

# 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?

**QUIZ TIME!**

# Haskell98

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

```
AppInt :: ?
```

# Haskell98

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

**AppInt :: ?**

**AppInt :: (\* → \*) → \***

```
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	
<a href="#">AllowAmbiguousTypes</a>	Allow the user to write ambiguous types and the type inference engine to infer them.	
App <a href="#">DeriveFunctor</a>	Enable deriving for the Functor class. Implied by <a href="#">DeriveTraversable</a> .	
Arr <a href="#">Deriv</a>	<a href="#">ExtendedDefaultRules</a>	
Ban <a href="#">Deriv</a>	Use GHCi's extended default rules in a normal module.	
Bin <a href="#">Deriv</a>	<a href="#">Flexi</a>	
Blo <a href="#">Deriv</a>	<a href="#">LiberalTypeSynonyms</a>	
Cap <a href="#">Deriv</a>	Enable liberalised type synonyms.	
Con <a href="#">GADTs</a>	<a href="#">MagicHash</a>	
Con <a href="#">Disam</a>	<a href="#">OverloadedLabels</a>	
CPP <a href="#">Deriv</a>	Enable overloaded labels.	
Dat <a href="#">Empty</a>	<a href="#">OverloadedLists</a>	
Dat <a href="#">Empty</a>	Enable overloaded lists.	
Def <a href="#">Exist</a>	<a href="#">OverloadedStringLiterals</a>	
Der <a href="#">Expli</a>	<a href="#">OverloadedStringLiterals</a>	
Der <a href="#">Expli</a>	<a href="#">StrictData</a>	
Der <a href="#">Expli</a>	Enable default strict datatype fields.	
Der <a href="#">Expli</a>	<a href="#">TemplateHaskell</a>	
Der <a href="#">Expli</a>	Enable Template Haskell.	
Der <a href="#">Expli</a>	<a href="#">TemplateHaskellQuotes</a>	
Der <a href="#">Expli</a>	Enable quotation subset of Template Haskell.	
Der <a href="#">Expli</a>	<a href="#">TraditionalRecordSyntax</a>	
Der <a href="#">Expli</a>	Disable support for traditional record syntax (as supported by Haskell 98) <code>c {f = x}</code> .	
Der <a href="#">Expli</a>	<a href="#">TransformListComp</a>	
Der <a href="#">Expli</a>	Enable generalised list comprehensions.	
Der <a href="#">Expli</a>	<a href="#">Trustworthy</a>	
Der <a href="#">Expli</a>	Enable the Safe Haskell Trustworthy mode.	
Dat <a href="#">Empty</a>	<a href="#">MultiParamTypeClasses</a>	
Dat <a href="#">Empty</a>	<a href="#">MultiWayTypeClasses</a>	
Dat <a href="#">Empty</a>	<a href="#">PostfixOperators</a>	
Dat <a href="#">Empty</a>	<a href="#">QuantifiedTypeVariables</a>	
Dat <a href="#">Empty</a>	<a href="#">QuasiQuotes</a>	
Dat <a href="#">Empty</a>	Enable type families. Implies <a href="#">ExplicitNamespaces</a> , <a href="#">KindSignatures</a> , and <a href="#">MonoLocalBinds</a> .	
Def <a href="#">Exist</a>	<a href="#">NamedFieldAnnotations</a>	
Def <a href="#">Exist</a>	<a href="#">Rank2Types</a>	
Def <a href="#">Exist</a>	<a href="#">TypeFamilies</a>	
Def <a href="#">Exist</a>	<a href="#">TypeFamilyDependencies</a>	
Der <a href="#">Expli</a>	<a href="#">TypeInType</a>	
Der <a href="#">Expli</a>	Deprecated. Enable kind polymorphism and datatype promotion.	
Der <a href="#">Expli</a>	<a href="#">TypeOperators</a>	
Der <a href="#">Expli</a>	Enable type operators. Implies <a href="#">ExplicitNamespaces</a> .	
Der <a href="#">Expli</a>	<a href="#">TypeSynonymInstances</a>	
Der <a href="#">Expli</a>	Enable type synonyms in instance heads. Implied by <a href="#">FlexibleInstances</a> .	
Der <a href="#">Expli</a>	<a href="#">UnboxedSums</a>	
Der <a href="#">Expli</a>	Enable unboxed sums.	
Der <a href="#">Expli</a>	<a href="#">UnboxedTuples</a>	
Der <a href="#">Expli</a>	Enable the use of unboxed tuple syntax.	
Der <a href="#">Expli</a>	<a href="#">RoleAnnotations</a>	
Der <a href="#">Expli</a>	<a href="#">UndecidableInstances</a>	
Der <a href="#">Expli</a>	Enable undecidable instances.	
Der <a href="#">Expli</a>	<a href="#">UndecidableSuperClasses</a>	
Der <a href="#">Expli</a>	Allow all superclass constraints, including those that may result in non-termination of the typechecker.	
Der <a href="#">Expli</a>	<a href="#">UnicodeSyntax</a>	
Der <a href="#">Expli</a>	Enable unicode syntax.	
Der <a href="#">Expli</a>	<a href="#">Unsafe</a>	
Der <a href="#">Expli</a>	Enable Safe Haskell Unsafe mode.	
Der <a href="#">Expli</a>	<a href="#">StaticPointers</a>	
Der <a href="#">Expli</a>	<a href="#">ViewPatterns</a>	
Der <a href="#">Expli</a>	Enable view patterns.	

using

```
{-# 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 :: (\* → \*) → \*

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...

<p><b>PGM-DT</b></p> $\Omega; \Gamma \vdash^{\text{pgm}} pgm : \sigma$ <p>A-PGM-DT</p> $\Theta_i \vdash^{\text{dt}} \mathcal{T}_i$ <p><b>A-PGM-EXPR</b></p> $[\Omega] \Omega; [\Omega] \Gamma \vdash e : \sigma$ $\Omega; \Gamma \vdash^{\text{pgm}} e : \sigma$ <p><b>A-DT-DECL</b></p> $(T : \kappa) \in \Delta \quad \Delta, \overline{\widehat{\alpha}}_i^i \vdash^{\text{u}} [\Delta]\kappa \approx (\overline{\widehat{\alpha}}_i^i)$ $\Delta \vdash^{\text{dt}} \text{data } T$ <p><b>K-TCON</b></p> $(T : \kappa) \in \Delta$ $\Sigma \vdash^{\text{dt}} \mathcal{D} \rightsquigarrow \tau' + \Theta$	<p><b>Kinding</b></p> $\Delta \vdash^k \tau : \kappa \dashv \Theta$ <p>A-K-ARROW</p> $\Delta \vdash^k \star \rightarrow \star \rightarrow \star \dashv \Delta$ <p>A-K-APP</p> $\Delta \vdash^k \tau_1 : \kappa_1 \dashv \Theta_1 \quad \Theta_1 \vdash^k \tau_2 : \kappa_2 \dashv \Theta_2 \quad \Theta_2 \vdash^{\text{kapp}} [\Theta_2]\kappa_1 \bullet [\Theta_2]\kappa_2 : \kappa_3 \dashv \Theta$ $\Delta \vdash^k \tau_1 \tau_2 : \kappa_3 \dashv \Theta$ <p>(Application Kinding)</p> $\Delta \vdash^{\text{kapp}} \kappa_1 \bullet \kappa_2 : \kappa \dashv \Theta$ <p>A-KAPP-KUVAR</p> $\Delta[\widehat{\alpha}_1, \widehat{\alpha}_2, \widehat{\alpha} = \widehat{\alpha}_1 \rightarrow \widehat{\alpha}_2] \vdash^{\text{u}} \widehat{\alpha}_1 \approx \kappa \dashv \Theta$ $\Delta[\widehat{\alpha}] \vdash^{\text{kapp}} \widehat{\alpha} \bullet \kappa : \widehat{\alpha}_2 \dashv \Theta$ <p>A-KAPP-ARROW</p> $\Delta \vdash^{\text{u}} \kappa_1 \approx \kappa \dashv \Theta$ $\Delta \vdash^{\text{kapp}} \kappa_1 \rightarrow \kappa_2 \bullet \kappa : \kappa_2 \dashv \Theta$ <p>(Kind Unification)</p> $\Delta \vdash^{\text{u}} \kappa_1 \approx \kappa_2 \dashv \Theta$ <p>A-U-REFL</p> $\Delta \vdash^{\text{u}} \kappa \approx \kappa \dashv \Delta$ <p>A-U-KVARL</p> $\Delta \vdash^{\text{pr}}_{\widehat{\alpha}} \kappa \rightsquigarrow \kappa_2 \dashv \Theta[\widehat{\alpha}]$ $\Delta[\widehat{\alpha}] \vdash^{\text{u}} \widehat{\alpha} \approx \kappa \dashv \Theta[\widehat{\alpha} = \kappa_2]$ <p>A-U-KVARR</p> $\Delta \vdash^{\text{pr}}_{\widehat{\alpha}} \kappa \rightsquigarrow \kappa_2 \dashv \Theta[\widehat{\alpha}]$ $\Delta[\widehat{\alpha}] \vdash^{\text{u}} \kappa \approx \widehat{\alpha} \dashv \Theta[\widehat{\alpha} = \kappa_2]$ <p>(Promotion)</p> $\Delta \vdash^{\text{pr}}_{\widehat{\alpha}} \kappa_1 \rightsquigarrow \kappa_2 \dashv \Theta$ <p>A-PR-STAR</p> $\Delta \vdash^{\text{pr}}_{\widehat{\alpha}} \star \rightsquigarrow \star \dashv \Delta$ <p>A-PR-KUVARL</p> $\Delta[\widehat{\beta}][\widehat{\alpha}] \vdash^{\text{pr}}_{\widehat{\alpha}} \widehat{\beta} \rightsquigarrow \widehat{\beta} \dashv \Delta[\widehat{\beta}][\widehat{\alpha}]$ <p>A-PR-KUVARR</p> $\Delta[\widehat{\alpha}][\widehat{\beta}] \vdash^{\text{pr}}_{\widehat{\alpha}} \widehat{\beta} \rightsquigarrow \widehat{\beta}_1 \dashv \Delta[\widehat{\beta}_1, \widehat{\alpha}][\widehat{\beta} = \widehat{\beta}_1]$
--	--

# Inferring Datatypes

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