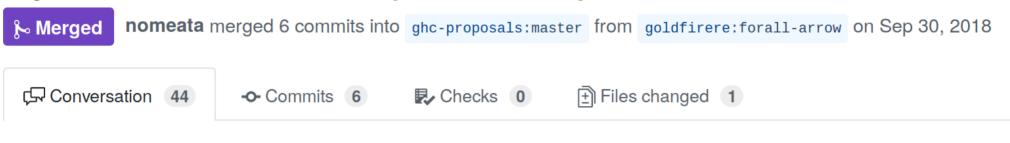
# Visible Dependent Quantification (VDQ)

**Ryan Scott** 

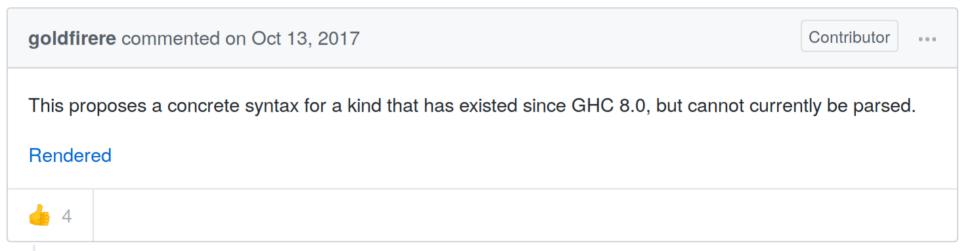
PL Wonks March 8, 2019

# Code time!

### Syntax for visible dependent quantification #81







#### Dependent types in Haskell: Progress Report

#### 12 Replies

It was drawn to my attention that there is an active <u>Reddit thread</u> about the future of dependent types in Haskell. (Thanks for the heads up, @thomie!) Instead of writing a long response inline in Reddit, it seems best to address the (very knowledgeable, respectful, and all around heartening) debate here.

#### When can we expect dependent types in GHC?

The short answer: GHC 8.4 (2018) at the very earliest. More likely 8.6 or 8.8 (2019-20).

#### Dependent types in Haskell: Progress Report

#### 12 Replies

It was drawn to my attention that there is an active <u>Reddit thread</u> about the future of dependent types in Haskell. (Thanks for the heads up, @thomie!) Instead of writing a long response inline in Reddit, it seems best to address the (very knowledgeable, respectful, and all around heartening) debate here.

#### When can we expect dependent types in GHC?

The short answer: GHC 8.4 (2018) at the very earliest. More likely 8.6 or 8.8 (2019-20).

#### GHC plans for 8.8.1

This page is our road-map for what will be in 8.8.

If you believe your favorite thing belongs in this list, but isn't there, please yell. If it's not in the road map, it probably won't get done. Without a lot of support, many things in the road map won't get done either, so we need your help!

#### **Dates**

- 18 November 2018: Cut release branch
- 25 November 2018: Release alpha1
- 16 December 2018: Release alpha2
- 6 January 2019: Release alpha3
- 27 January 2019: Release alpha4
- 17 February 2019: Release beta1
- 15 March 2019: Final release

#### GHC plans for 8.8.1

This page is our road-map for what will be in 8.8.

If you believe your favorite thing belongs in this list, but isn't there, please yell. If it's not in the road map, it probably won't get done. Without a lot of support, many things in the road map won't get done either, so we need your help!

#### **Dates**

- 18 November 2018: Cut release branch
- 25 November 2018: Release alpha1
- 16 December 2018: Release alpha2
- 6 January 2019: Release alpha3
- 27 January 2019: Release alpha4
- 17 February 2019: Release beta1
- 15 March 2019: Final release

Below are the major highlights of 8.8.

- A safer and more efficient with# combinator to control object lifetime (#14375)
- Improved compilation time for type-family-heavy programs (#8095, ⇒ Phab:D4766)
- More efficient code generation for nested closures (#14461)
- Next iteration of Trees That Grow (tickets/patches for this?)
- Continued focus on performance:
  - Some possible tickets: #15418, #15455, #14980, #14013, #15488, #15519, #1
  - New codelayout algorithm for the NCG: #15124
  - Optimize based on limited static analysis: #14672
- A late lambda lifting optimisation on STG (#9476)
- More locations where users can write forall: ⇒ https://github.com/ghc-proposals/

Below are the major highlights of 8.8.

- A safer and more efficient with# combinator to control object lifetime (#14375)
- Improved compilation time for type-family-heavy programs (#8095, ➡ Phab:D4766)
- More efficient code generation for nested closures (#14461)
- Next iteration of Trees That Grow (tickets/patches for this?)
- Continued focus on performance:
  - Some possible tickets: #15418, #15455, #14980, #14013, #15488, #15519, #1
  - New codelayout algorithm for the NCG: #15124
  - Optimize based on limited static analysis: #14672
- A late lambda lifting optimisation on STG (#9476)
- More locations where users can write forall: ⇒ https://github.com/ghc-proposals/

Below are the major highlights of 8.8.

- A safer and more efficient with# combinator to control object lifetime (#14375)
- Improved compilation time for type-family-heavy programs (#8095, ➡ Phab:D4766)
- More efficient code generation for nested closures (#14461)
- Next iteration of Trees That Grow (tickets/patches for this?)
- Continued focus on performance:
  - Some possible tickets: #15418, #15455, #14980, #14013, #15488, #15519, #1
  - New codelayout algorithm for the NCG: #15124
  - Optimize based on limited static analysis: #14672
- A late lambda lifting optimisation on STG (#9476)
- More locations where users can write forall: ⇒ https://github.com/ghc-proposals/

Below are the major highlights of 8.8.

- A safer and more efficient with# combinator to control object lifetime (#14375)
- Improved compilation time for type-family-heavy programs (#8095, ⇒ Phab:D4766)
- More efficient code generation for nested closures (#14461)
- Next iteration of Trees That Grow (tickets/patches for this?)
- Continued focus on performance:
  - Some possible tickets: #15418, #15455, #14980, #14013, #15488, #15519, #1
  - New codelayout algorithm for the NCG: #15124
  - Optimize based on limited static analysis: #14672
- A late lambda lifting optimisation on STG (#9476)
- More locations where users can write forall: ⇒ https://github.com/ghc-proposals/

# The VDQ patch

```
--- a/compiler/parser/Parser.y
+++ b/compiler/parser/Parser.y
+forall_vis_flag :: { ForallVisFlag }
+ : '.' { ForallInvis }
+ | '->' { ForallVis }
+
 -- A ctype is a for-all type
ctype :: { LHsType GhcPs }
        : 'forall' tv_bndrs '.' ctype
                                           {% ...
        : 'forall' tv_bndrs forall_vis_flag ctype {% ...
                 HsForAllTy { hst_bndrs = $2
                 HsForAllTy { hst_fvf = $3
                           , hst_bndrs = $2
```

```
--- a/compiler/typecheck/TcHsType.hs
+++ b/compiler/typecheck/TcHsType.hs
 ----- Foralls
-tc_hs_type forall@(HsForAllTy { ... })
+tc_hs_type forall@(HsForAllTy { hst_fvf = fvf, ... })
  = do { ...
        ; let bndrs
                          = mkTyVarBinders Specified tvs'
        ; let argf
                          = case fvf of
                              ForallVis -> Required
                              ForallInvis -> Specified
+
```

bndrs

= mkTyVarBinders argf tvs'

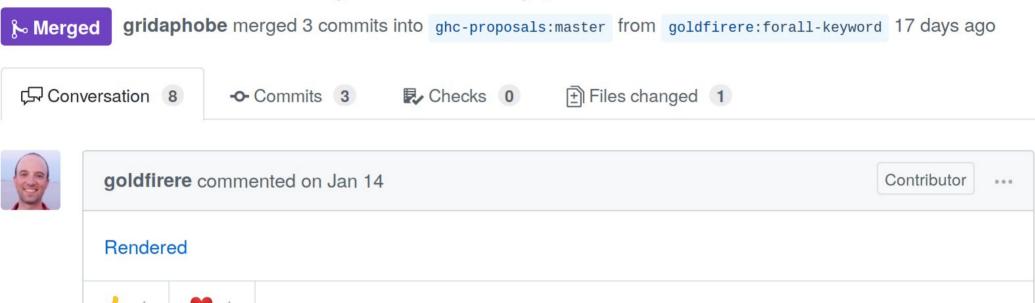
## forall k -> k -> Type

```
forall k -> k -> Type
```

{-# LANGUAGE ExplicitForAll #-}

# Code time! (again)

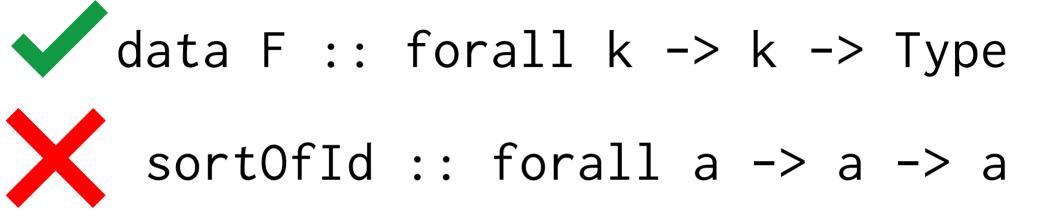
## Make `forall` a keyword in types #193



data F :: forall k -> k -> Type

data F :: forall k -> k -> Type

sortOfId :: forall a -> a -> a



vdqAllowed :: UserTypeCtxt -> Bool

```
vdqAllowed :: UserTypeCtxt -> Bool
-- Currently allowed in the kinds of types...
vdqAllowed (KindSigCtxt {}) = True
vdqAllowed (TySynCtxt {}) = True
```

```
vdqAllowed :: UserTypeCtxt -> Bool
-- Currently allowed in the kinds of types...
vdqAllowed (KindSigCtxt {}) = True
vdqAllowed (TySynCtxt {}) = True
-- ...but not in the types of terms.
vdqAllowed (FunSigCtxt {}) = False
vdqAllowed (InstDeclCtxt {}) = False
```





Glasgow Haskell Compiler > (9hc) GHC > Merge Requests > !378



Opened 2 weeks ago by Ryan Scott

Report abuse

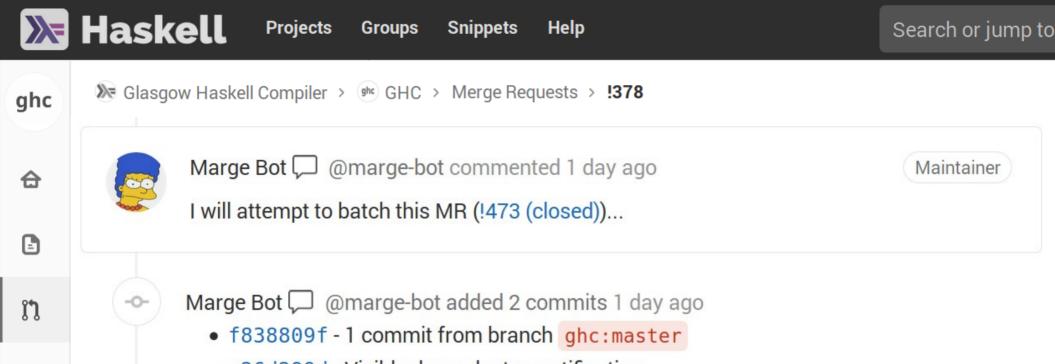


ij

ghc

## Visible dependent quantification

This implements GHC proposal 35 (https://github.com/ghc-proposals/ghc-proposals/blob/master /proposals/0035-forall-arrow.rst) by adding the ability to write kinds with visible dependent quantification (VDQ).



• c26d299d - Visible dependent quantification

Compare with previous version

Marge Bot @marge-bot merged 1 day ago

## **VDQ**

- An important step towards dependent types in Haskell
- Amaze your friends, impress your coworkers, wow!

## Debuts in GHC 8.10!