# Final Report of Dependently Typed Core Replacement in GHC

Ningning Xie

August 12, 2018

## 1 Introduction

It was great to have our proposal accepted in Google Summer of Code this year. I have been working entirely on this project for this summer. In this report, I will 1) present the current state of the project; 2) describe the changes I have done; 3) summarize the challenges I have faced, and what I have learned from this summer; 4) conclude and discuss about future work.

## 2 Current State

We planed to implement dependently typed core. After intensive discussions, we adjusted the goal of this summer and focused on one phase of dependent Haskell.

To implement dependent typed core, we first need to embrace homogeneous equality, which means equality is between types of a same kind. Homogeneous equality simplifies meta-theory. More importantly, it enables us to prove the important lemma, *congruence*, for the dependently typed core. Adopting homogeneous equality is not straightforward. It requires us to make the type of primitive equality, `~#`, homogeneous, and requires patches to the constraint solver.

To be able to have homogeneous equality, we implemented coercion quantification. Adding coercion quantification means now polymorphic quantifications (over both types and coercions) could have a coercion in their bindings. Refactor of those basic types has a significant impact to files in the compilation pipeline and introduces several subtleties involving binders, substitutions, representations of datatypes, etc. And coercion quantification also opens up new questions to the design space in *source* Haskell.

## 3 Contributions

In this summer, I have made the following contributions: 1) Definition of *MCoercion*; 2) Refactor of coercions; 3) Literature review of GHC Core. 3) Bug fixing about coercion optimization. 5) Implementation of coercion quantification.

In the rest of this section, I will discuss them in detail.

### 3.1 Definition of MCoercion

Feature link: https://phabricator.haskell.org/D4699.

Status: Merged

Motivation: During compilation, reflexive casts is discarded for computation. Currently in some places we use Maybe coercion as inputs. So if a cast is reflexive it is denoted as Nothing, otherwise Just coercion.

This patch defines the type

```
data MCoercion = MRefl | MCo Coercion
```

which is isomorphic to `Maybe` Coercion but useful in a number of places, and super-helpful documentation.

### 3.2 Refactor of Coercions

Feature link: https://phabricator.haskell.org/D4747 (1000+ LOC)

Status: Merged

Motivation:

This patch aims at generalizing the reflexive coercion to

```
  | GRefl Role Type MCoercion
```

As its name suggests, and as its typing rule makes precise, GRefl is generalised version of Refl:

```
     t1 : k1
  ------------------------------------
  GRefl r t1 MRefl: t1 ~r t1

     t1 : k1       co :: k1 ~ k2
  ------------------------------------
  GRefl r t1 (MCo co) : t1 ~r t1 |> co
```

This new coercion form will replace both Refl and the current CoherenceCo:

```
  | Refl Role Type
  | CoherenceCo Coercion KindCoercion
```

## 3.3 Literature Review of GHC Core

Link: https://github.com/xnning/GHC-Core-Literature-Review

Status: As contributed documentation in yaskell.org. https://ghc.haskell.org/trac/ghc/wiki/Commentary

Motivation: This document aims at giving an rough overview of the evolution of GHC Core, System FC, through publications. The motivation of this document is to aid developers who hack into GHC Core in gaining a theoretical understanding of each design choice involved in the type system.

## 3.4 Bug Fixing in Coercion Optimization

Commit Link: https://phabricator.haskell.org/D5018

Status: Merged

Motivation: This patch fixed a bug in the current implementation of coercion optimization in GHC Core.

Given `co1;co2`, where

```
    co1 = \/ tv1 : eta1. r1
    co2 = \/ tv2 : eta2. r2
```

We would like optimize the transitivity coercion. Our wanted result is

```
\/tv1 : (eta1;eta2).  (r1; r2[tv2 |-> tv1 |> eta1])
```

## 3.5 Coercion Quantification

Feature Link: https://phabricator.haskell.org/D5054 (2500+ LOC )

Status: Under code revision.

Related talk accepted in Haskell Implementor's Workshop, co-colated with ICFP 18: https://icfp18.sigplan.org/track/hiw-2018-papers#event-overview

An overview of changes:

- Both ForAllTy and ForAllCo can quantify over coercion variables, but only in Core. All relevant functions are updated accordingly.
- Small changes that should be irrelevant to the main task:
    1. removed dead code mkTransAppCo in Coercion

2. removed out-dated Note Computing a coercion kind and roles in Coercion
3. Added Eq4 in Note Respecting definitional equality in TyCoRep, and updated mkCastTy accordingly.
4. Various updates and corrections of notes and typos.

- Haddock api needs to be changed too. An overview of the change can be found in here.

## 4 Challenges and Learnings

During this project, I have met two main challenges.

Firstly, GHC is a big compiler, and every time re-compilation takes several minutes up to one hour or so. It means I cannot see immediately the effects to the compiler after I made some changes, which makes progress slow. To solve the problem, I usually only stop at some important points to start re-compilation. GHC also has several tricks to accelerate the re-compilation, for example, `make 2` only re-compiles the second stage of GHC.

Secondly, unlike toy projects I have done in school, for which functionality is all you need, GHC is a compiler used in real-world, whose performance is thus very important. For example, for the commit described in Section 3.2, it failed performance tests when I first committed it. Later on we worked so hard to improve it, but still admitted one regression.

## 5 Conclusion

Dependent Haskell has been desired in the community of Haskell programmers for a long time, and we believe that our project is on the right track towards our ultimate goal.

I have learned a lot from this project, since this is the first time for me to work on a compiler that is being used in real life. Now not only do I have better Haskell skills, but I also have understood the infrastructure of a compiler better.

One of the future work is to fix the performance issue in Section 3.2, as mentioned in Section 4. Moreover, after coercion quantification, we plan to have homogeneous equality to prepare GHC for having real dependent Core. After that, we can start merging the syntax of expressions, types, coercions in Core. We believe that dependent core will set the stage for source-level dependent Haskell.