

Pattern Matching

Wot's... Uh the Deal?

Stack Builders

0.1.0

Praise for pattern matching

The patch will not be noticeable if the pattern is skilfully matched.

—Idabelle McGlaulin, *Handicraft for Girls*

Wot's... Uh the Deal?

► `newtype N = N Bool`
`data D = D !Bool`

► `> (\ (N True) -> True) undefined`
`?`
`> (\ (D True) -> True) undefined`
`?`

► `> (\ ~(N True) -> True) undefined`
`?`
`> (\ ~(D True) -> True) undefined`
`?`

Wot's... Uh the Deal?

► `newtype N = N Bool`
`data D = D !Bool`

► `> (\ (N True) -> True) undefined`
`undefined`
`> (\ (D True) -> True) undefined`
`undefined`

► `> (\ ~(N True) -> True) undefined`
`True`
`> (\ ~(D True) -> True) undefined`
`True`

Booleans

► `data Bool = False | True`

► `not :: Bool -> Bool`
`not False = True`
`not True = False`

Maybe

► `data Maybe a = Nothing | Just a`

► `isNothing :: Maybe a -> Bool`
`isNothing Nothing = True`
`isNothing _ = False`

Lists

-
- ▶ `data [] a = [] | a : [a]`
-
- ▶ `map :: (a -> b) -> [a] -> [b]`
`map _ [] = []`
`map f (x:xs) = f x : map f xs`
-

Pattern matching

Patterns

Example

```
span :: (a -> Bool) -> [a] -> ([a],[a])
span _ xs@[]      = (xs,xs)
span p xs@(x:xs')
  | p x           = let (ys,zs) = span p xs' in (x:ys,zs)
  | otherwise     = ([],xs)
```

Pattern matching

Patterns

Example

► `foldr :: (a -> b -> b) -> b -> [a] -> b`
`foldr _ n [] = n`
`foldr c n (x:xs) = c x (foldr c n xs)`

► `unzip :: [(a,b)] -> ([a],[b])`
`unzip = foldr (\(a,b) ~(as,bs) -> (a:as,b:bs)) ([],[b])`

Pattern matching

Patterns

Example

```
fromMaybe :: a -> Maybe a -> a
fromMaybe d mx =
  case mx of
    Nothing -> d
    Just x   -> x
```

```
undefined :: a  
undefined = undefined
```

```
const1 :: a -> Int  
const1 x = 1
```

Informal semantics of pattern matching

Several species of small furry `null` functions

```
null1 :: [a] -> Bool
null1 []      = True
null1 (_:_)   = False
```

Informal semantics of pattern matching

Several species of small furry `null` functions

```
null2 :: [a] -> Bool
null2 []      = True
null2 _      = False
```

Examples

- ▶ If `['a', 'b']` is matched against `['x', undefined]`, then
- ▶ If `['a', 'b']` is matched against `[undefined, 'x']`, then

Examples

- ▶ If `['a', 'b']` is matched against `['x', undefined]`, then
 - ▶ `'a'` *fails* to match against `'x'`, and
 - ▶ the result is a failed match.
- ▶ If `['a', 'b']` is matched against `[undefined, 'x']`, then
 - ▶ attempting to match `'a'` against `undefined` causes the match to *diverge*.

Informal semantics of pattern matching

Example

> ($\backslash \sim(x,y) \rightarrow 0$) undefined

> ($\backslash (x,y) \rightarrow 0$) undefined

Informal semantics of pattern matching

Example

```
> (\ ~ (x,y) -> 0) undefined
```

```
0
```

```
> (\ (x,y) -> 0) undefined
```

```
undefined
```

Informal semantics of pattern matching

Example

> ($\backslash \sim[x] \rightarrow 0$) []

> ($\backslash \sim[x] \rightarrow x$) []

Informal semantics of pattern matching

Example

```
> (\ ~[x] -> 0) []
```

```
0
```

```
> (\ ~[x] -> x) []
```

```
undefined
```

Informal semantics of pattern matching

Example

> ($\backslash \sim[x, \sim(a,b)] \rightarrow x$) [(0,1),undefined]

> ($\backslash \sim[x, (a,b)] \rightarrow x$) [(0,1),undefined]

Informal semantics of pattern matching

Example

```
> (\ ~[x,~(a,b)] -> x) [(0,1),undefined]
(0,1)
> (\ ~[x, (a,b)] -> x) [(0,1),undefined]
undefined
```

Informal semantics of pattern matching

Example

```
> (\ (x:xs) -> x:x:xs) undefined
```

```
> (\ ~(x:xs) -> x:x:xs) undefined
```

Informal semantics of pattern matching

Example

```
> (\ (x:xs) -> x:x:xs) undefined
undefined
> (\ ~(x:xs) -> x:x:xs) undefined
undefined:undefined:undefined
```

Informal semantics of pattern matching

Several species of small furry take functions

```
take1 :: Int -> [a] -> [a]
take1 n _      | n <= 0 = []
take1 _ []      = []
take1 n (x:xs)  = x : take1 (n - 1) xs
```

Informal semantics of pattern matching

Several species of small furry take functions

```
take1 :: Int -> [a] -> [a]
take1 n _      | n <= 0 = []
take1 _ []      = []
take1 n (x:xs)  = x : take1 (n - 1) xs
```

```
> take1 undefined []
```

```
> take1 0 undefined
```

Informal semantics of pattern matching

Several species of small furry take functions

```
take1 :: Int -> [a] -> [a]
take1 n _      | n <= 0 = []
take1 _ []      = []
take1 n (x:xs)  = x : take1 (n - 1) xs
```

```
> take1 undefined []
undefined
> take1 0 undefined
[]
```

Informal semantics of pattern matching

Several species of small furry take functions

```
take2 :: Int -> [a] -> [a]
take2 _ []                = []
take2 n _                 | n <= 0 = []
take2 n (x:xs)            = x : take2 (n - 1) xs
```

Informal semantics of pattern matching

Several species of small furry take functions

```
take2 :: Int -> [a] -> [a]
take2 _ []           = []
take2 n _            | n <= 0 = []
take2 n (x:xs)       = x : take2 (n - 1) xs
```

```
> take2 undefined []
```

```
> take2 0 undefined
```

Informal semantics of pattern matching

Several species of small furry take functions

```
take2 :: Int -> [a] -> [a]
take2 _ []           = []
take2 n _           | n <= 0 = []
take2 n (x:xs)       = x : take2 (n - 1) xs
```

```
> take2 undefined []
[]
> take2 0 undefined
undefined
```

Informal semantics of pattern matching

Several species of small furry take functions

```
> take1 undefined []  
undefined  
> take1 0 undefined  
[]
```

```
> take2 undefined []  
[]  
> take2 0 undefined  
undefined
```

Informal semantics of pattern matching

Several species of small furry take functions

```
take1' :: Int -> [a] -> [a]
take1' n xs      | n <= 0 = seq xs []
take1' _ []      = []
take1' n (x:xs)  = x : take1' (n - 1) xs
```

```
> take1' undefined []
undefined
> take1' 0 undefined
undefined
```

Informal semantics of pattern matching

Several species of small furry take functions

```
take2' :: Int -> [a] -> [a]
take2' n []                = seq n []
take2' n _ | n <= 0 = []
take2' n (x:xs)           = x : take2' (n - 1) xs
```

```
> take2' undefined []
undefined
> take2' 0 undefined
undefined
```

Formal semantics of pattern matching

Sd.

Bibliography



Hudak, Paul, John Peterson, and Joseph H. Fasel (1999).

A Gentle Introduction to Haskell 98.

<https://www.haskell.org/tutorial/>



Marlow, Simon, editor (2010).

Haskell 2010 Language Report.

<https://www.haskell.org/onlinereport/haskell2010/>