Polimorfismo paramétrico e ad hoc

- Polimorfismo paramétrico
  - Operação sobre vários tipos de dados
  - Variáveis de tipo
  - Valor polimórfico
  - Instanciação de variáveis de tipo
  - Funções polimórficas predefinidas

Polimorfismo paramétrico e ad hoc

- Polimorfismo ad hoc
  - Algumas classes de tipo pré-definidas
  - Sobrecarga de literais
  - Conversão entre tipos numéricos

Operação sobre vários tipos de dados

- Algumas funções podem operar sobre vários tipos de dados.
- Por exemplo: a função head recebe uma lista e retorna o primeiro elemento da lista:
  - head [’b’,’a’,’n’,’a’,’n’,’a’] ==> ’b’
  - head [“maria","paula","peixoto"] ==> "maria"
  - head [True,False,True,True] ==> True
  - head [(“ana",2.8),("pedro",4.3)] ==> ("ana",2.8)

Operação sobre vários tipos de dados

- Não importa qual é o tipo dos elementos da lista. A função sempre retorna o primeiro elemento.
- Qual deve ser o tipo de head?
- head :: [Char] -> Char
  head :: [String] -> String
  head :: [Bool] -> Bool
  head :: [(String,Double)] -> (String,Double)
- Como fazer para que head possa tratar vários tipos diferentes?

Variáveis de tipo

- Quando um tipo pode ser qualquer tipo da linguagem, ele é representado por uma **variável de tipo**.
- No exemplo dado, sendo ***a*** o tipo dos elementos da lista que é passada como argumento para a função head, então
- head :: [***a***] -> ***a***
- ***a*** é uma **variável de tipo** e pode ser substituída por qualquer tipo
- variável de tipo é uma variável que serve para representar um tipo

Variáveis de tipo

- O tipo de head estabelece que head recebe uma lista com elementos de um tipo qualquer,
- e retorna um valor deste mesmo tipo.

- Em Haskell, variáveis de tipo devem começar com uma letra minúscula,
- e são geralmente denominadas ***a, b, c, d, etc***.

Valor polimórfico

- Um valor é chamado polimórfico (de muitas formas) se o seu tipo contém uma ou mais variáveis de tipo.
- Por exemplo, o tipo da função head pode ser escrito como
- head :: [*a*] -> *a*
- para qualquer tipo *a*, head recebe uma lista de valores do tipo *a* e retorna um valor do tipo *a*.

Valor polimórfico

- Já o tipo da função length, que recebe uma lista e resulta no tamanho da lista, é dado por:
- length :: [*a*] -> Int
- para qualquer tipo *a*, length recebe uma lista de valores do tipo *a* e retorna um inteiro.

Valor polimórfico

- A função fst (retorna o primeiro valor de um par) é do tipo:
- fst :: (*a*, *b*) -> *a*
- para quaisquer tipos *a* e *b*, fst recebe um par de valores do tipo (*a, b*) e retorna um valor do tipo *a*.

Instanciação de variáveis de tipo

- As variáveis de tipo podem ser instanciadas para diferentes tipos em diferentes circunstâncias.
- Por exemplo, a função length
- length :: [*a*] -> Int
- pode ser aplicada em diferentes tipos listas, como mostra a tabela a seguir:

Instanciação de variáveis de tipo

expressão	valor	instanciação da variável de tipo
`length [False,True]`	2	`a = Bool`
`length "54321"`	5	`a = Char`
`length ["ana","joel","mara"]`	3	`a = String`
`length [("ana",True)]`	1	`a = (String,Bool)`
`length [(&&),(||)]`	2	`a = Bool -> Bool -> Bool`

Funções polimórficas predefinidas

- Muitas das funções definidas no Prelúdio são polimórficas.
- Algumas delas são mencionadas a seguir:

```
id   :: a -> a                -- função identidade
fst  :: (a,b) -> a            -- seleciona o primeiro elemento de um par
snd  :: (a,b) -> b            -- seleciona o segundo elemento de um par
head :: [a] -> a              -- seleciona o primeiro el. de uma lista
tail :: [a] -> [a]            -- seleciona a cauda de uma lista
take :: Int -> [a] -> [a]     -- seleciona os primeiros el. de uma lista
zip  :: [a] -> [b] -> [(a,b)] -- combina duas listas, elemento a elemento
```

- Observe que a lista vazia é polimórfica:
- [] :: [*a*]

Polimorfismo ad hoc

- Alguns tipos possuem operações semelhantes, porém com implementações separadas para cada tipo.
- Por exemplo, a comparação de igualdade pode ser feita entre
  - dois números inteiros,
  - ou dois números racionais,
  - ou dois caracteres,
  - ou dois valores lógicos,
  - entre outros.

Polimorfismo ad hoc

- Para cada tipo dos argumentos deve haver uma implementação da operação.

- Para se ter o benefício de uma **interface** uniforme pode ser desejável que estas operações tenham o mesmo nome.
  - pois as operações são semelhantes entre vários tipos

Polimorfismo ad hoc

- Um mesmo nome de variável ou um mesmo nome de função pode estar associado a mais de um valor em um mesmo escopo de um programa,
- Isso caracteriza o polimorfismo ad hoc
- também chamado de **sobrecarga**.
- Por exemplo, o módulo Prelude apresenta algumas sobrecargas, tais como:

Polimorfismo ad hoc

- O identificador **abs** é sobrecarregado e denota funções que calculam o valor absoluto,
- cujo argumento pode ser de qualquer tipo numérico,
- e cujo resultado é do mesmo tipo que o argumento,

- Tipos numéricos: Int, Integer, Float, etc.

Polimorfismo ad hoc

- O operador (/) é sobrecarregado
- e denota funções de divisão fracionária com dois argumentos de qualquer tipo numérico fracionário,
- e resultado do mesmo tipo dos argumentos.

Polimorfismo ad hoc

- O identificador **pi** é sobrecarregado
- e denota variáveis dos tipos numéricos com representação em ponto flutuante
- cujo valor é uma aproximação de π

Polimorfismo ad hoc

- Para expressar a sobrecarga, Haskell usa classes de tipo.
- Uma classe de tipo é uma coleção de tipos (chamados de instâncias da classe)
- Para os quais é definido um conjunto de funções (aqui chamadas de métodos) que podem ter diferentes implementações, de acordo com o tipo considerado.

Polimorfismo ad hoc

- Uma classe especifica uma interface indicando o nome e a assinatura de tipo de cada função.

- Cada tipo que é instância (faz parte) da classe define (implementa) as funções especificadas pela classe.

Polimorfismo ad hoc

- Por exemplo:
- A classe Num é formada por todos os tipos numéricos e sobrecarrega algumas operações aritméticas básicas, como adição.
- Os tipos Int e Double são instâncias da classe Num.
- Logo existe uma definição da adição para o tipo Int e outra para o tipo Double, usando algoritmos diferentes.

Polimorfismo ad hoc

- Outro exemplo:
- A classe Eq é formada por todos os tipos cujos valores podem ser verificados se são iguais ou diferentes, e sobrecarrega os operadores (==) e (/=).
- Logo para cada instância desta classe existe uma definição destes operadores.
- Todos os tipos básicos apresentados anteriormente são instâncias de Eq.
- Nenhum tipo função é instância de Eq, pois de forma geral não é possível comparar duas funções.

Polimorfismo ad hoc

- Em uma expressão de tipo usamos variáveis de tipo para denotar um tipo qualquer desconhecido, **e um contexto para restringi-las aos tipos que são instâncias de classes específicas**.
- Por exemplo:
- o tipo da função abs é Num *a* => *a* -> *a* , ou seja,
- abs é uma função que recebe um argumento de um tipo *a* e resulta em um valor do mesmo tipo *a*, **sendo *a* qualquer tipo que seja instância da classe Num**
- abs :: Num *a* => *a* -> *a*

Polimorfismo ad hoc

- Outro exemplo:
- O tipo do operador (*) é Num *a* => *a* -> *a* -> *a*
- ou seja, (*) é uma função que recebe dois argumentos de um mesmo tipo *a*
- e resulta em um valor deste mesmo tipo *a*, sendo *a* qualquer tipo que seja instância da classe Num
- (*) :: Num *a* => *a* -> *a* -> *a*

Polimorfismo ad hoc

- Quando uma função sobrecarregada é usada, a escolha da implementação adequada baseia-se nos tipos dos argumentos e do resultado da função no contexto em que ela é usada.
- Semelhantemente quando uma variável sobrecarregada é usada, a escolha da implementação é feita de acordo com o contexto.
- Se o contexto não oferecer informação suficiente pode ser necessário fazer anotações de tipo.

Polimorfismo ad hoc

- Classes de tipos podem ser parecidas com as classes das linguagens orientadas a objetos, mas elas são realmente muito diferentes.
- Elas são mais parecidas com interfaces (como na linguagem Java, por exemplo).
- Pode existir uma hierarquia de classes.
- Se uma classe A possuir uma superclasse B, os tipos que são instâncias de A também devem ser instâncias de B.
- Dizemos também neste caso que A é uma subclasse de B.

Algumas classes de tipo pré-definidas

- Haskell tem várias classes predefinidas e o programador pode definir suas próprias classes.
- Listamos algumas classes a seguir com os seus principais métodos.
- Você não precisa conhecer todas elas.
- Use a informação para consulta quando estiver desenvolvendo em Haskell.
- No entanto, é bom que se familiarize com Eq, Ord, e Num.

Algumas classes de tipo pré-definidas

- Classe Eq: Valores podem ser comparados quanto à igualdade e desigualdade.
- Algumas instâncias:
- Bool, Char, String, Int, Integer, Float, Double, Rational
- Alguns métodos:

(==)	`Eq a => a -> a -> Bool`	igual
(/=)	`Eq a => a -> a -> Bool`	diferente

Algumas classes de tipo pré-definidas

- Classe Ord: Valores podem ser ordenados sequencialmente.
- Superclasses: Eq
- Algumas instâncias:
- Bool, Char, String, Int, Integer, Float, Double, Rational
- Alguns métodos

(<)	`Ord a => a -> a -> Bool`	menor que
(<=)	`Ord a => a -> a -> Bool`	menor ou igual a
(>)	`Ord a => a -> a -> Bool`	maior que
(>=)	`Ord a => a -> a -> Bool`	maior ou igual a
`min`	`Ord a => a -> a -> a`	menor de dois valores
`max`	`Ord a => a -> a -> a`	maior de dois valores

Algumas classes de tipo pré-definidas

- Classe Num: Valores numéricos
- Algumas instâncias:
- Int, Integer, Float, Double, Rational
- Alguns métodos:

`(+)`	`Num a => a -> a -> a`	adição
`(-)`	`Num a => a -> a -> a`	subtração
`(*)`	`Num a => a -> a -> a`	multiplicação
`negate`	`Num a => a -> a`	mudança de sinal
`abs`	`Num a => a -> a`	valor absoluto (módulo)
`signum`	`Num a => a -> a`	sinal (negativo: -1, nulo: 0, positivo: 1)
`fromInteger`	`Num a => Integer -> a`	converte de inteiro

Sobrecarga de literais

- Algumas formas de **literais** são sobrecarregadas: um mesmo literal pode ser considerado de diferentes tipos.
- O tipo usado pode ser decidido pelo contexto em que o literal é usado ou por anotações de tipo.
- Se não for possível determinar o tipo, o compilador escolhe um tipo default.

Sobrecarga de literais

- Literais inteiros
- Podem ser de qualquer tipo numérico (como Int, Integer, Float, Double e Rational).
- Logo o seu tipo mais geral é Num *a* => *a*.
- O tipo default é Integer.

Sobrecarga de literais

- Literais em ponto flutuante
- Podem ser de qualquer tipo numérico fracionário (como Float, Double e Rational).
- Logo o seu tipo mais geral é Fractional *a* => *a*
- O tipo default é Double.

Sobrecarga de literais

- Exemplos
- 187 :: Num *a* => *a*
- 5348 :: Num *a* => *a*
- 3.4 :: Fractional *a* => *a*
- 56.78e13 :: Fractional *a* => *a*

Conversão entre tipos numéricos

- Devido ao sistema de tipo rígido de Haskell, não podemos converter entre os tipos numéricos arbitrariamente.
- Às vezes pode não ser imediatamente claro como converter de um tipo numérico para outro.
- A tabela a seguir, lista funções que podem ser utilizadas para converter entre os tipos mais comuns.

Conversão entre tipos numéricos

	Int	Integer	Rational	Float	Double
Int	id	fromIntegral	fromIntegral	fromIntegral	fromIntegral
Integer	fromIntegral	id	fromIntegral	fromIntegral	fromIntegral
Rational	round	round	id	fromRational	fromRational
Float	round	round	toRational	id	realToFrac
Double	round	round	toRational	realToFrac	id

Tabela 11.1: Funções para conversão entre tipos numéricos.