A DEPENDENTLY-TYPED CORE CALCULUS FOR GHC

STEPHANIE WEIRICH

UNIVERSITY OF PENNSYLVANIA

PHILADELPHIA, USA

Π

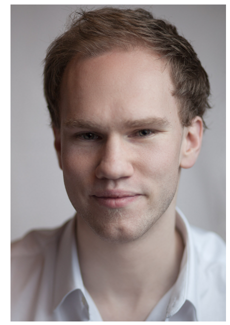
DEPENDENT HASKELL PROJECT

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

CURRENT STATUS

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
    #-}
```

(MANDATORY)

EXAMPLE

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


type family Vrepl (n :: Nat) (x :: a) :: Vec a n
   where
      Vrepl Zero     x = Nil
      Vrepl (Succ n) x = Cons x (Vrepl n x)
```

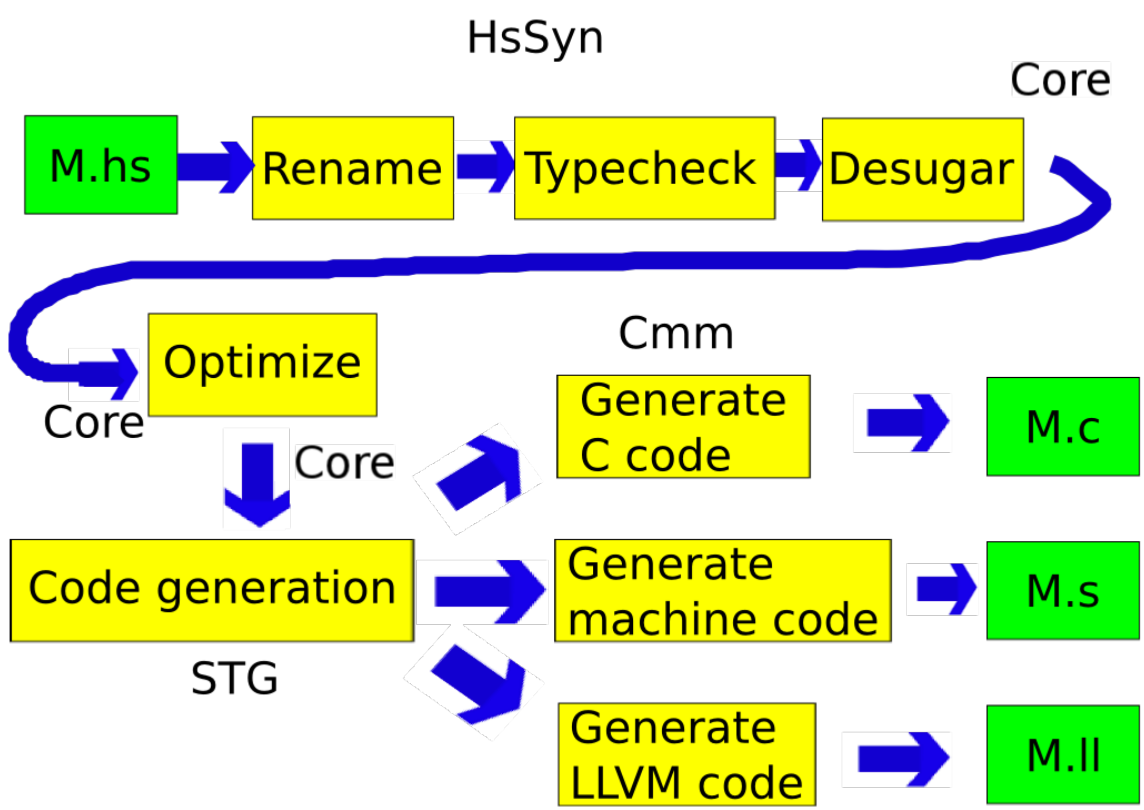
[ICFP 2017]

WITH ANTOINE VOIZARD,
PEDRO AMORIM AND
RICHARD EISENBERG

A DEPENDENTLY-TYPED CORE LANGUAGE FOR GHC

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	$::=$	$\star \mid y \mid A \to B \mid \forall y{:}K.A \mid \forall c{:}\phi.A$
		$\mid$	$T \mid A\,B \mid A[\gamma] \mid A \triangleright \gamma$
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	ϕ	$::=$	$A \sim B$
coercion proofs	γ	$::=$	$\ldots$

1. Constants
2. Normal functions
3. Polymorphism
4. Equality coercions
5. Coercion abstraction

SYSTEM FC – TYPE LEVEL COMPUTATION

types, kinds	A, B, K	$::=$	$\star \mid y \mid A \rightarrow B \mid \forall y{:}K.A \mid \forall c{:}\phi.A$
		$\mid$	$T \mid A\,B \mid A[\gamma] \mid A \triangleright \gamma$
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	ϕ	$::=$	$A \sim B$
coercion proofs	γ	$::=$	$\ldots$

1. Constants & definitions
2. Normal functions
3. Dependent functions
4. Equality coercions
5. Coercion abstraction

SYSTEM DC - COMBINED

$$
\begin{array}{llcl}
\textit{terms, types, kinds} & a, b, A, B, K & ::= & \star \mid x \mid \lambda x : A.a \mid a\ b \mid \Pi x : A.B \\
 & & \mid & \lambda^- x : A.a \mid a\ A^- \mid \forall x : K.A \\
 & & \mid & \Lambda c : \phi.a \mid a[\gamma] \mid \forall c : \phi.A \\
 & & \mid & T \mid a \rhd \gamma
\end{array}
$$

$$
\begin{array}{llcl}
\textit{equality constraints} & \phi & ::= & A \sim B \\
\textit{coercion proofs} & \gamma & ::= & \ldots
\end{array}
$$

1. Constants & definitions
2. Dependent functions
3. Irrelevant abstraction
4. Equality coercions
5. Coercion abstraction

Ceci n'est pas un poulet

General recursion

Type-in-Type

Coercions and coercion abstraction

Irrelevant abstraction

COERCIONS NOT CONVERSION

$$\frac{\Gamma \vdash a : A \quad A \equiv B \quad \Gamma \vdash B : \star}{\Gamma \vdash a : B}$$

$$\frac{\Gamma \vdash a : A \quad \Gamma ; \widetilde{\Gamma} \vdash \gamma : A \sim B \quad \Gamma \vdash B : \star}{\Gamma \vdash a \rhd \gamma : B}$$

- Proof justifies type equality
- Even β-equality requires justification (cf. Weak Type Theory)
- Explicit use of coercions enables decidable type checking in GHC

IRRELEVANT ABSTRACTION

- Irrelevant variables must not appear in *relevant* parts of the term [Barras & Bernardo 2008]
- Erasure operation removes annotations, irr. arguments and coercion proofs

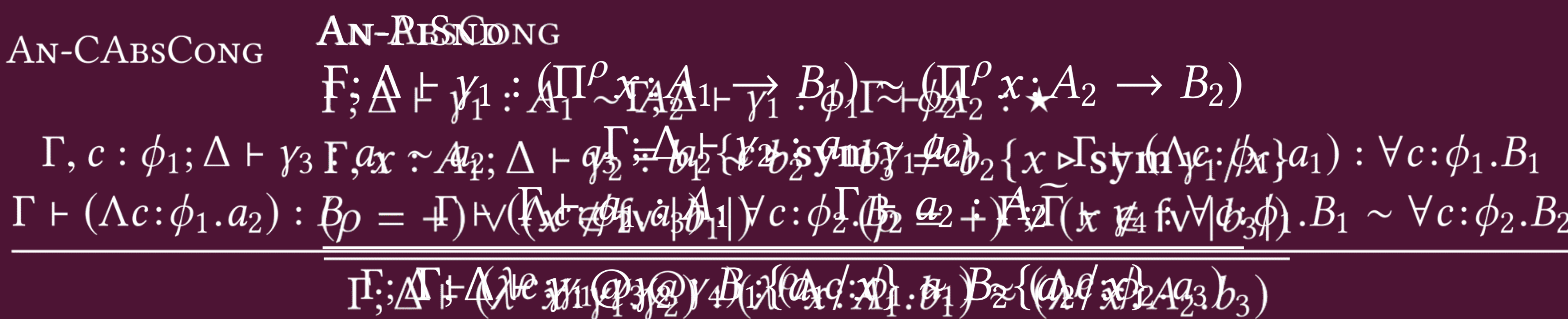
$$\frac{\Gamma, x : A \vdash a : B \qquad x \notin \mathsf{fv}|a|}{\Gamma \vdash \lambda^{-} x : A . a : \forall x : A . B}$$

$$\frac{\Gamma \vdash b : \forall x : A . B \qquad \Gamma \vdash a : A}{\Gamma \vdash b\ a^{-} : B\{a/x\}}$$

MANAGING COMPLEXITY

PROBLEM

- FC and DC are *complicated type systems*

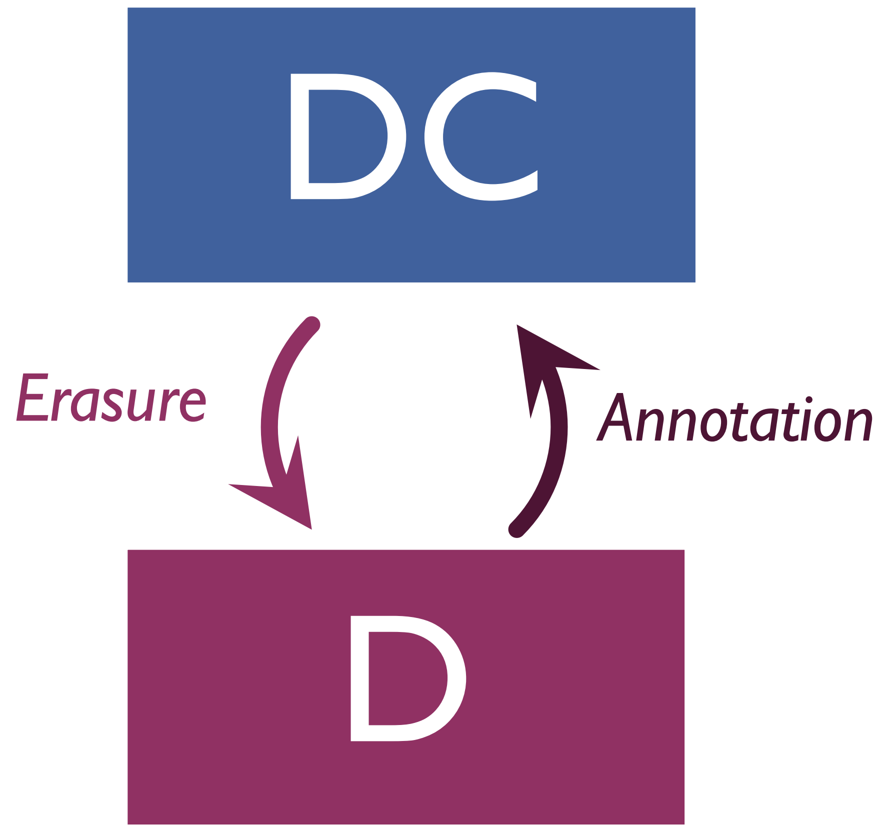


PROBLEM: SYSTEM DC IS COMPLICATED

- 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

For GHC

D	DC
$\Gamma \vDash a : A$	$\Gamma \vdash a : A$
$\Gamma \vDash \phi$ ok	$\Gamma \vdash \phi$ ok
$\Gamma; \Delta \vDash a \equiv b : A$	$\Gamma; \Delta \vdash \gamma : a \sim b$
$\Gamma; \Delta \vDash \phi_1 \equiv \phi_2$	$\Gamma; \Delta \vdash \gamma : \phi_1 \sim \phi_2$
$\vDash \Gamma$	$\vdash \Gamma$

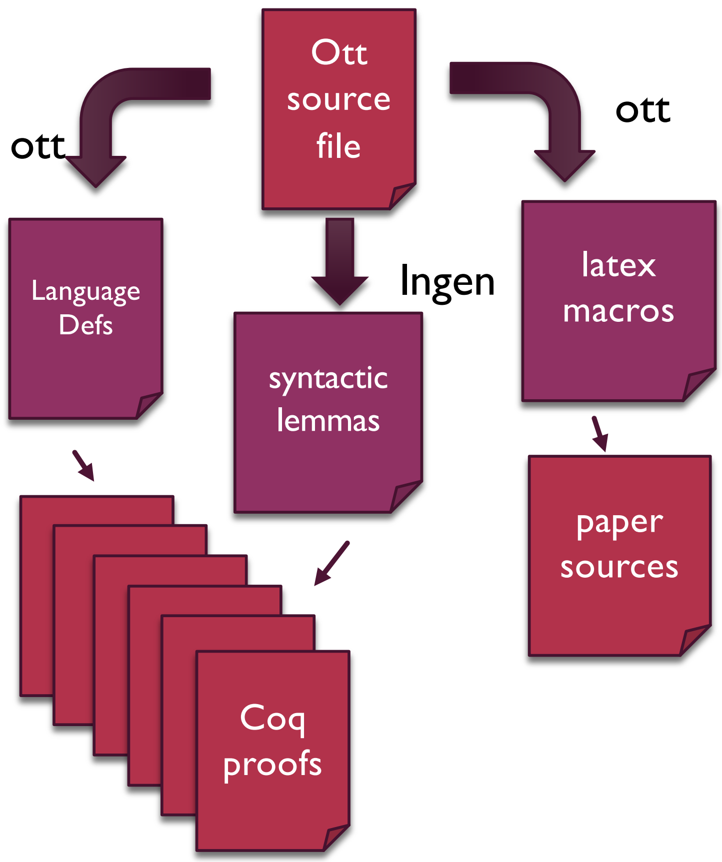
COMPLEXITY SOLUTION, PART 2: MECHANIZATION

- 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



	LOC
Ott spec	1423
LaTeX macros	1851
Paper sources	2317
Coq	32828
Language def	1432
Syntactic lemmas	11730
System D	5399
Consistency	2417
System DC	8142
Decidability	3529
Connection	2215
Utils	629
Other	2732

LOCALLY NAMELESS REPRESENTATION

```
   G, x:A |= B : TYPE
   G |= A : TYPE
------------------------------- :: Pi
  G |= Pi rho x:A -> B : TYPE
```

$$\frac{\Gamma, x : A \vDash B : \star \qquad \Gamma \vDash A : \star}{\Gamma \vDash \Pi^{\rho} x{:}A \to B : \star} \ \textsc{E\_Pi}$$

```
E_Pi :
forall (L:vars) (G:context) (rho:relflag)(A B:tm),
 (forall x , x \notin L ->
    Typing ((x ~ Tm A) ++ G)
             (open_tm_wrt_tm B (a_Var_f x)) a_Star)
-> Typing G A a_Star
-> Typing G (a_Pi rho A B) a_Star
```

ETA-EQUIVALENCE
[COQPL 2018]

JOINT WORK WITH ANASTASIYA KRAVCHUK-KIRILYUK

SAFE COERCIONS
[ICFP 2019]

JOINT WORK WITH PRITAM CHOUDHURY, ANTOINE VOIZARD, RICHARD EISENBERG

EXTENSIONS

ETA-EQUIVALENCE

- Add new coercion forms (DC) and equivalence rules (D)
- Extend all proofs

$$\frac{\Gamma \vdash b : \Pi x\!:\!A.B}{\Gamma;\Delta \vdash \mathbf{eta}\, b : (\lambda x\!:\!A.b\ x) \sim b}$$

$$\frac{\Gamma \vdash b : \forall x\!:\!A.B}{\Gamma;\Delta \vdash \mathbf{eta}\, b : (\lambda^{-} x\!:\!A.b\ \ x^{-}) \sim b}$$

$$\frac{\Gamma \vdash b : \forall c\!:\!\phi.B}{\Gamma;\Delta \vdash \mathbf{eta}\, b : (\Lambda c\!:\!\phi.b[c]) \sim b}$$

CONSISTENCY

- Progress lemma for D requires consistency of definitional equality $(\Gamma; \Delta \vDash a \equiv b : A)$

  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)

PROOF ENGINEERING

- 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

SAFE COERCIONS [ICFP 2019]

- GHC includes zero-cost coercions for newtypes

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

- Must be careful with respect to safety and abstraction

```
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
  - *representational* --- 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