A Foundation for Dependently Typed Haskell

Stephanie Weirich
University of Pennsylvania

An alternate history…

What if Haskell was based, not on the Hindley-Milner type system, but on a *different* ML type system?

Proposal

- Base Haskell on a core dependently-typed language with $\star : \star$

$$\begin{array}{lllll} \textit{terms},\ \textit{types} & a, b, A, B & ::= & \star \mid x \mid \lambda x\!:\!A.b \mid a\ b \\ & & \mid & \Pi x\!:\!A.B \end{array}$$

- Full-spectrum dependently typed language with a single sort
- Proposed by Martin Löf (1971 draft paper)
- **Not logically consistent**
- Not good for proof checking …but neither is Haskell
- Type checking is undecidable
- Type sound (Cardelli 1985) and *supremely* uniform
- Acknowledgements: Conor, Adam Gundry, Richard

$\star : \star$

- Subsumes higher-order polymorphism, type families, kind polymorphism, etc.

$$\frac{\vdash \Gamma}{\Gamma \vdash \star : \star}\ \text{B\_STAR} \qquad \frac{\begin{array}{l}\vdash \Gamma \\ x : A \in \Gamma\end{array}}{\Gamma \vdash x : A}\ \text{B\_VAR}$$

$$\frac{\begin{array}{l}\Gamma, x : A \vdash B : \star \\ \Gamma \vdash A : \star\end{array}}{\Gamma \vdash \Pi x{:}A.B : \star}\ \text{B\_PI} \qquad \frac{\begin{array}{l}\Gamma \vdash A : \star \\ \Gamma, x : A \vdash a : B\end{array}}{\Gamma \vdash \lambda x{:}A.a : \Pi x{:}A.B}\ \text{B\_ABS}$$

$$\frac{\begin{array}{l}\Gamma \vdash b : \Pi x{:}A.B \\ \Gamma \vdash a : A\end{array}}{\Gamma \vdash b\ a : B\{a/x\}}\ \text{B\_APP} \qquad \frac{\begin{array}{l}\Gamma \vdash a : A \\ \Gamma \vdash B : \star \\ A \equiv B\end{array}}{\Gamma \vdash a : B}\ \text{B\_CONV}$$

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 *propositional equality*: the ability of the type system to assume equality

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

f :: forall a. T a -> a
f TInt = 0 + 0
```

- Can we just encode propositional equality?

$$a \sim b \triangleq \Pi c :(\star \to \star).\ c\ a \to c\ b$$

- No!
  - Logic is inconsistent --- need to run proofs
  - Type inference decides where coercions are placed in terms.
  -
```
f :: forall a. T a -> a
f (TInt @c) = ((0 + 0) ▷ c)

f (TInt @c) = (+ ▷ (Int -> Int -> c)) 0 0
```

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 `Int ~ Bool`
- Elaboration requires irrelevant type coercion; it cannot matter how we use propositional equality

- *So (a ~ 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

$$
\begin{array}{llcl}
\textit{terms},\ \textit{types} & a, b, A, B & ::= & \star \mid x \mid \lambda x\!:\!A.b \mid a\ b \\
 & & \mid & \Pi x\!:\!A.B \\
 & & \mid & \Lambda c\!:\!\phi.a \mid a[\gamma] \mid \forall c\!:\!\phi.A \\
 & & \mid & a \rhd \gamma \\
\textit{propositions} & \phi & ::= & a \sim_A b \\
\textit{coercions} & \gamma & ::= & \ldots
\end{array}
$$

- Coercions are proof witnesses of equality between terms

$$\boxed{\Gamma ; \Delta \vdash \gamma : a \sim b}$$

Coercion abstraction

$$\frac{\Gamma \vdash \phi \ \mathsf{ok} \qquad \Gamma, c:\phi \vdash B : \star}{\Gamma \vdash \forall c{:}\phi.B : \star}\ \textsc{An\_CPi}$$

$$\frac{\Gamma \vdash \phi \ \mathsf{ok} \qquad \Gamma, c:\phi \vdash a : B}{\Gamma \vdash \Lambda c{:}\phi.a : \forall c{:}\phi.B}\ \textsc{An\_CAbs}$$

$$\frac{\Gamma \vdash b : \forall c{:}a_1 \sim_{A_1} b_1.B \qquad \Gamma; \mathsf{dom}(\Gamma) \vdash \gamma : a_1 \sim b_1}{\Gamma \vdash b[\gamma] : B\{\gamma/c\}}\ \textsc{An\_CApp}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma; \mathsf{dom}(\Gamma) \vdash \gamma : A \sim B \qquad \Gamma \vdash B : \star}{\Gamma \vdash a \rhd \gamma : B}\ \textsc{An\_Conv}$$

Coercion proofs

$$\boxed{\Gamma; \Delta \vdash \gamma : a \sim b}$$

- 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?

$$\boxed{\Gamma; \Delta \vdash \gamma : a \sim b}$$

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

$$\frac{\begin{array}{l} \Gamma; \Delta \vdash \gamma_1 : a_1 \sim b_1 \\ \Gamma; \Delta \vdash \gamma_2 : a_2 \sim b_2 \\ \Gamma \vdash a_1\ a_2 : A \\ \Gamma \vdash b_1\ b_2 : B \end{array}}{\Gamma; \Delta \vdash \gamma_1\ \gamma_2 : a_1\ a_2 \sim b_1\ b_2}\ \textsc{An\_AppCong}$$

Consistency

- Progress lemma requires consistency

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

- Proof based on confluence of parallel reduction

Definition 2 (Joinable).

$$\frac{\vdash a_1 \Rightarrow^* b \qquad \vdash a_2 \Rightarrow^* b}{\vdash a_1 \Leftrightarrow a_2}\ \text{JOIN}$$

Theorem 3 (Joinability implies consistency). *If* $\vdash A \Leftrightarrow B$ *then* $\mathbf{consistent}\ A\ B$.

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

A Difficulty

- Consider this equality

$$\forall c\!:\!(\mathbf{Int} \sim_\star \mathbf{Bool}).\mathbf{Int} \equiv \forall c\!:\!(\mathbf{Int} \sim_\star \mathbf{Bool}).\mathbf{Bool}$$

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

$$\frac{\begin{array}{l}\vdash \Gamma \\ c : a \sim_A b \in \Gamma \\ c \in \Delta\end{array}}{\Gamma; \Delta \vdash c : a \sim b}\ \textsc{An\_Assn}$$

Equality for cpi

$$\frac{\begin{array}{l}\Gamma; \Delta \vdash \gamma_1 : \phi_1 \equiv \phi_2 \\ \Gamma, c : \phi_1; \Delta \vdash \gamma_3 : B_1 \sim (B_2\{c/c\}) \\ B_3 = B_2\{c \triangleright \mathbf{sym}\, \gamma_1 / c\} \\ \Gamma \vdash \forall c\!:\!\phi_1 . B_1 : \star \\ \Gamma \vdash \forall c\!:\!\phi_2 . B_3 : \star\end{array}}{\Gamma; \Delta \vdash (\forall c\!:\!\gamma_1 . \gamma_3) : (\forall c\!:\!\phi_1 . B_1) \sim (\forall c\!:\!\phi_2 . B_3)} \text{AN\_CPICONG}$$

$$\frac{\begin{array}{l}\Gamma; \Delta \vDash \phi_1 \equiv \phi_2 \\ \Gamma, c : \phi_1; \Delta \vDash A \equiv B : \star\end{array}}{\Gamma; \Delta \vDash \forall c\!:\!\phi_1 . A \equiv \forall c\!:\!\phi_2 . B : \star} \text{E\_CPICONG}$$

Implicit Dependent FC

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

$$
\begin{array}{llcl}
\textit{terms, types} & a, b, A, B & ::= & \star \mid x \mid \lambda x.b \mid a\ b \\
 & & \mid & \Pi x\!:\!A.B \\
 & & \mid & \Lambda c.a \mid a[\gamma] \mid \forall c\!:\!\phi.A \\
\textit{propositions} & \phi & ::= & a \sim_A b \\
\textit{coercions} & \gamma & ::= & \bullet
\end{array}
$$

- Coercion replaced by definitional equality between types

$$
\frac{\Gamma \vDash a : A \qquad \Gamma; \mathsf{dom}(\Gamma) \vDash A \equiv B : \star}{\Gamma \vDash a : B}\ \textsc{E\_Conv}
$$

Erasure & Annotation

LEMMA 5 (Erasure).
- *If* $\Gamma \vdash a : A$ *then* $|\Gamma| \vDash |a| : |A|$
- *If* $\Gamma;\Delta \vdash \gamma : a \sim b$ *and* $\Gamma \vdash a : A$, *then* $|\Gamma|;\Delta \vDash |a| \equiv |b| : |A|$

LEMMA 6 (Annotation).
- *If* $\Gamma \vdash a : A$, *then for all* Γ_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* Γ_0 *such that* $|\Gamma_0| = \Gamma$, *there exists* γ, 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)