Pattern Synonyms

Matthew Pickering[1] Gergő Érdi[2] Simon Peyton Jones[3]
Richard A. Eisenberg[4]

[1]University of Oxford

[2]Standard Chartered Bank

[3]Microsoft Research

[4]Bryn Mawr College

Haskell Symposium, September 2016.

Example

```
   -- Datatype definition
data Type = TyApp String [Type]

   -- Functions abstract construction
tyInt :: Type
tyInt = TyApp "Int" []
mkFunTy :: Type → Type → Type
mkFunTy t u = TyApp "->" [t, u]
plusTy :: Type
plusTy = tyInt ‘mkFunTy‘ tyInt ‘mkFunTy‘ tyInt
```

Example

```
  -- Datatype definition
data Type = TyApp String [Type]

  -- Pattern synonym abstracts matching
pattern FunTy :: Type → Type → Type
pattern FunTy t u = TyApp "->" [t, u]
funArgTys :: Type → ([Type], Type)
funArgTys (FunTy t u) = case funArgTys u of
  (ts, r) → (t : ts, r)
funArgTys t = ([ ], t)
```

Pattern synonyms

- Goal: bring function-like abstraction to pattern matching
- Touches all parts of the GHC frontend:
    - Parser, renamer
    - Typechecker
    - Desugarer
    - Interface files
- No backend changes needed
- Our paper shows the breadth; here we show some depth:
    - Typechecking
    - Desugaring

Pattern types

Pattern types

The Typing Principle

It should be possible to determine whether a use of P is well-typed based only on P's type, without reference to P's definition.

What is the type of a pattern?

Pattern types

Scrutinee type:

pattern $\mathit{P1} = \mathit{True}$

pattern $\mathit{P2} = \mathit{Just}\ \mathit{True}$

pattern $\mathit{P3} = \mathit{Nothing}$

Parametric patterns:

pattern $\mathit{P4}\ x = \mathit{Just}\ x$

Pattern types

Scrutinee type:

$$\textbf{pattern}\ \mathit{P1} :: \mathit{Bool}$$
$$\textbf{pattern}\ \mathit{P1} = \mathit{True}$$

$$\textbf{pattern}\ \mathit{P2} :: \mathit{Maybe}\ \mathit{Bool}$$
$$\textbf{pattern}\ \mathit{P2} = \mathit{Just}\ \mathit{True}$$

$$\textbf{pattern}\ \mathit{P3} :: \forall a. \mathit{Maybe}\ a$$
$$\textbf{pattern}\ \mathit{P3} = \mathit{Nothing}$$

Parametric patterns:

$$\textbf{pattern}\ \mathit{P4}\ x = \mathit{Just}\ x$$

Pattern types

Scrutinee type:

pattern *P1* :: *Bool*
pattern *P1* = *True*

pattern *P2* :: *Maybe Bool*
pattern *P2* = *Just True*

pattern *P3* :: $\forall a.Maybe\ a$
pattern *P3* = *Nothing*

Parametric patterns:

pattern *P4* :: $\forall a.a \rightarrow Maybe\ a$
pattern *P4* x = *Just* x

Pattern types: required constraints

Required constraints:

$$\textbf{pattern}\ \textit{P5} = 42$$

$$\textbf{pattern}\ \textit{P6} = (\textit{show} \rightarrow \texttt{"foo"})$$

Pattern types: required constraints

Required constraints:

```
pattern P5 :: ∀a.(Num a, Eq a) ⇒ a
pattern P5 = 42

pattern P6 = (show → "foo")
```

Pattern types: required constraints

Required constraints:

$$\textbf{pattern}\ \mathit{P5} :: \forall a.(\mathit{Num}\ a, \mathit{Eq}\ a) \Rightarrow a$$
$$\textbf{pattern}\ \mathit{P5} = 42$$

$$\textbf{pattern}\ \mathit{P6} :: \forall a.(\mathit{Show}\ a) \Rightarrow a$$
$$\textbf{pattern}\ \mathit{P6} = (\mathit{show} \rightarrow \texttt{"foo"})$$

Pattern types: existentials and provided constraints

```
data T a where
  MkT :: (Eq a, Show b) ⇒ a → a → b → T a

f (MkT x y v) = if x ≡ y then Just (show v) else Nothing
```

Matching on *MkT* brings in scope

- the type *b*
- $(\mathit{Eq}\ a, \mathit{Show}\ b)$

allowing $(\equiv)$ and *show* to be used on the right-hand side

Pattern types: existentials and provided constraints

```
data T a where
  MkT :: (Eq a, Show b) ⇒ a → a → b → T a

f (MkT x y v) = if x ≡ y then Just (show v) else Nothing
```

Matching on *MkT* brings in scope

- the type *b*
- (*Eq a*, *Show b*)

allowing (≡) and *show* to be used on the right-hand side

```
f :: T a → Maybe String
```

Pattern types: existentials and provided constraints

$$
\begin{array}{l}
\textbf{data}\ T\ a\ \textbf{where} \\
\quad MkT :: (Eq\ a, Show\ b) \Rightarrow a \to a \to b \to T\ a \\
\\
\textbf{pattern}\ P\ x\ y\ v = MkT\ x\ y\ v
\end{array}
$$

Matching on P brings in scope

- the type b
- $(Eq\ a, Show\ b)$

Pattern types: existentials and provided constraints

```
data T a where
  MkT :: (Eq a, Show b) ⇒ a → a → b → T a

pattern P x y v = MkT x y v
```

Matching on *P* brings in scope

- the type *b*
- $(Eq\ a, Show\ b)$

```
pattern P :: ∀a.() ⇒ ∀b.(Eq a, Show b) ⇒ a → a → b → T a
```

Pattern types: required & provided constraints

data $T\ a$ **where**
  $MkT :: (Eq\ a, Show\ b) \Rightarrow a \rightarrow b \rightarrow T\ a$
pattern $P\ v = MkT\ 1\ v$

- Matching on P **requires** $(Eq\ a, Num\ a)$
- Matching on P **provides** $(Eq\ a, Show\ b)$

Pattern types: required & provided constraints

$$
\begin{array}{l}
\textbf{data}\ T\ a\ \textbf{where} \\
\quad MkT :: (Eq\ a, Show\ b) \Rightarrow a \rightarrow b \rightarrow T\ a \\
\textbf{pattern}\ P\ v = MkT\ 1\ v
\end{array}
$$

- Matching on P **requires** $(Eq\ a, Num\ a)$
- Matching on P **provides** $(Eq\ a, Show\ b)$

$$\textbf{pattern}\ P :: \forall a.(Num\ a) \Rightarrow \forall b.(Eq\ a, Show\ b) \Rightarrow b \rightarrow T\ a$$

Pattern synonym types

Pattern synonym types are fully specified on six axes:

1. Universially bound type variables *univ*
2. The required context *Req* (with *univ* in scope)
3. The scrutinee type *t* (with *univ* in scope)

Surface syntax for pattern synonym type signatures:

$$\textbf{pattern}\ P :: \forall \mathit{univ}. \mathit{Req} \Rightarrow t$$

Pattern synonym types

Pattern synonym types are fully specified on six axes:

1. Universially bound type variables *univ*
2. The required context *Req* (with *univ* in scope)
3. The scrutinee type *t* (with *univ* in scope)
4. Existentially bound type variables *ex*
5. The provided context *Prov* (with *univ* and *ex* in scope)
6. The types of parameters *t1*, *t2*, . . . (with *univ* and *ex* in scope)

Surface syntax for pattern synonym type signatures:

$$\textbf{pattern}\ P :: \forall \mathit{univ}.\mathit{Req} \Rightarrow \forall \mathit{ex}.\mathit{Prov} \Rightarrow \mathit{t1} \rightarrow \mathit{t2} \rightarrow \ldots \rightarrow t$$

Desugaring

Desugaring

$$
\begin{array}{l}
\textbf{pattern}\ P :: \forall a.(\mathit{Num}\ a) \Rightarrow \forall b.(\mathit{Eq}\ a, \mathit{Show}\ b) \Rightarrow \\
\qquad\qquad\quad b \rightarrow T\ a \\
\textbf{pattern}\ P\ v = \mathit{MkT}\ 3\ v
\end{array}
$$

$$
\begin{array}{l}
\$\mathit{mP} :: \forall r\ a.(\mathit{Num}\ a) \Rightarrow \\
\qquad\quad T\ a \rightarrow \\
\qquad\quad (\forall b.(\mathit{Eq}\ a, \mathit{Show}\ b) \Rightarrow b \rightarrow r) \rightarrow r \rightarrow r \\
\$\mathit{mP}\ x\ \mathit{sk}\ \mathit{fk} = \textbf{case}\ x\ \textbf{of} \\
\qquad \mathit{MkT}\ 3\ v \rightarrow \mathit{sk}\ v \\
\qquad \_ \qquad\qquad\ \ \rightarrow \mathit{fk}
\end{array}
$$

Desugaring

Can be represented in existing GHC Core

- No changes needed anywhere downstream
- Exported, linked, and potentially inlined just like any other function
- Synthetic *Void*# parameter can be used to prevent incorrect strictness when r is unboxed

Desugaring

Semantics is different from macro-substitution!
The full (potentially nested) structure of the pattern synonym is matched first:

$$\begin{array}{l} \mathit{f1} :: [\,\mathit{Bool}\,] \rightarrow \mathit{Bool} \\ \mathit{f1}\ [\,\mathit{True}\,] = \mathit{True} \\ \mathit{f1}\ \_ \qquad\quad = \mathit{False} \end{array}$$

$$\begin{array}{l} \textbf{pattern}\ \mathit{Single}\ a = [\,a\,] \\ \mathit{f2} :: [\,\mathit{Bool}\,] \rightarrow \mathit{Bool} \\ \mathit{f2}\ (\mathit{Single}\ \mathit{True}) = \mathit{True} \\ \mathit{f2}\ \_ \qquad\qquad\qquad = \mathit{False} \end{array}$$

$$\begin{array}{l} \mathit{f1}\ [\bot,\bot] = \bot \\ \mathit{f1}\ (\mathit{False} : \bot) = \mathit{False} \end{array}$$

$$\begin{array}{l} \mathit{f2}\ [\bot,\bot] = \mathit{False} \\ \mathit{f2}\ (\mathit{False} : \bot) = \bot \end{array}$$

Conclusion

History

- Proposal added in 2011
- New in GHC 7.8 in 2014
- Incremental improvements in GHC 7.10 and 8.0
- Used in 72 packages on Hackage (as of June 2016)

Check our paper for more. . .

- Lots of examples
- Pattern synonym directionality
- Record syntax support
- Importing/exporting
- Shorthand syntax for pattern synonym signatures
- Formalisation of pattern types (in the extended version)

`http://unsafePerform.IO/patsyn`