begin{verbatim}
data T :: * -> * where
    TInt :: forall a. (Int ~ a) => T a

f :: forall a. T a -> a
f TInt = 0 + 0
\end{verbatim}

- Can we just encode propositional equality?
  \[\textit{a} \sim \textit{b} \triangleq \Pi c : (\star \to \star). \textit{c} \textit{a} \to \textit{c} \textit{b}\]

- No!
  - Logic is inconsistent --- need to run proofs
  - Type inference decides where coercions are placed in terms.

\begin{verbatim}
f :: forall a. T a -> a
f (TInt @c) = ((\textit{0} + \textit{0}) \triangleright c)
f (TInt @c) = (+ \triangleright (\textit{Int} \to \textit{Int} \to c)) \textit{0} \textit{0}
\end{verbatim}
Coercion abstraction

```
data T :: * -> * where
  TInt :: forall a. (Int ~ a) => T a
```

- GADTs require *propositional equality*: the ability of the type system to reason about type (and term) equality
- Type soundness requires consistent propositional equality; cannot have a proof that \( \text{Int} \sim \text{Bool} \)
- Elaboration requires irrelevant type coercion; it cannot matter how we use propositional equality

- *So \((a \sim b)\) proposition CANNOT be a type*

  FC solution: separate language of equality proofs
Dependent types + coercions

- A core dependently-typed language with \( \star : \star \) and explicit coercions

\[
\begin{align*}
\text{terms, types} & \quad a, b, A, B & ::= & \quad \star | x | \lambda x : A. b | a b \\
& & & \Pi x : A. B \\
& & & \Lambda c : \phi. a | a[\gamma] | \forall c : \phi. A \\
& & & a \triangleright \gamma
\end{align*}
\]

- Coercions are proof witnesses of equality between terms

\[
\Gamma; \Delta \vdash \gamma : a \sim b
\]
Coercion abstraction

\[
\Gamma \vdash \phi \ ok \\
\Gamma, c : \phi \vdash B : \ast \quad \text{AN\_CPI} \\
\Gamma \vdash \forall c : \phi. B : \ast \\
\Gamma \vdash \phi \ ok \\
\Gamma, c : \phi \vdash a : B \quad \text{AN\_CABS} \\
\Gamma \vdash \forall c : \phi. a : \forall c : \phi. B \\
\Gamma \vdash b : \forall c : a_1 \sim_{A_1} b_1. B \\
\Gamma; \text{dom}(\Gamma) \vdash \gamma : a_1 \sim b_1 \quad \text{AN\_CAPP} \\
\Gamma \vdash b[\gamma] : B\{\gamma/c\} \\
\Gamma \vdash a : A \\
\Gamma; \text{dom}(\Gamma) \vdash \gamma : A \sim B \\
\Gamma \vdash B : \ast \quad \text{AN\_CONV} \\
\Gamma \vdash a \triangleright \gamma : B
\]
Coercion proofs

\[ \Gamma; \Delta \vdash \gamma : a \sim b \]

• Coercions show that type equality...
  – is an equivalence relation
  – is congruent
  – is injective for type constructors (needed for preservation proof)
  – ignores coercions in terms (type conversion is irrelevant)
  – contains reduction (now type checking is decidable!)

• 21 different coercion rules total
Coercion proofs and types

• Design decision: if we can prove two terms equal, what do we know about their types?

\[ \Gamma; \Delta \vdash \gamma : a \sim b \]

– Nothing?
– They have the same type?
– There is a coercion between their types?

\[ \begin{align*}
\Gamma; \Delta &\vdash \gamma_1 : a_1 \sim b_1 \\
\Gamma; \Delta &\vdash \gamma_2 : a_2 \sim b_2 \\
\Gamma &\vdash a_1 \ a_2 : A \\
\Gamma &\vdash b_1 \ b_2 : B \\
\hline
\Gamma; \Delta &\vdash \gamma_1 \gamma_2 : a_1 \ a_2 \sim b_1 \ b_2 \quad \text{AN\_APP\_CONG}
\end{align*} \]
Consistency

• Progress lemma requires consistency

**Definition 1 (Consistency).** *Define consistent* $A \, B$ *to mean that if* $A$ *and* $B$ *are both types (i.e. of the form* $\ast$, $\Pi x : A. B$ *or* $\forall c : \phi. A$ *then they have the same form.***

• Proof based on confluence of parallel reduction

**Definition 2 (Joinable).**

$$
\frac{
\vdash a_1 \Rightarrow^* b \\
\vdash a_2 \Rightarrow^* b \\
}{\vdash a_1 \leftrightarrow a_2}
$$

**Theorem 3 (Joinability implies consistency).** If $\vdash A \leftrightarrow B$ then consistent $A \, B$.

**Theorem 4 (Equality implies Joinability).** If $\emptyset ; \emptyset \vdash \gamma : a \sim b$ then $\vdash a \leftrightarrow b$. 
A Difficulty

• Consider this equality

\[ \forall c : (\text{Int} \sim_\ast \text{Bool}).\text{Int} \equiv \forall c : (\text{Int} \sim_\ast \text{Bool}).\text{Bool} \]

• Cannot be derived via parallel reduction…
• Solution: restrict type system to rule out above equivalence
• Judgment form includes set of "available" coercions

\[ \Gamma \vdash \Gamma \]
\[ c : a \sim_A b \in \Gamma \]
\[ c \in \Delta \]

\[ \frac{c \in \Delta}{\Gamma, \Delta \vdash c : a \sim b} \text{ AN\_ASSN} \]
Equality for cpi

\[
\Gamma; \Delta \vdash \gamma_1 : \phi_1 \equiv \phi_2 \\
\Gamma, c : \phi_1; \Delta \vdash \gamma_3 : B_1 \sim (B_2 \{c/c\}) \\
B_3 = B_2 \{c \triangleright \text{sym} \gamma_1/c\} \\
\Gamma \vdash \forall c : \phi_1. B_1 : \star \\
\Gamma \vdash \forall c : \phi_2. B_3 : \star \\
\Gamma; \Delta \vdash (\forall c : \gamma_1. \gamma_3) : (\forall c : \phi_1. B_1) \sim (\forall c : \phi_2. B_3)
\]

\[
\Gamma; \Delta \vdash \phi_1 \equiv \phi_2 \\
\Gamma, c : \phi_1; \Delta \vdash A \equiv B : \star \\
\Gamma; \Delta \vdash \forall c : \phi_1. A \equiv \forall c : \phi_2. B : \star
\]

AN_CPICONG

E_CPICONG
Implicit Dependent FC

- Curry-style language: type annotations and coercions not present in terms

\[
\text{terms, types } a, b, A, B ::= \star | x | \lambda x.b | a \, b \\
| \Pi x : A.B \\
| \Lambda c.a | a[\gamma] | \forall c : \phi.A
\]

\[
\text{propositions } \phi ::= a \sim_A b
\]

\[
\text{coercions } \gamma ::= \bullet
\]

- Coercion replaced by definitional equality between types

\[
\frac{
\Gamma \vdash a : A \\
\Gamma; \text{dom}(\Gamma) \vdash A \equiv B : \star
}{\Gamma \vdash a : B}
\]

E_CONV
**Lemma 5 (Erasure).** • If $\Gamma \vdash a : A$ then $|\Gamma| \models |a| : |A|

• If $\Gamma; \Delta \vdash \gamma : a \sim b$ and $\Gamma \vdash a : A$, then $|\Gamma|; \Delta \models |a| \equiv |b| : |A|

**Lemma 6 (Annotation).** • If $\Gamma \vdash a : A$, then for all $\Gamma_0$ such that $|\Gamma_0| = \Gamma$, there exists $a_0$ and $A_0$, such that $|a_0| = a$, $|A_0| = A$, and $\Gamma_0 \vdash a_0 : A_0$.

• If $\Gamma; \Delta \vdash a \equiv b : A$, then for all $\Gamma_0$ such that $|\Gamma_0| = \Gamma$, there exists $\gamma$, $a_0$, $b_0$, and $A_0$, such that $|a_0| = a$, $|b_0| = b$, $|A_0| = A$, and $\Gamma_0; \Delta \vdash \gamma : a_0 \sim b_0$ and $\Gamma_0 \vdash a_0 : A_0$. 
Current status

- Proofs in Coq (24k LOC, 11k generated)
  - Preservation & progress for implicit and explicit languages
  - Types are unique for explicit language
  - Erasure and annotation theorems
  - Many, many design changes
  - Me + 2 students since April

- Extensions in flight
  - Implicit quantification (erasure for parametric arguments)
  - Recursion/type families
  - Datatypes and pattern matching
Open problem: Consistency

Can we prove a stronger, less syntactic consistency result?

– Get rid of "available set"
– Allow richer equalities in coercions (eta equivalence, induction principles, contextual equivalence)
– Enable parametricity-like reasoning for implicit quantification (i.e. free theorems)