A DEPENDENTLY-TYPED CORE CALCULUS FOR GHC

STEPHANIE WEIRICH
UNIVERSITY OF PENNSYLVANIA
PHILADELPHIA, USA
Goals

- Promote dependently-typed programming with the Glasgow Haskell Compiler (GHC)
- Prove type-system extensions sound using Coq proof assistant
COLLABORATORS

- Richard Eisenberg
- Antoine Voizard
- Pritam Choudhury
- Pedro Henrique Avezedo de Amorim
- Anastasiya Kravchuk-Kirilyuk
- Joachim Breitner
- Simon Peyton Jones
WHAT IS DIFFERENT ABOUT DEPENDENT TYPES IN GHC?

- **Not starting from scratch** existing compiler, user base and ecosystem
- **Programs** (and types) may not terminate
- **Type soundness** instead of logical consistency
A set of language extensions for GHC that provides the ability to program as if the language had dependent types

{-# LANGUAGE DataKinds, TypeFamilies, PolyKinds, TypeInType, GADTs, RankNTypes, ScopedTypeVariables, TypeApplications, UndecidableInstances, InstanceSigs, TypeSynonymInstances, TypeOperators, KindSignatures, MultiParamTypeClasses, FunctionalDependencies, TypeFamilyDependencies, AllowAmbiguousTypes, FlexibleContexts, FlexibleInstances #-}
data Nat = Zero | Succ Nat

data Fin (n :: Nat) where
    Z :: Fin (Succ n)
    S :: Fin n -> Fin (Succ n)

data Vec :: Nat -> Type -> Type where
    Nil :: Vec Zero a
    Cons ::
        a -> Vec n a -> Vec (Succ n) a

idx :: Fin n -> Vec n a -> a
idx Z    (Cons x xs) = x
idx (S m) (Cons x xs) = idx m xs
MAJOR CHALLENGES

- Singletons (no $\Pi$ type)
- Lack of uniformity (type-level computation is different than run-time computation)
- Weak logic (can't prove much at compile-time)
MAJOR CHALLENGES

- Singletons (no Π type)
- Lack of uniformity (type-level computation is different than run-time computation)
- Weak logic (can't prove much at compile-time)
I. SINGLETONS

vrepl :: Π(n :: Nat) -> Bool -> Vec n Bool

vrepl Zero _ = Nil

vrepl (Succ n) x = Cons x (vrepl n x)
I. SINGLETONS

vrepl :: SN(n :: Nat) -> Bool -> Vec n Bool
vrepl SZero _ = Nil
vrepl (SSucc n) x = Cons x (vrepl n x)

data SN (n :: Nat) where
  SZero :: SN Zero
  SSucc :: SN n -> SN (Succ n)
2. LACK OF UNIFORMITY

\[
vrepl :: \text{SN}(n :: \text{Nat}) \rightarrow \text{Bool} \rightarrow \text{Vec} \ n \ \text{Bool}
\]

\[
vrepl \ \text{SZero} \ _ = \text{Nil}
\]

\[
vrepl \ (\text{SSucc} \ n) \ x = \text{Cons} \ x \ (vrepl \ n \ x)
\]

\[
\text{type family} \ \text{Vrepl} \ (n :: \text{Nat}) \ (x :: a) :: \text{Vec} \ a \ n
\]

\[
\text{where}
\]

\[
\text{Vrepl} \ \text{Zero} \ \ x = \text{Nil}
\]

\[
\text{Vrepl} \ (\text{Succ} \ n) \ x = \text{Cons} \ x \ (\text{Vrepl} \ n \ x)
\]
PLAN

- Extend GHC's **Core intermediate language** with dependent types
- **Skip hard stuff** namespace issues, type inference, pattern match compilation
- **Base design on a mathematical model of Core** aka System FC
FROM FC TO DC

FC: System F with type equality coercions
(and datatype promotion,
and type-in-type, and…)
[Sulzmann et al. 07,
Yorgey et al. 12, Weirich et al. 14]

DC: Dependently-typed calculus with type equality coercions
[Gundry14, Eisenberg16, WVAE17]
SYSTEM FC – TERM LEVEL COMPUTATION

types, kinds
\[ A, B, K ::= \star | y | A \rightarrow B | \forall y : K.A | \forall c : \phi.A \]
\[ | T | A \cdot B | A[\gamma] | A \triangleright \gamma \]

terms
\[ a, b ::= x | \lambda x : A.a | a \cdot b | \lambda y : K.a | a \cdot A \]
\[ | \Lambda c : \phi.a | a[\gamma] \]
\[ | T | a \triangleright \gamma \]

equality constraints
\[ \phi ::= A \sim B \]

coercion proofs
\[ \gamma ::= \ldots \]

1. Constants
2. Normal functions
3. Polymorphism
4. Equality coercions
5. Coercion abstraction
SYSTEM FC – TYPE LEVEL COMPUTATION

**types, kinds**

\[ A, B, K ::= \ast \mid y \mid A \rightarrow B \mid \forall y : K . A \mid \forall c : \phi . A \mid T \mid A B \mid A[\gamma] \mid A \triangleright \gamma \]

**terms**

\[ a, b ::= x \mid \lambda x : A . a \mid a \; b \mid \lambda y : K . a \mid a \; A \mid \Lambda c : \phi . a \mid a[\gamma] \mid T \mid a \triangleright \gamma \]

**equality constraints**

\[ \phi ::= A \sim B \]

**coercion proofs**

\[ \gamma ::= \ldots \]

1. Constants & definitions
2. Normal functions
3. Dependent functions
4. Equality coercions
5. Coercion abstraction
terms, types, kinds  $a, b, A, B, K$  ::=  $\star | x | \lambda x : A.a | a \; b | \Pi x : A.B \lambda^- x : A.a | a \; A^- | \forall x : K.A \Lambda c : \phi.a | a[\gamma] | \forall c : \phi.A T | a \triangleright \gamma$

equality constraints  $\phi$  ::=  $A \sim B$

coercion proofs  $\gamma$  ::=  $\ldots$

1. Constants & definitions
2. Dependent functions
3. Irrelevant abstraction
4. Equality coercions
5. Coercion abstraction
<table>
<thead>
<tr>
<th>Ceci n'est pas un poulet</th>
</tr>
</thead>
<tbody>
<tr>
<td>General recursion</td>
</tr>
<tr>
<td>Type-in-Type</td>
</tr>
<tr>
<td>Coercions and coercion abstraction</td>
</tr>
<tr>
<td>Irrelevant abstraction</td>
</tr>
</tbody>
</table>
COERCIONS NOT CONVERSION

- Proof justifies type equality
- Even $\beta$-equality requires justification (cf. Weak Type Theory)
- Explicit use of coercions enables decidable type checking in GHC
Irrelevant variables must not appear in relevant parts of the term [Barras & Bernardo 2008]

Erasure operation removes annotations, irr. arguments and coercion proofs

\[
\Gamma, x : A \vdash a : B \\
\text{\textcolor{blue}{\text{\textit{x} \not\in \text{fv}|a|}}} \quad \Gamma \vdash b : \forall x : A. B \\
\Gamma \vdash b \textcolor{red}{a} : B\{a/x\}
\]

\[
\Gamma \vdash a : A \\
\Gamma \vdash \lambda x : A. a : \forall x : A. B
\]
PROBLEM

- FC and DC are complicated type systems
PROBLEM: SYSTEM DC IS COMPLICATED

- Is the design correct?
- Is it type sound: Progress & Preservation
- Can types & coercions be erased?
- Is type checking decidable?
- Do design choices matter?
- Changing expressiveness…
- … or pushing annotations around?
SOLUTION: PART I, DROP DECIDABLE TYPE CHECKING

- Coercions and type annotations only present in terms to provide decidable type checking
- Connect to erased language (System D) with Curry-style type system
- Languages are equivalent via erasure and annotation
TWO RELATED LANGUAGES

- Curry style vs. Church style type systems
- Definitional equality in D is coercion checking in DC
- DC has decidable type checking, D does not
- Progress lemma for D implies progress for DC
- Preservation for DC implies preservation for D

For simplicity

\[
\begin{align*}
D & \quad \Gamma \vdash a : A \\
\Gamma & \vdash \phi \text{ ok} \\
\Gamma; \Delta & \vdash a \equiv b : A \\
\Gamma; \Delta & \vdash \phi_1 \equiv \phi_2 \\
\Gamma & \vdash \Gamma
\end{align*}
\]

For GHC

\[
\begin{align*}
DC & \quad \Gamma \vdash a : A \\
\Gamma & \vdash \phi \text{ ok} \\
\Gamma; \Delta & \vdash \gamma : a \sim b \\
\Gamma; \Delta & \vdash \gamma : \phi_1 \sim \phi_2 \\
\Gamma & \vdash \Gamma
\end{align*}
\]
COMPLEXITY SOLUTION, PART 2: MECHANIZATION

- All results proven in Coq
  - Type safety
  - Erasure & annotation theorems
  - Decidable type checking for DC

- Large development
  - Spec: 1,400 lines, Proof: 17k loc

- Tool support: essential
  - Ott [Sewell et al. 2007] & Ingen [Aydemir 2010]

The Coq Proof Assistant
https://github.com/sweirich/corespec/
FORMALIZATION IN COQ

<table>
<thead>
<tr>
<th></th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ott spec</td>
<td>1423</td>
</tr>
<tr>
<td>LaTeX macros</td>
<td>1851</td>
</tr>
<tr>
<td>Paper sources</td>
<td>2317</td>
</tr>
<tr>
<td>Coq</td>
<td>32828</td>
</tr>
<tr>
<td>Language def</td>
<td>1432</td>
</tr>
<tr>
<td>Syntactic lemmas</td>
<td>11730</td>
</tr>
<tr>
<td>System D</td>
<td>5399</td>
</tr>
<tr>
<td>Consistency</td>
<td>2417</td>
</tr>
<tr>
<td>System DC</td>
<td>8142</td>
</tr>
<tr>
<td>Decidability</td>
<td>3529</td>
</tr>
<tr>
<td>Connection</td>
<td>2215</td>
</tr>
<tr>
<td>Utils</td>
<td>629</td>
</tr>
<tr>
<td>Other</td>
<td>2732</td>
</tr>
</tbody>
</table>
LOCALLY NAMELESS REPRESENTATION

\[ \Gamma, x : A \vdash B : \text{TYPE} \]
\[ \Gamma \vdash A : \text{TYPE} \]
\[
\begin{array}{c}
\Gamma \vdash \Pi^\rho x : A \rightarrow B : \text{TYPE} \\
\hline
\text{E}_\Pi
\end{array}
\]

\[ \text{E}_\Pi : \]
\[ \text{forall } (L : \text{vars}) (G : \text{context}) (\rho : \text{relflag}) (A B : \text{tm}), \]
\[ \text{(forall } x, x \notin L \text{ ->)} \]
\[ \text{Typing } ((x \sim \text{Tm } A) \text{ ++ } G) \]
\[ \quad \text{(open_tm_wrt_tm } B (\text{a_Var_f } x)) \text{ a_Star) } \]
\[ \text{-} \text{Typing } G A \text{ a_Star} \]
\[ \text{-} \text{Typing } G (\text{a_Pi } \rho A B) \text{ a_Star} \]
EXTENSIONS

ETA-EQUIVALENCE
[COQPL 2018]
JOINT WORK WITH
ANASTASIYA
KRAVCHUK-KIRILYUK

SAFE COERCIONS
[ICFP 2019]
JOINT WORK WITH
PRITAM CHOUDHURY,
ANTOINE VOIZARD,
RICHARD EISENBERG
Add new coercion forms (DC) and equivalence rules (D)

- Extend all proofs

\[
\begin{align*}
\Gamma \vdash b : \Pi x : A. B \\
\Gamma; \Delta \vdash \text{eta } b : (\lambda x : A. b \; x) \sim b
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash b : \forall x : A. B \\
\Gamma; \Delta \vdash \text{eta } b : (\lambda^- x : A. b \; x^-) \sim b
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash b : \forall c : \phi. B \\
\Gamma; \Delta \vdash \text{eta } b : (\Lambda c : \phi. b[c]) \sim b
\end{align*}
\]
Progress lemma for D requires consistency of definitional equality \( (\Gamma; \Delta \models a \equiv b : A) \)

i.e. we don't equate terms/types with different head forms

Consistency proof based on confluence of parallel reduction (cf. Tait / Martin-Löf proof for \( \beta \eta \)-reduction for untyped lambda calculus)
Good news: Coq points out new required cases in existing confluence proof
Not so good news: Need induction on height of term, not structure
  - Height function automatically defined by Ingen
  - Omega tactic handles arithmetic
Not not-good news:
  - Parallel eta-reduction rules don't always preserve typeability
  - But consistency proof doesn't need them to
GHC includes zero-cost coercions for newtypes

```haskell
newtype Html = MkHtml String

unpackList :: [Html] -> [String]
unpackList = coerce
```

Must be careful with respect to safety and abstraction

```haskell
coerce :: Set Html -> Set String

coerce :: F Html -> F String
when F defined via intensional-type-analysis
GHC SOLUTION: ROLES [WZVPJ11, BEPJW14]

- Type system has different equalities
  - *nominal* — Normal Haskell
  - *representation*al — Types with equal representation
- *Role* annotations on type constructors determine congruence for representational equality
  - Set/F: arguments *must* be nominally equal when coercing
  - List: arguments may be representationally equal
DEPENDENT TYPES AND ROLES

- Extension of System D with two different equalities
  - nominal — Normal Haskell
  - representational — Types with equal representation
- Significantly larger system (100+ rules) built on existing Coq proofs
  - Intensional type analysis to model type families
  - Separate type checking & role checking judgements
CONCLUSIONS & FUTURE WORK

- Add GHC to the list of dependently-typed languages (at least at the type-level)
- Mechanized metatheory important at this scale
  - Collaboration tool
  - Starting point for extension
- Still many problems to overcome
  - Type inference, namespace management, etc.
  - More expressive proof theory