Inferring Datatypes

Ningning Xie
HIW 2019.08.23

```
data Maybe a = Nothing | Just a

data List a = Nil | Cons a (List a)

data Either a b = Left a | Right b
```

Which datatype declarations should be accepted?

What kinds do accepted datatypes have?

QUITZ TIME!

Haskell98

```
data AppInt f = Mk (f Int)

AppInt :: ?
```

Haskell98

```
data AppInt f = Mk (f Int)
```

AppInt :: ?

AppInt :: $(* \to *) \to *$

```
data Q1 a = MkQ1

data Q2 = MkQ2 (Q1 Maybe)
```

```
data Q1 a = MkQ1                -- Q1 :: * → *

data Q2 = MkQ2 (Q1 Maybe)
```

```
data Q1 a = MkQ1                    -- Q1 :: * → *

data Q2 = MkQ2 (Q1 Maybe)           -- Rejected!
```

```
data Q1 a = MkQ1 Q2

data Q2 = MkQ2 (Q1 Maybe)
```

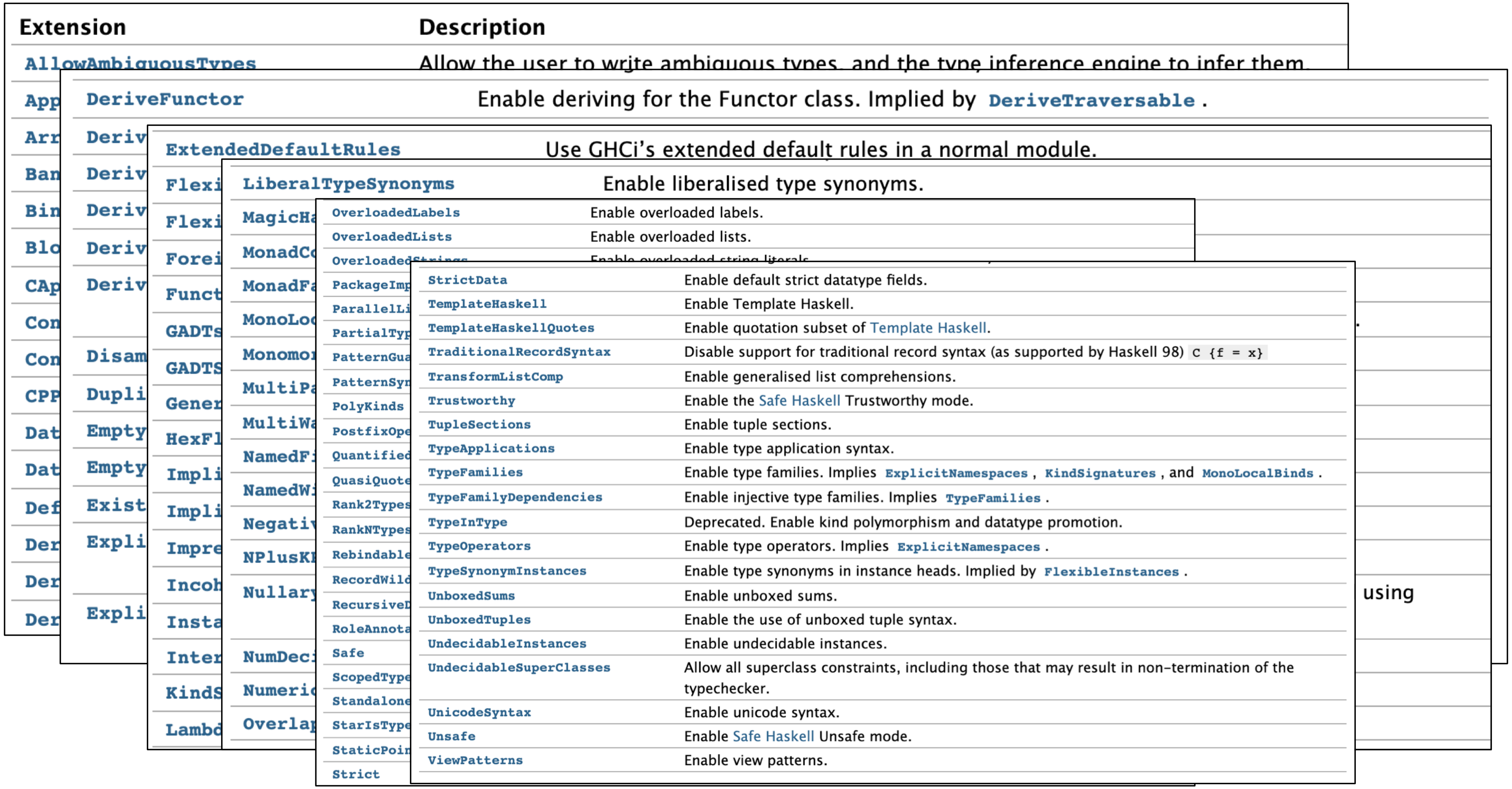
```
data Q1 a = MkQ1 Q2                  -- Q1 :: (* → *) → *

data Q2 = MkQ2 (Q1 Maybe)
```

```
data Q1 a = MkQ1 Q2                -- Q1 :: (* → *) → *

data Q2 = MkQ2 (Q1 Maybe)          -- Accepted!
```

Modern Haskell



Extension	Description
DeriveFunctor	Enable deriving for the Functor class. Implied by DeriveTraversable .
ExtendedDefaultRules	Use GHCi's extended default rules in a normal module.
LiberalTypeSynonyms	Enable liberalised type synonyms.
OverloadedLabels	Enable overloaded labels.
OverloadedLists	Enable overloaded lists.
StrictData	Enable default strict datatype fields.
TemplateHaskell	Enable Template Haskell.
TemplateHaskellQuotes	Enable quotation subset of Template Haskell.
TraditionalRecordSyntax	Disable support for traditional record syntax (as supported by Haskell 98) `C {f = x}`
TransformListComp	Enable generalised list comprehensions.
Trustworthy	Enable the Safe Haskell Trustworthy mode.
TupleSections	Enable tuple sections.
TypeApplications	Enable type application syntax.
TypeFamilies	Enable type families. Implies ExplicitNamespaces , KindSignatures , and MonoLocalBinds .
TypeFamilyDependencies	Enable injective type families. Implies TypeFamilies .
TypeInType	Deprecated. Enable kind polymorphism and datatype promotion.
TypeOperators	Enable type operators. Implies ExplicitNamespaces .
TypeSynonymInstances	Enable type synonyms in instance heads. Implied by FlexibleInstances .
UnboxedSums	Enable unboxed sums.
UnboxedTuples	Enable the use of unboxed tuple syntax.
UndecidableInstances	Enable undecidable instances.
UndecidableSuperClasses	Allow all superclass constraints, including those that may result in non-termination of the typechecker.
UnicodeSyntax	Enable unicode syntax.
Unsafe	Enable Safe Haskell Unsafe mode.
ViewPatterns	Enable view patterns.

```haskell
{-# LANGUAGE
  ExplicitForAll
, PolyKinds
, ExistentialQuantification
, TypeInType
, TypeApplications
#-}
```

```
data X :: forall a (b::*->*). a b -> *

data Y :: forall (c :: Maybe Bool). X c -> *
```

```
data X :: forall a (b::*->*). a b -> *

data Y :: forall (c :: Maybe Bool). X c -> *
```

```
data X :: forall a (b::*->*). a b -> *

data Y :: forall (c :: Maybe Bool). X c -> *
```

$a :: (* \to *) \to *$

Kind mismatch!

```
data Q :: forall (a :: f b) (c :: k) . f c -> *
```

```
data Q :: forall (a :: f b) (c :: k) . f c -> *
```

```
data Q :: forall (a :: f b) (c :: k) . f c -> *

data Q :: forall (f::?) (b::?) (k::?) forall (a :: f b) (c :: k) . f c -> *
```

```
data Q :: forall (a :: f b) (c :: k). f c -> *

c :: k
k :: *
```

```
data Q :: forall (a :: f b) (c :: k) . f c -> *
```

```
c :: k
k :: *
f :: k -> *
```

```
data Q :: forall (a :: f b) (c :: k) . f c -> *

c :: k
k :: *
f :: k -> *
b :: k
```

```
data Q :: forall (a :: f b) (c :: k) . f c -> *
```

```
k :: *
f :: k -> *
b :: k
```

```
data Q :: forall (a :: f b) (c :: k) . f c -> *

data Q :: forall (f::?) (b::?) (k::?) forall (a :: f b) (c :: k) . f c -> *
```

```
data Q :: forall (a :: f b) (c :: k) . f c -> *

data Q :: forall (k::?) (f::?) (b::?) forall (a :: f b) (c :: k) . f c -> *
```

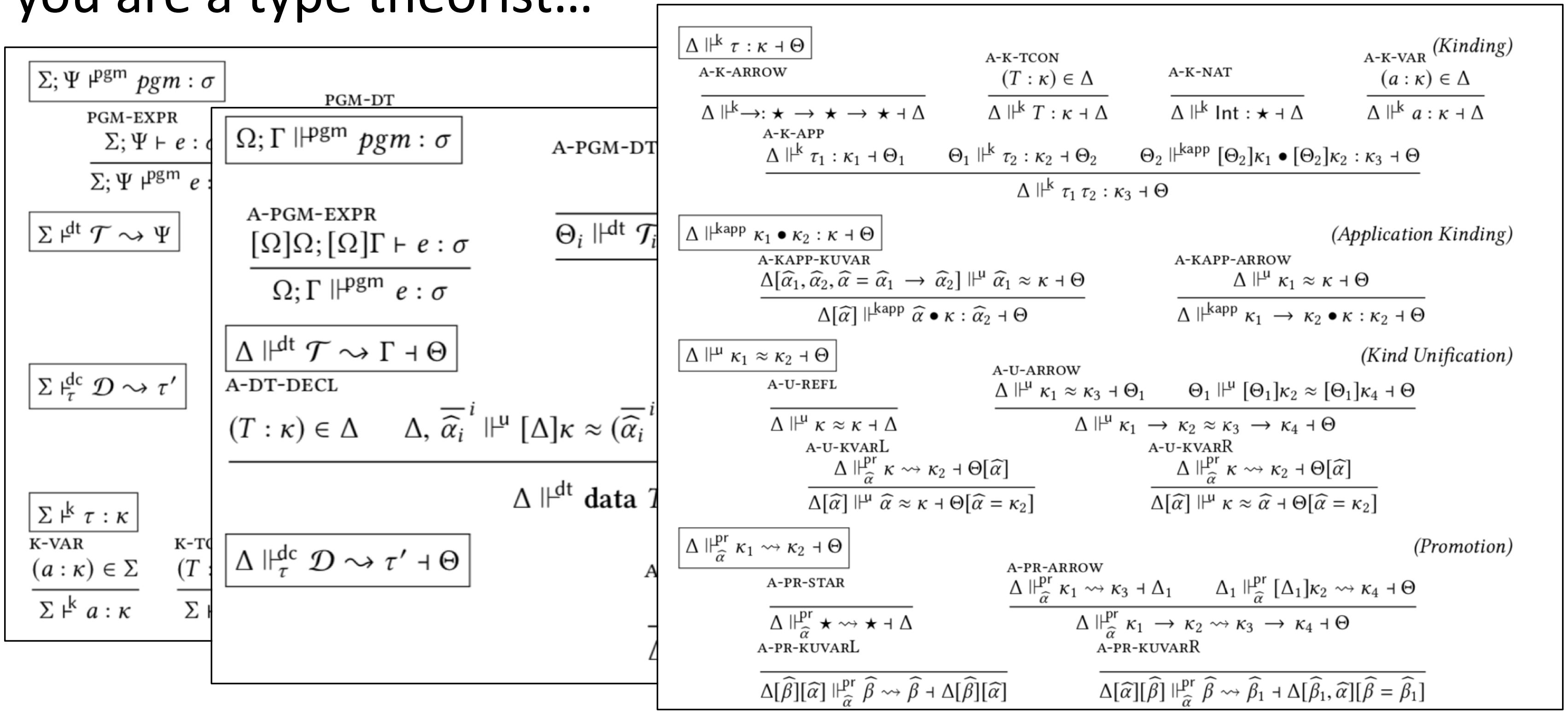
```
data Q :: forall (a :: f b) (c :: k) . f c -> *

data Q :: forall (k::*) (f::k->*) (b::k) forall (a :: f b) (c :: k) . f c -> *
```

Inferring Datatypes

Inferring Datatypes

- If you are a type theorist…



Inferring Datatypes

- If you are a GHC hacker/developer…
  - Further language extensions
  - How our work relates to GHC