Visible Dependent Quantification

Ryan Scott
Indiana University

Haskell Implementors Workshop
23 August 2019
Berlin, Germany

VDQ

Ryan Scott
Indiana University

Haskell Implementors Workshop
23 August 2019
Berlin, Germany

The origins of VDQ

Syntax for visible dependent quantification #81

Merged **nomeata** merged 6 commits into `ghc-proposals:master` from `goldfirere:forall-arrow` on Sep 30, 2018

Conversation 44 Commits 6 Checks 0 Files changed 1

goldfirere commented on Oct 13, 2017 • edited by nomeata Contributor

The proposal has been accepted; the following discussion is mostly of historic interest.

This proposes a concrete syntax for a kind that has existed since GHC 8.0, but cannot currently be parsed.

Rendered

Syntax for visible dependent quantification #81

Merged nomeata merged 6 commits into `ghc-proposals:master` from `goldfirere:forall-arrow` on Sep 30, 2018

a kind that has existed since GHC 8.0, but cannot currently be parsed.

The proposal has been accepted; the following discussion is mostly of historic interest.

This proposes a concrete syntax for a kind that has existed since GHC 8.0, but cannot currently be parsed.

Rendered

```
data Bool where
  False :: Bool
  True  :: Bool
```

```
data Bool where
  False :: Bool
  True  :: Bool

data Bool :: * where
  ...
```

```
data Bool where
  False :: Bool
  True  :: Bool

data Bool :: Type where
  ...
```

```haskell
data Maybe a where
  Nothing :: Maybe a
  Just    :: a -> Maybe a
```

```haskell
data Maybe a where
  Nothing :: Maybe a
  Just    :: a -> Maybe a

data Maybe :: Type -> Type where
  ...
```

```
data Proxy (a :: k) where
  MkProxy :: Proxy a
```

```haskell
data Proxy (a :: k) where
  MkProxy :: Proxy a

data Proxy :: k -> Type where
  ...
```

```haskell
data Proxy (a :: k) where
  MkProxy :: Proxy a

data Proxy :: forall k. k -> Type where
  ...
```

```
data ProxyExp k (a :: k) where
  MkProxyExp :: ProxyExp k (a :: k)
```

```
data ProxyExp k (a :: k) where
  MkProxyExp :: ProxyExp k (a :: k)


data ProxyExp :: ??? where
  ...
```

New in GHC 8.0!

```haskell
data ProxyExp k (a :: k) where
  MkProxyExp :: ProxyExp k (a :: k)
```

New in GHC 8.0!

```
$ ghci
λ>
```

```haskell
data ProxyExp k (a :: k) where
  MkProxyExp :: ProxyExp k (a :: k)
```

New in GHC 8.0!

```
$ ghci
λ> :kind ProxyExp
```

```
data ProxyExp k (a :: k) where
  MkProxyExp :: ProxyExp k (a :: k)
```

New in GHC 8.0!

```
$ ghci
λ> :kind ProxyExp
ProxyExp :: forall k -> k -> Type
```

```haskell
data ProxyExp k (a :: k) where
  MkProxyExp :: ProxyExp k (a :: k)
```

New in GHC 8.0!

```
$ ghci
λ> :kind ProxyExp
ProxyExp :: forall k -> k -> Type
λ> data ProxyExp :: forall k -> k -> Type
```

```
data ProxyExp k (a :: k) where
  MkProxyExp :: ProxyExp k (a :: k)
```

New in GHC 8.0!

```
$ ghci
λ> :kind ProxyExp
ProxyExp :: forall k -> k -> Type
λ> data ProxyExp :: forall k -> k -> Type

<interactive>:11:27: error: parse error on
input ‘->’
```

What is this `forall k ->` thing, anyway?

Visible Dependent Quantification

Visible Dependent Quantification

Visible Dependent

The argument *must* be written in the source code.

Visible Dependent Quantification

The argument *must* be written in the source code.

```
  Maybe
:: Type -> Type
```

Visible Dependent

The argument *must* be written in the source code.

```
  Maybe
:: Type -> Type
  -----
Maybe Int
```

Visible Dependent Quantification

The argument *must* be written in the source code.

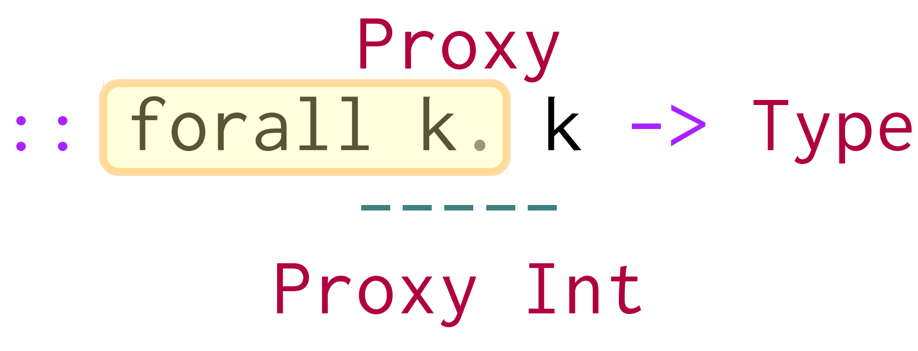
```
    Maybe
:: Type -> Type
    -----
  Maybe Int
```

```
         Proxy
:: forall k. k -> Type
```

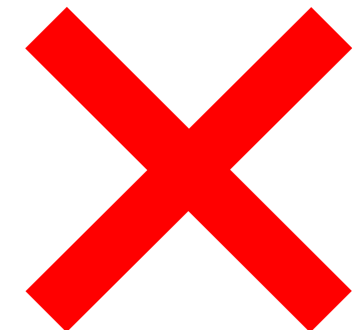
Visible Dependent Quantification

The argument *must* be written in the source code.

```
    Maybe
:: Type -> Type
   ------
  Maybe Int
```



```
       Proxy
:: forall k. k -> Type
       ------
     Proxy Int
```



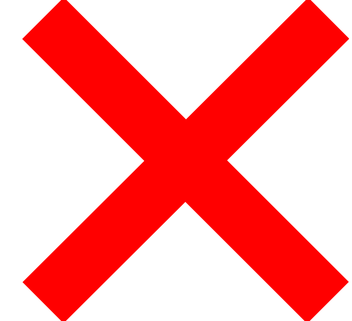
Visible Dependent

The argument *must* be written in the source code.

```
Maybe
:: Type -> Type
------
Maybe Int
```

```
Proxy
:: forall k. k -> Type
------
```

```
Proxy @Type Int
```



Visible Dependent Quantification

Visible **Dependent** Quantification

Visible Dependent Quantification

The body of the kind changes depending on its type.

Visible Dependent

The body of the kind changes depending on its type.

```
Proxy
:: forall k. k -> Type
```

Visible **Dependent** Quantification

The body of the kind changes depending on its type.

```
        Proxy
:: forall k. k -> Type
```

Visible Dependent Quantification

The body of the kind changes depending on its type.

```
          Proxy
  :: forall k. k -> Type
          ------
Proxy @Type :: Type -> Type
```

Visible **Dependent**

The body of the kind changes depending on its type.

```
          Proxy
   :: forall k. k -> Type
          -----
Proxy @()  :: ()      -> Type
```

Visible Dependent Quantification

The body of the kind changes depending on its type.

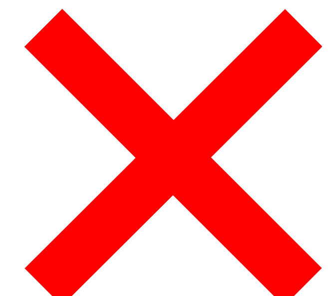
```
          Proxy
   :: forall k. k -> Type
          ------
Proxy @Bool :: Bool -> Type
```

Visible **Dependent**

The body of the kind changes depending on its type.

```
      Proxy
:: forall k. k -> Type
      -----
Proxy @Bool :: Bool -> Type
```

```
   Maybe
:: Type -> Type
```

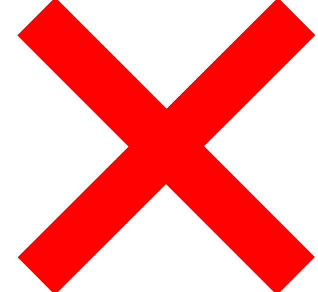


Visible Dependent

The body of the kind changes depending on its type.

```
      Proxy
 :: forall k. k -> Type
      ------
Proxy @Bool :: Bool -> Type
```

```
      Maybe
 :: Type -> Type
      ------
Maybe Int  :: Type
```



Visible Dependent

The body of the kind changes depending on its type.

```
Proxy
:: forall k. k -> Type
------
Proxy @Bool :: Bool -> Type
```

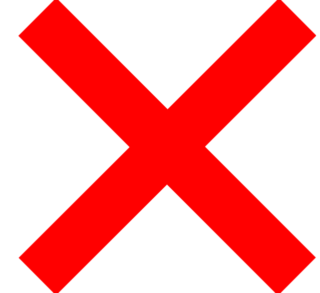
```
Maybe
:: Type -> Type
------
Maybe Char :: Type
```

Visible Dependent

The body of the kind changes depending on its type.

```
     Proxy
:: forall k. k -> Type
     ------
Proxy @Bool :: Bool -> Type
```

```
     Maybe
:: Type -> Type
     ------
Maybe Bool :: Type
```



Visible Dependent Quantification

Visible Dependent Quantification

```
data ProxyExp :: forall k -> k -> Type
```

Visible Dependent Quantification

```
data ProxyExp :: forall k -> k -> Type
```

Visible Dependent Quantification

```
data ProxyExp :: forall k -> k -> Type
```

Visible Dependent Quantification

```
data ProxyExp :: forall k -> k -> Type
                 ------
     ProxyExp Bool
```

Visible Dependent Quantification

```
data ProxyExp :: forall k -> k -> Type
                 ------
    ProxyExp Bool :: Bool -> Type
```

Visible Dependent Quantification

```
data ProxyExp :: forall k -> k -> Type
                 ------
ProxyExp Type :: Type -> Type
```

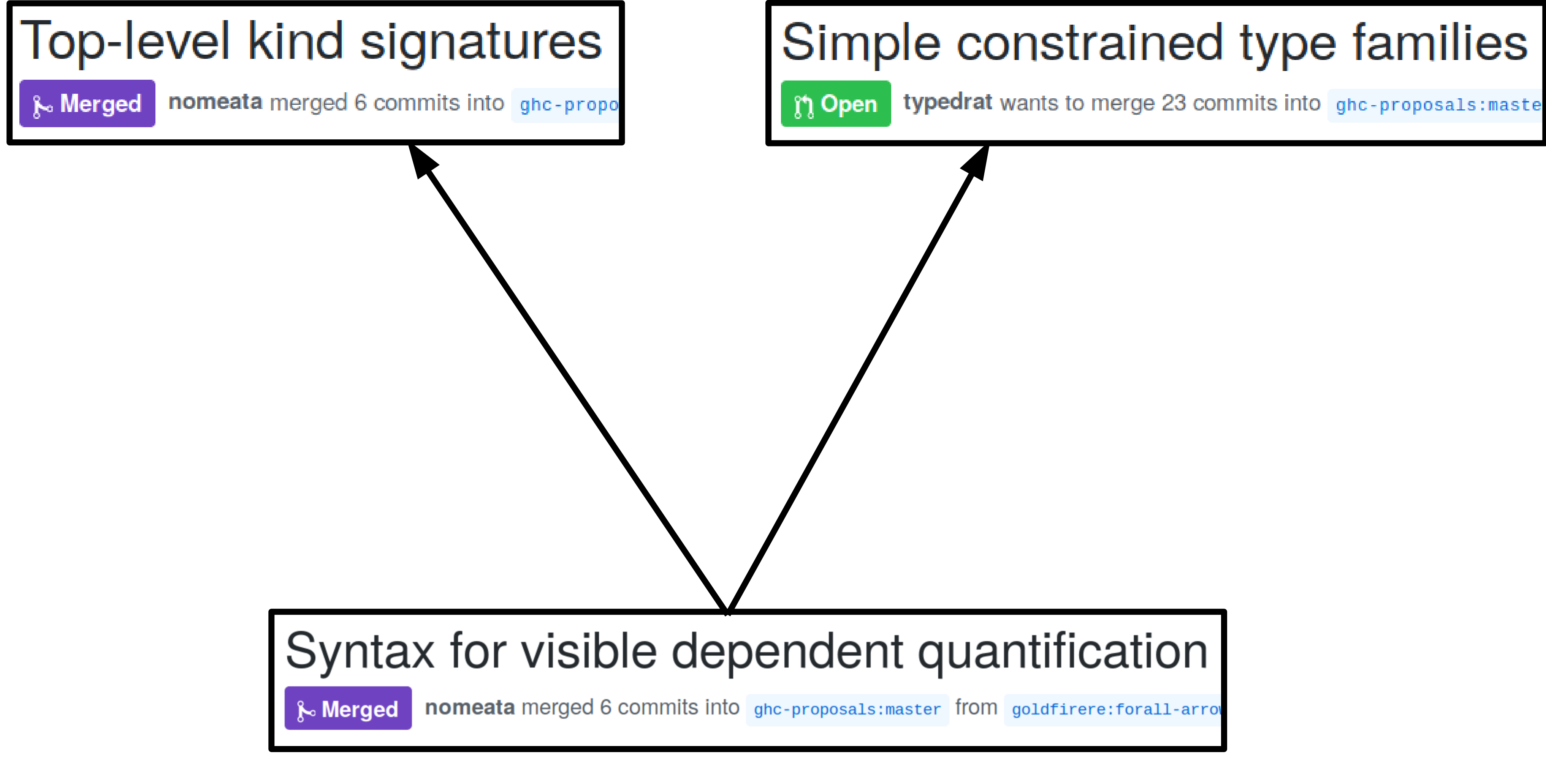
Visible Dependent Quantification

```
data ProxyExp :: forall k -> k -> Type
                 ------
     ProxyExp ()    :: ()    -> Type
```

Why do I want this?

Syntax for visible dependent quantification

Merged **nomeata** merged 6 commits into `ghc-proposals:master` from `goldfirere:forall-arro`


Top-level kind signatures
Merged
nomeata merged 6 commits into ghc-propo
Simple constrained type families
Open
typedrat wants to merge 23 commits into ghc-proposals:maste
Syntax for visible dependent quantification
Merged
nomeata merged 6 commits into ghc-proposals:master from goldfirere:forall-arro

Top-level kind signatures

Merged nomeata merged 6 commits into `ghc-propo`

Top-level kind signatures

Merged nomeata merged 6 commits into ghc-propo

```
type T :: (k -> Type) -> k -> Type
data T m a =
  MkT (m a) (T Maybe (m a))
```

Top-level kind signatures

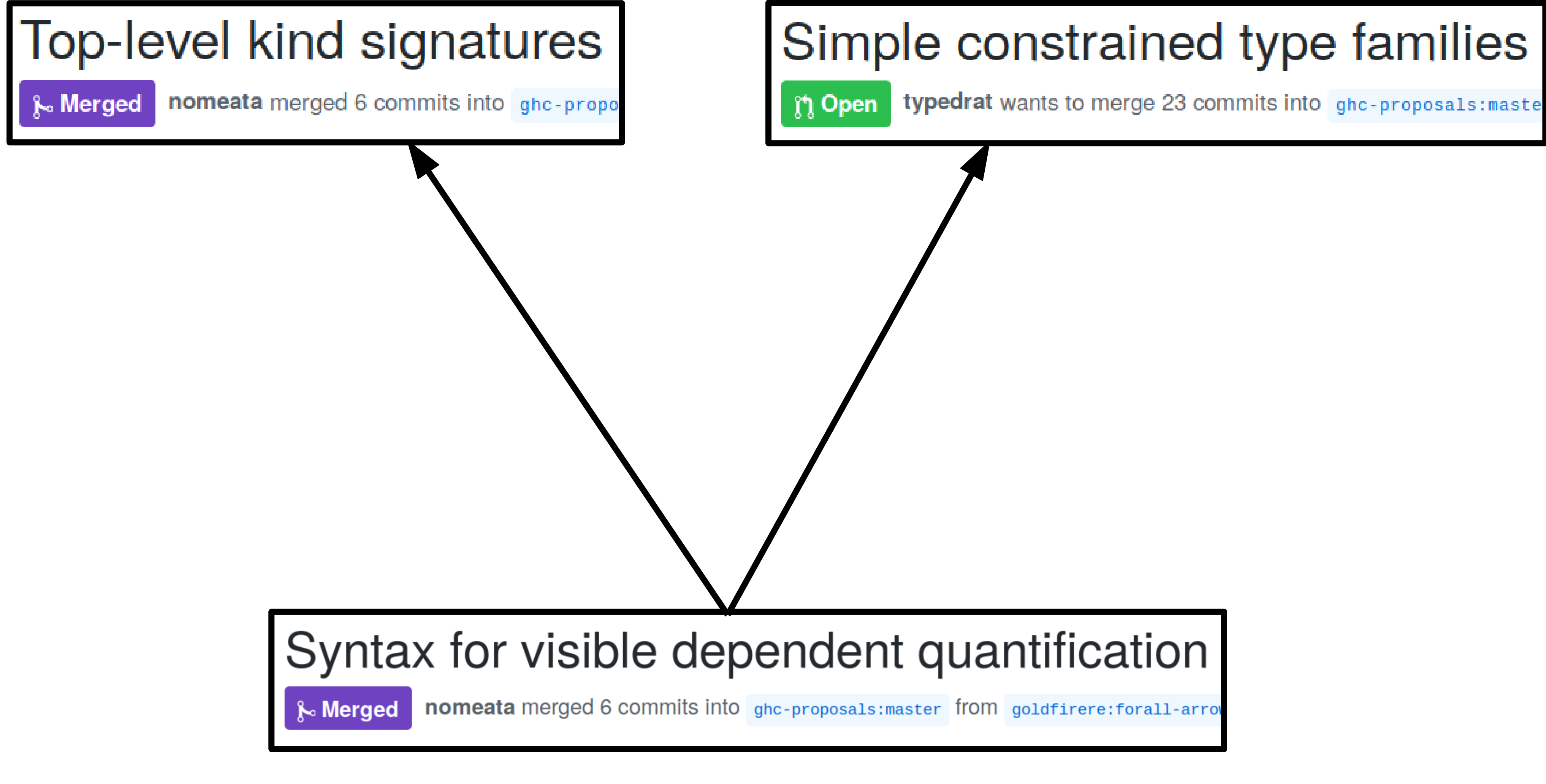
Merged nomeata merged 6 commits into ghc-propo

```
type T :: (k -> Type) -> k -> Type
data T m a =
  MkT (m a) (T Maybe (m a))


data T2 k m (a :: k) =
  MkT2 (m a) (T2 Type Maybe (m a))
```

Top-level kind signatures

Merged nomeata merged 6 commits into ghc-propo

```
type T :: (k -> Type) -> k -> Type
data T m a =
  MkT (m a) (T Maybe (m a))

type T2 :: forall k -> (k -> Type) -> k -> Type
data T2 k m (a :: k) =
  MkT2 (m a) (T2 Type Maybe (m a))
```


Top-level kind signatures
Merged
nomeata merged 6 commits into ghc-propo
Simple constrained type families
Open
typedrat wants to merge 23 commits into ghc-proposals:maste
Syntax for visible dependent quantification
Merged
nomeata merged 6 commits into ghc-proposals:master from goldfirere:forall-arro

Simple constrained type families

Open typedrat wants to merge 23 commits into `ghc-proposals:maste`

Simple constrained type families

Open typedrat wants to merge 23 commits into ghc-proposals:maste

```
class C a where
  type T a
```

Simple constrained type families

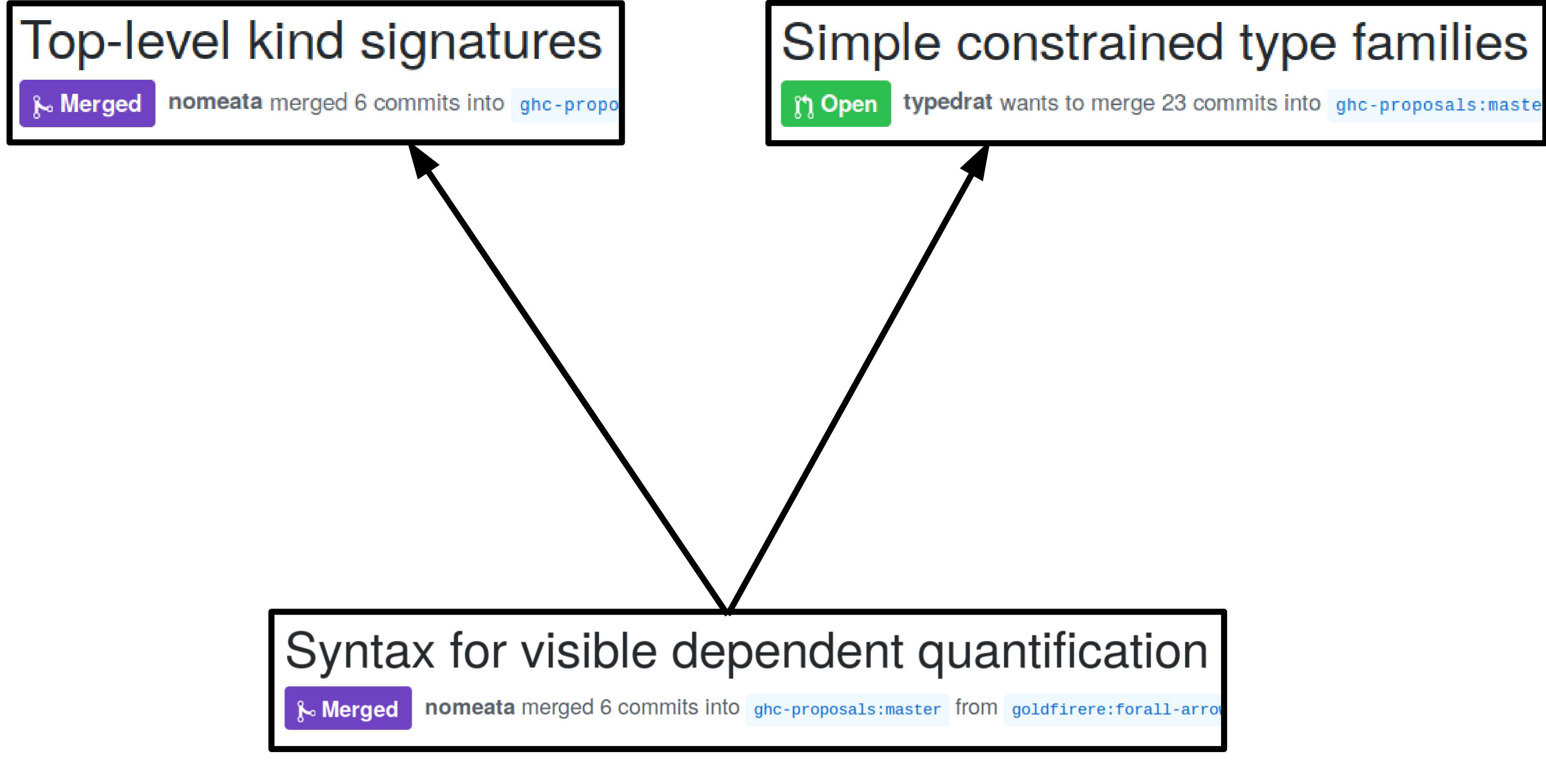
Open typedrat wants to merge 23 commits into ghc-proposals:maste

```
class C a where
  type T a

T :: Type -> Type -- Old kind
```

Simple constrained type families

Open typedrat wants to merge 23 commits into ghc-proposals:maste

```
class C a where
  type T a

T :: Type -> Type -- Old kind
T :: forall (a :: Type) -> C a => Type
                   -- New kind
```


Top-level kind signatures
Merged
nomeata merged 6 commits into ghc-propo
Simple constrained type families
Open
typedrat wants to merge 23 commits into ghc-proposals:maste
Syntax for visible dependent quantification
Merged
nomeata merged 6 commits into ghc-proposals:master from goldfirere:forall-arro

Top-level kind signatures

Merged nomeata merged 6 commits into ghc-propo

Simple constrained type families

Open typedrat wants to merge 23 commits into ghc-proposals:maste

Dependent Haskell? #???

I dunno someone probably wants to merge 23 comm

Syntax for visible dependent quantification

Merged nomeata merged 6 commits into ghc-proposals:master from goldfirere:forall-arro

```
_$_ :  ∀ {A : Set} {B : Set} →
       (A → B) →
       (A → B)
f $ x = f x
```

```
_$_ :  ∀ {A : Set} {B : A → Set} →
      (∀ (x : A) → B x) →
      (∀ (x : A) → B x)
f $ x = f x
```

```
_$_ :  ∀ {A : Set} {B : A → Set} →
       (∀ (x : A) → B x) →
       (∀ (x : A) → B x)
f $ x = f x

type ($) :: forall (a :: Type) (b :: a -> Type).
            (forall (x :: a) -> b x) ->
            (forall (x :: a) -> b x)
type f $ x = f x
```

```
_$_ :  ∀ {A : Set} {B : A → Set} →
       (∀ (x : A) → B x) →
       (∀ (x : A) → B x)
f $ x = f x
```

```
{-# LANGUAGE UnicodeSyntax #-}
type ($) :: ∀ (a :: Type) (b :: a → Type).
            (∀ (x :: a) → b x) →
            (∀ (x :: a) → b x)
type f $ x = f x
```

Implementing VDQ

Glasgow Haskell Compiler › GHC › Commits › **c26d299d**

Dec 18, 2018 1:54am

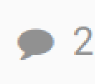
Commit c26d299d authored 7 months ago by **Ryan Scott** Committed by **Marge Bot** 4 months ago

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).

Glasgow Haskell Compiler > GHC > Commits > c26d299d

Dec 18, 2018 1:54am

Commit c26d299d authored 7 months ago by **Ryan Scott** Committed by **Marge Bot** 4 months ago 

Visible dependent quantification

> This implements [obscured]
> (https://github.[obscured] arrow.rst)
> by adding the ab[obscured]
> visible dependent quantification (VDQ).

Showing **64 changed files** with **814 additions** and **149 deletions**

```
data T (a :: Type) :: Type
```

```
data T (a :: Type) :: Type

T :: Type -> Type
```

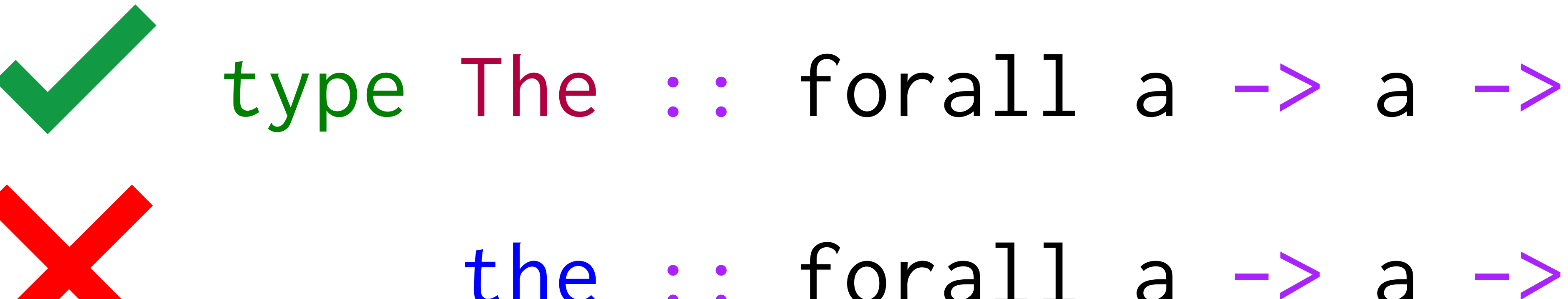
```haskell
data T (a :: Type) :: Type

T :: Type -> Type
T :: forall (a :: Type) -> Type
```

```
data T (a :: Type) :: Type

T :: Type -> Type
T :: forall (a :: Type) -> Type
```

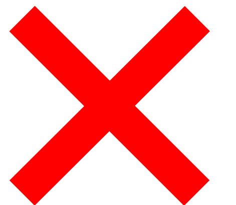
```
data T (a :: Type) :: Type

T :: Type -> Type
T :: forall (a :: Type) -> Type
```

These two kinds are *not* equivalent (currently)

```
type The :: forall a -> a -> a
     the :: forall a -> a -> a
```


type The :: forall a -> a -> a
the :: forall a -> a -> a

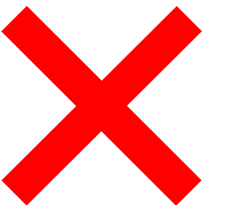
```
type The :: forall a -> a -> a
```



```
the :: forall a -> a -> a
```

```
λ> the :: forall a -> a -> a
```

```
type The :: forall a -> a -> a
```



```
the :: forall a -> a -> a
```

```
λ> the :: forall a -> a -> a
<interactive>:7:8: error:
    • Illegal visible, dependent quantification in the type of a term:
        forall a -> a -> a
      (GHC does not yet support this)
```

VDQ

- Fills in a gaping hole in GHC's kind language
- An important step towards dependent types in Haskell
- Amaze your friends, impress your coworkers, wow!

Debuts in GHC 8.10!