UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA
ENGENHARIA DA COMPUTAÇÃO

Linguagem de Programação 2019/2

Professor Vitor E Silva Souza

Haskell

Alan Vitorino
Bruno Gomes
Fernando Gavazza
Victor de Oliveira

Haskell

- História
- Conceito
- Amarrações, identificadores, definições e declarações
- Valores e tipos de dados
- Variáveis e constantes, gerência de memória
- Comandos e expressões
- Modularização, TADs, compilação
- Polimorfismo
- Exceções
- Concorrência
- Avaliação

História

Cálculo Lambda λ

- 1930 – Alonzo Church : investigação dos fundamentos da Matemática
- 1935 – Stephen Kleene e J. Barkley Rosser: paradoxo de Kleene-Rosser
- 1936 – Church: isolou, publicou lambda cálculo não-tipada
- 1940 – Church: versão mais fraca, logicamente consistente:
  - Lambda cálculo simplesmente tipado
- Curiosidade: Alonzo Church foi orientador de Alan Turing

Haskell

História

Cálculo Lambda λ

- Haskell Brooks Curry
  - 12 de setembro de 1900, EUA
  - 1 de setembro de 1982 (81 anos)
- Lógica combinatória
- 1942 – Publicação Curry's Paradox
- 1945 – projeto ENIAC
- 1947 – Descreveu uma das primeiras liguagem de programação de alto nível

História

Cálculo Lambda λ

- Entidades
  - podem ser utilizadas como argumento
  - podem ser retornadas como valores de outras funções
- Currying
  - ... é uma técnica para reescrita de funções com múltiplos parâmetros como a composição de funções de um parâmetro.
- f(x,y)= x.x + 2.x.y + 4.y
- ((x → (y → x.x + 2.x.y + 4.y))(x))(y)
- Para x=10 e y=2
- ((x → (y → x.x + 2.x.y + 4.y))(10))(2)
- (y → 100 + 20y + 4y)(2)
  - 100 + 20.2 + 4.2
  - 100+40+8
  - 148

História

Programação Funcional

- LISP 1950's, John McCarthy (MIT)
- SCHEME Tentativa de simplificar e melhorar Lisp
- ML 1970's, Robin Milner, Universidade de Edimburgo
- MIRANDA David Turner (Universidade de Kent)

História

Programação Funcional

- Conferência FPCA ’87
  - Functional Programming Languages and Computer Architecture
  - Comitê
    - Padrão aberto linguagens
    - Base para pesquisas e desenvolvimento
- 1ª reunião: 1988
- Viável para ensino, pesquisa e aplicações
- Ser descritiva referente a sintaxe e semântica
- Não proprietaria
- Senso comum
- Reduzir diversidade desnecessária

História

Programação Funcional

- 1990 – 1ª versão
- 1998 – "Padrão 98" versão mínima, estável, portável
- Compilador Glascow Haskell representando o padrão atual
- Haskell 2010: atualizações incrementais

Conceito

Aspetos Gerais

- Homenagem a Haskell Curry
- Puramente funcional
- + O que?
- - Como?
- Problemas matemáticos
- Mais se realiza pesquisas (dentre LPs funcionais)
- Muito utilizada no meio acadêmico

Conceito

Características

- Rigidamente tipada
- Verificação de tipos em tempo de compilação
- Funções recursivas
- Casamento de padrões
- List Comprehensions
- Guard statements
- Avaliação preguiçosa

Haskell

Amarrações

- Sistema de Tipos **estáticos**
- Todos os tipos são conhecidos em tempo de compilação.
- Existe inferências de tipos.

Haskell
Identificadores

- Haskell permite que um mesmo identificador seja declarado em diferentes partes do programa, possivelmente representando diferentes entidades.
- Os identificadores necessariamente devem começar com uma letra **maiúscula** ou **minúscula** - seguida por uma sequência opcional de letras,dígitos, sublinhas ou apóstrofes.
- Definições de **funções** ou variáveis devem começar com letra **minúscula**.
- **Tipos**, **construtores**, **módulos** e **classes de tipos** têm que começar com letras **maiúsculas**.

Haskell
Declarações e Definições

- **Definições:** produzem amarrações entre identificadores e entidades criadas na própria definição
- **Declarações:** produzem amarrações entre identificadores e entidades já criadas ou ainda por criar
- Não existe **atribuição** em Haskell, e sim **definição**. Uma variável é definida.

Haskell

Definições

- **where:**
- A cláusula where faz definições **locais** à equação
- **let:**
- **let** definições **in** expressão
- Com **where** as definições são colocadas no final, e com **let** elas são colocadas no início

Variáveis

- let pi = 3.141592

Haskell

Tipos de Dados

Primitivo

- Int
- Integer
- Float
- Double
- Bool
- Char (entre ‘’)
- Void
- Unit (implementação do conjunto 1)

Haskell

Tipos de Dados

Caracteres Especiais

'\t'	Tabulação
'\n'	Nova linha
'\' '	Aspas simples (')
'\" '	Aspas duplas (")
'\\'	Barra (\)

Haskell

Tipos de Dados

Compostos

- String (entre “”)
- Tuplas
- Listas

Haskell

Declaração de tipo algébrico

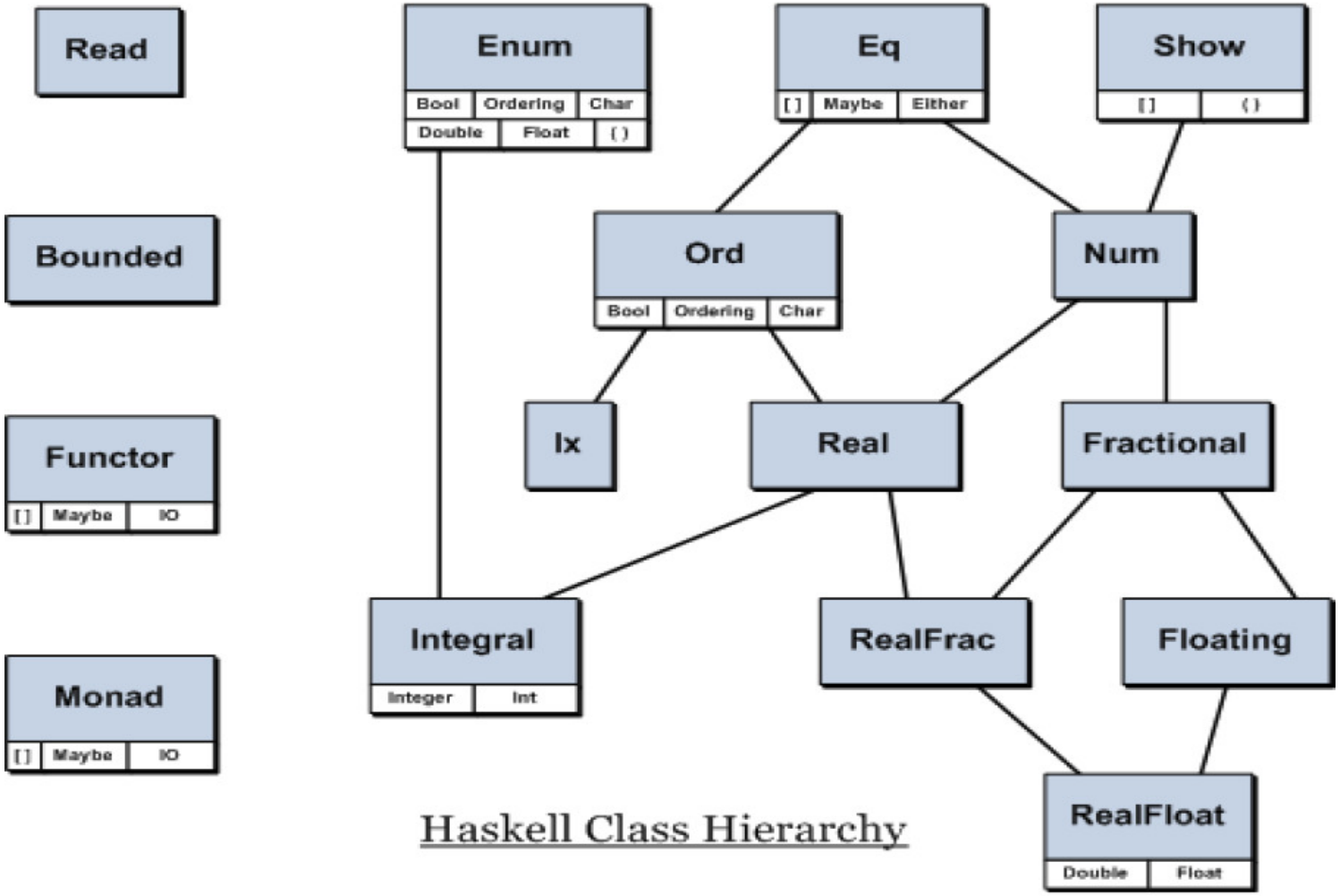
```
data Curso = "Ccomp" | "Engcomp"
Ccomp :: Curso
Engcomp :: Curso


data Ano = Ac Int | Dc Int deriving (Show)
Ac :: Int -> Ano
Dc :: Int -> Ano
```

Haskell

Typeclass

Hierarquia de classes



Haskell Class Hierarchy

Haskell

Typeclass

- Uma Typeclass é como uma interface que define um comportamento. Se um tipo é parte de uma typeclass, quer dizer que ela suporta e implementa o comportamento especificado pela classe de tipo.
- Eq
- Ord
- Show
- Read
- Enum
- Bounded
- Num
- Integral
- Floating

Haskell

Typeclass

```
data Shape = Circle Float Float Float | Rectangle Float Float Float Float deriving (Show)

surface :: Shape -> Float
surface (Circle _ _ r) = pi * r ^ 2
surface (Rectangle x1 y1 x2 y2) = (abs $ x2 - x1) * (abs $ y2 - y1)

ghci> surface $ Circle 10 20 10
314.15927
ghci> surface $ Rectangle 0 0 100 100
10000.0
```

Haskell

Gerência de Memória

- Novos dados são alocados em um "berçário" de 512 kb
- Uma vez que este berçário está esgotado, ele é verificado e valores não usados são liberados
- Os valores sobreviventes são então copiados para a memória principal
- Quanto menos valores alocados, menos trabalho a fazer.

Haskell

Gerência de Memória

- Haskell produz muito lixo (já que os dados não mudam)
- Isso gera uma grande quantidade de dados temporários

- Por padrão, o GHC usa um Garbage Collector por geração
- Por isso, Haskell tem um comportamento contra intuitivo: Quanto maior o percentual de seus valores são lixo, mais rápido ele funciona

Haskell

Comandos e Expressões

Em Haskell um tipo função é escrito usando o operador de tipo →:

t1 → … → tn

t1 até tn-1 são os tipos de argumentos
Tn é o resultado

Int -> Double -> Double -> Bool

Tipo das funções com três argumentos, sendo o primeiro do tipo Int
e os demais do tipo Double, e o resultado do tipo Bool
Variáveis devem começar com letras minúsculas ou sublinhado

Haskell

Comandos e Expressões

Haskell permite:
- Definir tipos de dados pelo usuário;
- Polimorfismo paramétrico;
- Sobrecarga (usando tipos classes);

Erros em Haskell são tratados por exceções.
É case sensitive.
Não possui comandos de repetição como While e For.
Haskell utiliza a ordem normal de avaliação (leftmost-outermost).
Não possui comando de go to.

Haskell

Comandos e Expressões

Palavras reservadas:

case	class	data	deriving	do
else	if	import	in	infix
infixl	infixr	instance	let	of
module	newtype	then	type	where

Haskell

Comandos e Expressões

Operadores

>	Maior
>=	Maior ou igual
==	Igual
/=	Diferente
<	Menor
<=	Menor ou igual
&&	e
||	ou
not	Negação
++	Concatenação

+	Soma
-	Subtração
*	Multiplicação
^	Potência
div	Divisão inteira
mod	Resto da divisão
abs	valor absoluto de um inteiro
negate	troca o sinal do valor
:	Composição de lista

Haskell

Comandos e Expressões

Equivalência Funções Matemáticas

Matematicamente	Haskell
`f(x)`	`f x`
`f(x,y)`	`f x y`
`f(g(x))`	`f (g x)`
`f(x,g(y))`	`f x (g y)`
`f(x)g(y)`	`f x * g y`

Haskell

Comandos e Expressões

Regra de Layout

A regra é basicamente a omissão de "{ }" e ";"

A regra de layout entra em vigor quando existe a omissão de chaves antes de:

- where
- let
- do
- of
- case

```haskell
-- agrupamento implícito

a = b + c
    where
      b = 1
      c = 2

d = a * 2
```

```haskell
-- agrupamento explícito

a = b + c
    where { b = 1 ; c = 2 }

d = a * 2
```

Todas as alternativas devem estar alinhadas.

Haskell

Comandos e Expressões

Em Haskell não existe atribuição, e sim definição. Uma variável é uma coisa. O comando de definição é o sinal " = ". Como é Funcional as funções não há efeitos colaterais.

x = 2

y = "string"

Iterativos: Apresenta iterações com auxílio da recursão.

Como é Funcional as funções não há efeitos colaterais.

Se uma função é chamada duas vezes com os mesmos parâmetros, o resultado retornado por ela será o mesmo.

Haskell

Comandos e Expressões

Ao fazer uma definição de variável ou função, o seu tipo pode ser anotado usando uma assinatura de tipo

```
media2 :: Double -> Double -> Double
media2 x y = (x + y)/2

notaFinal :: Double
notaFinal = media2 4.5 7.2

discriminante :: Double -> Double -> Double -> Double
discriminante a b c = b^2 - 4*a*c

ladosTriangulo :: Float -> Float -> Float -> Bool
ladosTriangulo a b c = a < b + c &&
                       b < a + c &&
                       c < a + b
```

Haskell

Modularização

Subprogramas

- O paradigma de programação funcional enxerga todos os subprogramas como funções que recebem argumentos e retornam soluções simples.

- A solução retornada é baseada inteiramente na entrada e o tempo em que uma função é chamada é irrelevante sendo possível a passagem de uma função como parâmetro.

Haskell

Modularização

TAD

A estrutura sintática e semântica abstrata em Haskell, assim como a forma como se relaciona pode ser dividida em 4 níveis:

Conjunto de módulos
Os módulos oferecem uma maneira de controlar os namespaces

Um módulo é composto de
Uma coleção de declarações
(definições de tipos de dados, classes e tipos de informação)

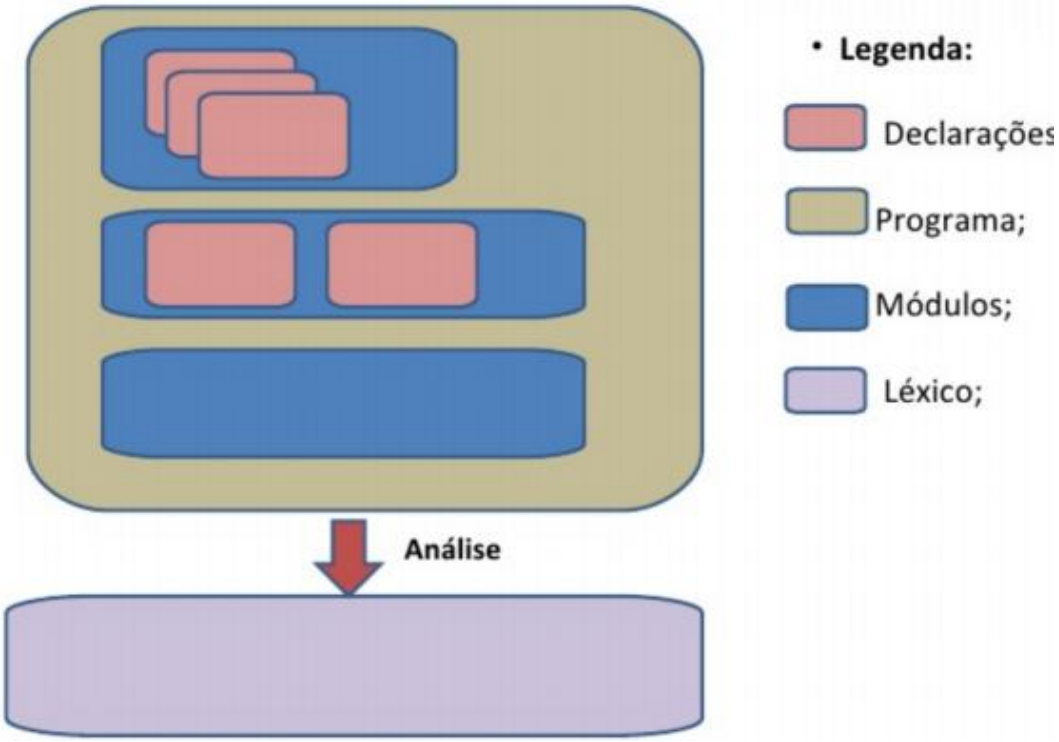
Expressões
Uma expressão denota um valor e tem um tipo estático
Haskell é composto de expressões

Estrutura Léxica
Capta a representação concreta dos programas Haskell em arquivos de texto.

Haskell

Modularização

Estrutura do Programa



Haskell

Modularização

Estrutura do Programa

Os módulos são importados como bibliotecas.

```
import System.Environment
import System.Directory
import System.IO
import Data.List
import Data.List.Split --Biblioteca para usar a função splitOn.
import System.IO.Error
import System.Exit
```

Haskell

Modularização

Estrutura de Dados

Em Haskell uma expressão lista é formada por uma sequência de expressões separadas por vírgula e delimitada por colchetes:

[exp1 , . . . , expn]

onde n >= 0, e exp1, . . ., expn são expressões cujos valores são os elementos da lista.

Um tipo lista é formado pelo tipo dos seus elementos delimitado por colchetes:
[t]

onde e t é o tipo dos elementos da lista.

Haskell

Modularização

Estrutura de Dados

Listas em Haskell

lista	tipo
`['O','B','A']`	`[Char]`
`['B','A','N','A','N','A']`	`[Char]`
`[False,True,True]`	`[Bool]`
`[ [False,True], [], [True,False,True] ]`	`[[Bool]]`
`[1,8,6,10.48,-5]`	`Fractional a => [a]`

Haskell

Modularização

Estrutura de Dados

Operações com listas

`lenght`: calcula o tamanho (quantidade de elementos) de uma lista:

```
Prelude> length [1,2,3,4,5]
5
Prelude> length []
0
```

(!!): seleciona o i-ésimo elemento de uma lista ($0 \leq i < n$, onde n é o comprimento da lista):

```
Prelude> [1,2,3,4,5] !! 2
3
Prelude> [1,2,3,4,5] !! 0
1
```

Haskell

Modularização

Estrutura de Dados

Operações com listas

`null`: verifica se uma lista é vazia:

```
Prelude> null []
True
Prelude> null [1,2,3,4,5]
False
```

`head`: seleciona a **cabeça** (primeiro elemento) de uma lista:

```
Prelude> head [1,2,3,4,5]
1
Prelude> head []
*** Exception: Prelude.head: empty list
```

Haskell

Modularização

Estrutura de Dados

Operações com listas

(++): concatena duas listas:

```
Prelude> [1,2,3] ++ [4,5]
[1,2,3,4,5]
```

`reverse`: inverte uma lista:

```
Prelude> reverse [1,2,3,4,5]
[5,4,3,2,1]
```

`zip`: junta duas listas em uma única lista formada pelos pares dos elementos correspondentes:

```
Prelude> zip ["pedro","ana","carlos"] [19,17,22]
[("pedro",19),("ana",17),("carlos",22)]
```

Haskell

Modularização

Estrutura de Dados

Operações com listas

`take`: seleciona os primeiros *n* elementos de uma lista:

```
Prelude> take 3 [1,2,3,4,5]
[1,2,3]
```

`drop`: remove os primeiros *n* elementos de uma lista:

```
Prelude> drop 3 [1,2,3,4,5]
[4,5]
```

`sum`: calcula a soma dos elementos de uma lista de números:

```
Prelude> sum [1,2,3,4,5]
15
```

Haskell

Modularização

Estrutura de Dados

Pilha

```
module Stack(Stack, push, pop, top, stackEmpty, newStack) where

push       :: a -> Stack a -> Stack a
pop        :: Stack a -> Stack a
top        :: Stack a -> a
stackEmpty :: Stack a -> Bool
newStack   :: Stack a

data Stack a = EmptyStk
             | Stk a (Stack a)

push x s = Stk x s

pop EmptyStk  = error "pop em stack vazia."
pop (Stk _ s) = s

top EmptyStk  = error "top em stack vazia."
top (Stk x _) = x
```

Haskell

Modularização

Estrutura de Dados

Pilha

```
module Main where

import Stack

listTOstack :: [a] -> Stack a
listTOstack []     = newStack
listTOstack (x:xs) = push x (listTOstack xs)

stackTOlist :: Stack a -> [a]
stackTOlist s
    | stackEmpty s = []
    | otherwise    = (top s):(stackTOlist (pop s))

ex1 = push 2 (push 7 (push 3 newStack))
ex2 = push "abc" (push "xyz" newStack)
```

Haskell

Modularização

Estrutura de Dados

Exemplos:

```
*Main> ex1
2|7|3|#
*Main> ex2
"abc"|"xyz"|#
```

```
*Main> listTOstack [1,2,3,4,5]
1|2|3|4|5|#
*Main> stackTOlist ex2
["abc","xyz"]
*Main> stackTOlist (listTOstack [1,2,3,4,5])
[1,2,3,4,5]
```

Haskell

Sistemas de Tipos

Há 3 aspectos interessantes

- Eles são fortes
- Eles são estáticos
- Eles podem ser inferidos automáticamente

[1] *Real World Haskell,* de O'Sullivan & Don Stewart, Chapter 2: Types and Functions

Haskell

Sistemas de Tipos

Tipos Fortes

- Fortemente tipada (*well typed*)
- Sem coerções implícitas
- Coerção por meio de funções explicitamente

Haskell

Sistemas de Tipos

Tipos Fortes

Vantagens

- Capiturar bugs antes de dar problemas

Desvantagens

- Dificulta a criação de certos tipos de códigos (*low-level*)

Haskell

Sistemas de Tipos

Tipos Estáticos

O compilador sabe o valor
de todos os valores e expressões
em tempo de compilação

Há como simular a tipagem dinâmica em Haskell ?

TYPECLASSES

in a safe and convenient form

Haskell possui suporte para prover tipagem dinâmica,
Todavia, não é trivial como nas outras LPs

Haskell

Sistemas de Tipos

Tipos Estáticos

A combinação de ser fortemente tipada com a tipagem estática
redunda na impossibilidade de erros de tipo
em tempo de execução

Um programa Haskell que compila
não vai sofrer erros de tipo
quando rodar

Haskell

Sistemas de Tipos

Inferência de Tipos

O compilador Haskell pode deduzir
automaticamente
o tipo de quase todas as expressões no programa

Variável declarada funciona como variável global e constante

Haskell

Sistemas de Tipos

Polimorfismo

Type variable

x :: Float

- x = 3,25
- x = 1,11115
- x = 0,12
- x = 101,4

f :: a -> a-> a

- f = Float -> Floar -> Floar
- f = Int -> Int -> Int
- f = Pessoa -> Pessoa -> Pessoa
- f = [Ponto] ->[Ponto] -> Ponto

FONTE: https://www.schoolofhaskell.com/school/starting-with-haskell/introduction-to-haskell/5-type-classes

Haskell

Sistemas de Tipos

Polimorfismo

Num , Eq , Ord , and Show are type classes, and we say that (==) , (<) , and (+) are "typeclass polymorphic".
Intuitively, type classes correspond to sets of types which have certain operations defined for them, and type class polymorphic functions work only for types which are instances of the type class(es) in question

FONTE: https://www.schoolofhaskell.com/school/starting-with-haskell/introduction-to-haskell/5-type-classes

Haskell

Sistemas de Tipos

Polimorfismo

```
class Eq a where
    (==) :: a -> a -> Bool
    (/=) :: a -> a -> Bool
```

Eq is declared to be a type class with a single parameter, a .
Any type a which wants to be an instance of Eq
must define two functions, (==) and (/=),
with the indicated type signatures

FONTE: https://www.schoolofhaskell.com/school/starting-with-haskell/introduction-to-haskell/5-type-classes

Haskell

Sistemas de Tipos

Polimorfismo

```
class Eq a where
    (==) :: a -> a -> Bool
    (/=) :: a -> a -> Bool
```

Todo tipo a que queira ser instância de Eq
devem definir duas funções

```
(==) :: Int -> Int -> Bool
(/=) ::  Int -> Int -> Bool
```

FONTE: https://www.schoolofhaskell.com/school/starting-with-haskell/introduction-to-haskell/5-type-classes

Haskell

Sistemas de Tipos

Polimorfismo

Ad Hoc

Haskell realiza polimorfismo Ad Hod
via sistema de type class e
instancias de classes

Haskell

Sistemas de Tipos

Polimorfismo

Ad Hoc Coerção

Explicita em tempo de compilação

Haskell

Sistemas de Tipos

Polimorfismo

Ad Hoc Coerção

```
soma a b = a + b

somaFloat :: Float -> Float -> Float
somaFloat a b = a + b

x = 1
y = 2
z = 3
xF = 1
yF = 2
zF = 3
xF :: Float
yF :: Float
zF :: Float
```

Haskell

Sistemas de Tipos

Polimorfismo

Ad Hoc Coerção

- Ok, one module loaded.
- *Main> soma x y
- 3
- *Main> soma xF yF
- 3.0
- *Main> somaFloat xF yF
- 3.0
- *Main> somaFloat x y
- <interactive>:4:11: error:
- * Couldn't match expected type `Float' with actual type `Integer'
- * In the first argument of `somaFloat', namely `x'
- In the expression: somaFloat x y
- In an equation for `it': it = somaFloat x y

Haskell

Sistemas de Tipos

Polimorfismo

Ad Hoc Coerção

- *Main> soma x yF
- * Couldn't match expected type `Integer' with actual type `Float'
- *Main> somaFloat x yF
- * Couldn't match expected type `Float' with actual type `Integer'
- *Main> x::Float
- * Couldn't match expected type `Float' with actual type `Integer'
- *Main> w = 1
- *Main> soma x w
- 2
- *Main> soma xF w
- 2.0
- *Main> w::Float
- 1.0
- *Main> soma x w
- 2
- *Main> :t w
- w :: Num p => p

Haskell

Sistemas de Tipos

Polimorfismo

Ad Hoc Coerção

Many languages provide some form of coercion polymorphism; one example is automatic conversion between integers and floating-point numbers. Haskell ***deliberately avoids*** *even this kind of simple automatic Coercion*

[1] *Real World Haskell,* de O'Sullivan & Don Stewart, Chapter 2: Types and Functions, pág 38.

Haskell

Sistemas de Tipos

Polimorfismo

Ad Hoc Sobrecarga

A sobrecarga acontece quando
a classe de tipo (type class)
é instancia de uma outra classe e
herda as opercações nela definida

Haskell

Sistemas de Tipos

Polimorfismo

Universal

Parametric polymorphism refers to
when the type of a value contains one or More
(unconstrained) type variables,
so that the value may adopt any type that
results from substituting those variables with concrete types.

FONTE: https://wiki.haskell.org/Polymorphism

Haskell

Sistemas de Tipos

Polimorfismo

Universal Paramétrico

identidade :: **a** -> **a**
contains an unconstrained type variable **a**
in its type, and so can be used in a context requiring
Char -> Char or
Integer -> Integer or
(Bool -> Maybe Bool) -> (Bool -> Maybe Bool).

Haskell

Sistemas de Tipos

Polimorfismo

Universal Paramétrico

```
{- FUNCAO IDENTIDADE -}
Identidade :: a-> a
Identidade x = x
```

- ghci> identidade ‘$’
- ‘$’
- ghci> identidade 1.414213562
- 1.414213562
- ghci> identidade “UFES”
- “UFES”
- ghci> identidade [1.. 5]
- [1, 2, 3, 4, 5]
- ghci> :t identidade
- identidade :: a -> a

Haskell

Sistemas de Tipos

Polimorfismo

Universal Paramétrico

```
lenght [a] -> Int
fst (a,b) -> a
head [a] -> a
zip [a] [b] -> [(a,b)]
```

Haskell

Sistemas de Tipos

Polimorfismo

Universal Inclusão

Não há.
O polimorfismo por inclusão é característico
de linguagens orientadas a objetos.

F. M. Varejão. **Linguagens de programação: Java, C e C++ e outras: conceitos e técnicas**. Pág. 232

Haskell

Sistemas de Tipos

Polimorfismo

There are several more exotic flavours of polymorphism
that are implemented in some extensions to Haskell,
There are some kinds of polymorphism that Haskell doesn't support,
or at least not natively

FONTE: https://wiki.haskell.org/Polymorphism

Exceções

Os dois tratadores de erro mais usados

- Pure error handling
  - Algoritmos que não requerem nada dos "IO monads"
  - Tratadores expressos por sistema de tipos de dados
- Exception
  - Pode ser lançados para qualque lugar due to complexities of lazy evaluation
  - Mas capturados apenas por IO monad

FONTE:[1] *Real World Haskell,* de O'Sullivan & Don Stewart, Chapter 19, pág 447.

Exceções

Pure error handling

```
divBy :: Integral a => a -> [a] -> [a]
divBy numerator = map (numerator `div`)
```

- ghci> **divBy 50 [1,2,5,8,10]**
- [50,25,10,6,5]
- ghci> **take 5 (divBy 100 [1..])**
- [100,50,33,25,20]
- ghci> **divBy 50 [1,2,0,8,10]**
- [50,25,*** Exception: divide by zero

Exemplo de avaliação tardia

FONTE:[1] *Real World Haskell*, de O'Sullivan & Don Stewart, Chapter 19, pág 447.

Haskell

Exceções

Pure error handling : Uso do Maybe

```
divBy :: Integral a => a -> [a] -> Maybe [a]
divBy _ [] = Just []
divBy _ (0:_) = Nothing
divBy numerator (denom:xs) =
        case divBy numerator xs of
                Nothing -> Nothing
                Just results -> Just ((numerator `div` denom) : results)
```

- ghci> **divBy 50 [1,2,5,8,10]**
- Just [50,25,10,6,5]
- ghci> **divBy 50 [1,2,0,8,10]**
- Nothing

FONTE:[1] *Real World Haskell,* de O'Sullivan & Don Stewart, Chapter 19, pág 447.

Exceções

Exception

Em Haskell, graças às monads e os tipos Either and Maybe podemos alcançar os mesmo efeitos em código puro sem a necessidade de usar exceções e tratadores de exceções.

Exceções podem ser lançadas em qualquer lugar do programa
Todavia, devido a inespecífica ordem de avaliação, elas só podem ser capturadas numa IO monad.

Tratadores de exceção Haskell não envolvem sintaxes especiais como Python ou Java.
Os mecanismos de captura de tratamento de exceção são - surpresa - funções.

FONTE:[1] *Real World Haskell,* de O'Sullivan & Don Stewart, Chapter 19, pág 554.

Exceções

Exception

- ghci> **:m Control.Exception**
- ghci> **let x = 5 `div` 0**
- ghci> **let y = 5 `div` 1**
- ghci> **print x**
- *** Exception: divide by zero
- ghci> **print y**
- 5
- ghci> **try (print x)**
- Left divide by zero
- ghci> **try (print y)**
- 5
- Right ()

FONTE:[1] *Real World Haskell,* de O'Sullivan & Don Stewart, Chapter 19, pág 554.

Haskell

Concorrência

A parallel program is one that uses a multiplicity of
computational hard-ware
(e.g. multiple processor cores)
in order to perform computation more quickly.

concurrency is a program-structuring technique in which
there are multiple threads of control
Notionally the threads of control execute “at the same time";
that is, the user sees their effects interleaved.

FONTE: [2]Parallel and Concurrent Programming in Haskell. V1.2 Simon Marlow

Haskell

Concorrência

Em Haskell, a thread é uma ação de IO e
executa independentemente de outras threads
Para criar uma thread,
importamos o módulo **Control.Concurrent** e
usamos a função **forkIO**

```
ghci> :m +Control.Concurrent
ghci> :t forkIO
forkIO :: IO () -> IO ThreadId
ghci> :m +System.Directory
ghci> forkIO (writeFile "xyzzy" "seo craic nua!") >> doesFileExist "xyzzy"
False.
```

FONTE:[1] *Real World Haskell,* de O'Sullivan & Don Stewart, Chapter 24, pág 532.

Haskell

Concorrência

Comunicação Simples Entre Threads

"The simplest way to share information between two threads is to let them both use a variable. Because Haskell data is immutable by default, this poses no risks: neither thread can modify the other's view of the file's name or contents."

We often need to have threads actively communicate with each other. For example, GHC does not provide a way for one thread to find out whether another is still executing, has completed, or has crashed.* However, it provides a synchronizing variable type

Haskell

Concorrência

Comunicação Simples Entre Threads

MVar: *synchronizing variable* type

An MVar acts like a single-element box:
it can be either full or empty
We can put something into the box,
making it full, or take something out, making it empty

```
ghci> :t putMVar
putMVar :: MVar a -> a -> IO ()
ghci> :t takeMVar
takeMVar :: MVar a -> IO a
```

Haskell

Concorrência

Comunicação canal via única

Chan: variable type
Prove um canal de comunicação de via única.

Haskell

Concorrência

Deadlock

… situation, two or more threads get stuck forever
in a clash over access to shared resources

One classic way to make a multithreaded program deadlock is to forget the order in which we must acquire locks. This kind of bug is so common, it has a name: *lock order inversion*. While Haskell doesn't provide locks, the **MVar** type is prone to the order inversion problem.

Haskell

Concorrência

Starvation

The nonstrict (lazy evaluation) nature of the **MVar** type can either cause or exacerbate a starvation problem.

strict nature refere-se à funções ao qual os parâmetros devem ser completamente avaliados antes de serem chamados

Haskell

Concorrência

Paralelismo

Normal Form and Head Normal Form

The familiar **seq** function evaluates an expression to what we call head normal form (HNF)
It stops once it reaches the outermost constructor (the head)
This is distinct from normal form (NF),
in which an expression is completely evaluated.

FONTE:[1] *Real World Haskell,* de O'Sullivan & Don Stewart, Chapter 24, pág 545.

Haskell

Concorrência

Paralelismo

```
sort :: (Ord a) => [a] -> [a]
sort (x:xs) = lesser ++ x:greater
            where  lesser   = sort [y | y <- xs, y  <  x]
                   greater = sort [y | y <- xs, y >= x]
sort _ = []
```

```
module Sorting where
import Control.Parallel (par, pseq)
parSort :: (Ord a) => [a] -> [a]
parSort (x:xs) = force greater `par` (force lesser `pseq` (lesser ++ x:greater))
            where     lesser =    parSort [y | y <- xs, y  <  x]
                      greater = parSort [y | y <- xs, y >= x]
parSort _ = []
```

FONTE:[1] *Real World Haskell*, de O'Sullivan & Don Stewart, Chapter 24, pág 545.

Avaliação

Comparação entre C, C++, Java e Haskell

CRITÉRIOS GERAIS	C	C++	JAVA	HASKELL
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Parcial
Eficiência	Sim	Sim	Sim	Parcial
Portabilidade	Não	Não	Sim	Não
Método de projeto	Estruturado	Estruturado e OO	OO	Funcional

Avaliação

Comparação entre C, C++, Java e Haskell

CRITÉRIOS GERAIS	C	C++	JAVA	HASKELL
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confi				
Apre				
Eficiência	Sim	Sim	Sim	Parcial
Portabilidade	Não	Não	Sim	Não
Método de projeto	Estruturado	Estruturado e OO	OO	Funcional

Não há recursos nativos para controlar o hardware.
Não apta para problemas que envolvem muitas variáveis (banco de dados).
Uso intensivo de memória para empilhamento de chamadas recursivas.
Favorece a construção de códigos bastante compactos comparativamente.
Fácil manutenção graças à redigibilidade, legibilidade, modularidade.

Avaliação

Comparação entre C, C++, Java e Haskell

CRITÉRIOS GERAIS	C	C++	JAVA	HASKELL
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Sim
Aprendizado				
Eficiência				
Portabilidade	Não	Não	Sim	Não
Método de projeto	Estruturado	Estruturado e OO	OO	Funcional

Tipagem forte e estática em tempo de compilação, ausência de manipulação de espaços de memória, ausência de mudança de estado de variáveis: redundam em drástica redução na ocorrência de erros.

Avaliação

Comparação entre C, C++, Java e Haskell

CRITÉRIOS GERAIS	C	C++	JAVA	HASKELL
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Parcial
Eficiênc				
Portabilid				
Método projeto	Estruturado	Estruturado e OO	OO	Funcional

Apesar da programação funcional livrar o usuário de recursos avançados e perigosos (e.g. manipulação de ponteiros), Haskell apresenta o desafio de o usuário defrontar-se com novos conceitos e estética que convidam o novo usuário a abstrair-se em um novo mundo de entendimento.
A elegância conceitual da linguagem favorece na construção de códigos legíveis, compactos, de boa leitura.

Haskell

Avaliação

Comparação entre C, C++, Java e Haskell

CRITÉRIOS GERAIS	C	C++	JAVA	HASKELL
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Parcial
Eficiência	Sim	Sim	Sim	Parcial
Portabilidade				
Método de projeto				

Muitos aspectos são transferidos para a responsabilidade de sistemas automáticos (e.g. coletor de lixo, verificação estatica de tipos). Todavia, a ausência de efeitos colaterais, a avaliação preguiçosa, rigidez de tipos conferem códigos e processamento mais estáveis e confiáveis.

Avaliação

Comparação entre C, C++, Java e Haskell

CRITÉRIOS GERAIS	C	C++	JAVA	HASKELL
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Sim
Aprendizado				
Eficiência				
Portabilidade	Não	Não	Sim	Não
Método de projeto	Estruturado	Estruturado e OO	OO	Funcional

Haskell é compilado para plataformas específicas que exigem que se tenha um compilador próprio (GHC). Não há muita variação significatica de compiladores Haskell uma vez que a linguagem nasceu com o objetivo de unifcar as linguagens funcionais da época.

Avaliação

Comparação entre C, C++, Java e Haskell

CRITÉRIOS GERAIS	C	C++	JAVA	HASKELL
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Parcial
Eficiência	Sim	Sim	Sim	Parcial
Portabilidade	Não	Não		
Método de projeto	Estruturado	Estruturado e OO	OO	Funcional

Haskell é funcional.

Haskell

Avaliação

Comparação entre C, C++, Java e Haskell

CRITÉRIOS GERAIS	C	C++	JAVA	HASKELL
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade				
Integração				
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta	Baixo
Escopo	Sim	Sim	Sim	Sim
Expressões e comandos	Sim	Sim	Sim	Parcial

Haskell favorece a produção de códigos reduzidos, concisos e elegantes, de fácil legibilidade. Modularização em bibliotecas compactas. Bibiotecas simples de serem usadas.

Avaliação

Comparação entre C, C++, Java e Haskell

CRITÉRIOS GERAIS	C	C++	JAVA	HASKELL
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração				
Custo	ferramenta	ferramenta	ferramenta	Baixo
Escopo	Sim	Sim	Sim	Sim
Expressões e comandos	Sim	Sim	Sim	Parcial

Haskell oferece o reuso de funções, tipos de dados, classes de tipos e constantes, que distribuem-se em módulos.

Avaliação

Comparação entre C, C++, Java e Haskell

CRITÉRIOS GERAIS	C	C++	JAVA	HASKELL
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	Sim
Custo				
Escopo				
Expressões e comandos				

[8] Haskell's built-in features for interfacing to other languages are formally defined in the Foreign Function Interface (FFI).
The tools and libraries either simplify interfacing to C or allow interfacing to other languages and environments (COM, JVM, Python, Tcl, Lua...)

Avaliação

Comparação entre C, C++, Java e Haskell

CRITÉRIOS GERAIS	C	C++	JAVA	HASKELL
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta	Baixo
Escopo				
Expressões e comandos				

Haskell possui várias bibliotecas e ferramentas gratuitas de desenvolvimento. A filosofia da linguagem baseia-se em ser não proprietária.

Haskell

Avaliação

Comparação entre C, C++, Java e Haskell

CRITÉRIOS GERAIS	C	C++	JAVA	HASKELL
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração				
Custo				
Escopo	Sim	Sim	Sim	Sim
Expressões e comandos	Sim	Sim	Sim	Parcial

Haskell possui escopo em nível de módulos. Não há aninhamento dentro de funções.

Avaliação

Comparação entre C, C++, Java e Haskell

CRITÉRIOS GERAIS	C	C++	JAVA	HASKELL
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	Parcial
Custo				
Escopo				
Expressões e comandos	Sim	Sim	Sim	Parcial

Ampla variedade de expressões e comandos, muitas com abordagem conceitual e estética diferentes.

Haskell

Avaliação

Comparação entre C, C++, Java e Haskell

CRITÉRIOS GERAIS	C	C++	JAVA	HASKELL
Tipos primitivos e compostos	Sim	Sim	Sim	Sim
Gerenciamento de memória	Este conceito não é desenvolvido em Haskell de forma nativa. Definição na FFI standard.Biblioteca básica de Haskell Foreign.C.Types. Haskell prove definição de tipos			
Persistência de dados				
Passagem de parâmetros	Lista variável e por valor	Lista variável, default, por valor e por referencia	Lista varipavel, por calor e por cópia de referência	Por valor
Encapsulamento e proteção	Parcial	Sim	Sim	Sim
Sistema de tipos	Não	Parcial	Sim	

Avaliação

Comparação entre C, C++, Java e Haskell

CRITÉRIOS GERAIS	C	C++	JAVA	HASKELL
Tipos primitivos e compostos	Sim	Sim	Sim	Sim
Gerenciamento de memória	Programador	Programador	Sistema	Sistema
Persistência de dados	Coletor de lixo			
Passagem de parâmetros	Lista variável e por valor	Lista variável, default, por valor e por referencia	Lista varipavel, por calor e por cópia de referência	
Encapsulamento e proteção	Parcial	Sim	Sim	Sim
Sistema de tipos	Não	Parcial	Sim	

Avaliação

Comparação entre C, C++, Java e Haskell

CRITÉRIOS GERAIS	C	C++	JAVA	HASKELL
Tipos primitivos e compostos	Sim	Sim	Sim	Sim
Gerenciamento de memória	Programador	Programador	Sistema	Sistema
Persistência de dados	Biblioteca de funções	Biblioteca de classes e funções	JDB, biblioteca de classes, serialização	Frameworks DataBase
Passagem de parâmetros				
Encapsulamento e proteção				
Sistema de tipos	Não	Parcial	Sim	Sim

Haskel possui vários frameworks de banco de dados diferentes disponíveis. HDBC – Haskell DataBase Connectivity System

Avaliação

Comparação entre C, C++, Java e Haskell

CRITÉRIOS GERAIS	C	C++	JAVA	HASKELL
Tipos primitivos e compostos	Sim	Sim	Sim	Sim
Gerenciamento de memória	Programador	Programador	Sistema	Sistema
Persistência de dados	Biblioteca de funções	Biblioteca de classes e funções	JDB, biblioteca de classes, serialização	
Passagem de parâmetros	Lista variável e por valor	Lista variável, default, por valor e por referencia	Lista varipavel, por calor e por cópia de referência	Por valor
Encapsulamento e proteção	Passagem por valor com avaliação preguiçosa.			
Sistema de tipos	Não	Parcial	Sim	Sim

Avaliação

Comparação entre C, C++, Java e Haskell

CRITÉRIOS GERAIS	C	C++	JAVA	HASKELL
Tipos primitivos e compostos	Sim	Sim	Sim	Sim
Gerenciamento de memória	Programador	Programador	Sistema	Sistema
Persistência de dados	Biblioteca de funções	Biblioteca de classes e funções	JDB, biblioteca de classes, serialização	
Passagem de parâmetros	Haskell proporciona encapsulamento de dados dentro dos módulos.			
Encapsulamento e proteção	Parcial	Sim	Sim	Sim
Sistema de tipos	Não	Parcial	Sim	Sim

Avaliação

Comparação entre C, C++, Java e Haskell

CRITÉRIOS GERAIS	C	C++	JAVA	HASKELL
Tipos primitivos e compostos	Sim	Sim	Sim	Sim
Gerenciamento de memória	Programador	Programador	Sistema	Sistema
Persistência de dados	Biblioteca de funções	Biblioteca de classes e funções	JDB, biblioteca de classes, serialização	
Passagem de parâmetros				
Encapsulamento e proteção				
Sistema de tipos	Não	Parcial	Sim	Sim

Haskell possui um complexo sistema de tipos ao qual podem ser criados em forma de hierarquia e de forma a garantir polimorfismo dentro da cadeia de herança. Tipos compostos: lista e tupla.

Haskell

Avaliação

Comparação entre C, C++, Java e Haskell

CRITÉRIOS GERAIS	C	C++	JAVA	HASKELL
Verificação de tipos	Estática	Estática / dinâmica	Estática / dinâmica	Estática
Polimorfismo	Todas as verificações de Haskell são estáticas.			
Exceções	Não	Parcial	Sim	
Concorrência	Não (biblioteca de funções)	Não (biblioteca de funções)	Sim	Sim

Haskell

Avaliação

Comparação entre C, C++, Java e Haskell

CRITÉRIOS GERAIS	C	C++	JAVA	HASKELL
Verificação de tipos	Estática	Estática / dinâmica	Estática / dinâmica	Estática
Polimorfismo	Coerção e sobrecarga	Todos	Todos	Paramétrico e sobrecarga
Exceções	Paramétrico e sobrecarga.			
Concorrência	Não (biblioteca de funções)	Não (biblioteca de funções)	Sim	

Haskell

Avaliação

Comparação entre C, C++, Java e Haskell

CRITÉRIOS GERAIS	C	C++	JAVA	HASKELL
Verificação de tipos	Estática	Estática / dinâmica	Estática / dinâmica	Estática
Polimorfismo	Coerção e sobrecarga	Todos	Todos	Paramétrico e sobrecarga
Exceções	Não	Parcial	Sim	Sim
Concorrência				

Oferece mecanismos de exceções por meio de tipos de classe e por meio de funcionalidades de módulos.

Haskell

Avaliação

Comparação entre C, C++, Java e Haskell

CRITÉRIOS GERAIS	C	C++	JAVA	HASKELL
Verificação de tipos	Estática	Estática / dinâmica	Estática / dinâmica	Estática
Polimorfismo	Coerção e sobrecarga	Todos	Todos	Paramétrico e sobrecarga
Exceções	Não	Parcial	Sim	Sim
Concorrência	Não (biblioteca de funções)	Não (biblioteca de funções)	Sim	Sim

Bibliotecas de suporte à concorrência e threads.

OBRIGADO

BIBLIOGRAFIA

01 - ***Real World Haskell***, de O'Sullivan & Don Stewart 1ºed, novembro de 2008
02 - ***Learn You Haskell for Great Good!ell***. Miran Lipovaca
03 - ***Parallel and Concurrent Programming in Haskell***. V1.2 Simon Marlow
04 - ***Yet Another Haskell Tutorial***, Hal Daumé III
05 - https://wiki.haskell.org/Functional_programming
06 - https://wiki.haskell.org/All_About_Monads
07 - https://www.schoolofhaskell.com/school/starting
08 - https://wiki.haskell.org/Applications_and_libraries/Interfacing_other_languages
09 - ***Linguagens de programação: Java, C e C++ e outras***. F. M. Varejão.
10 - http://haskell.tailorfontela.com.br/input-and-output
11 - (https://www.schoolofhaskell.com/school/starting-with-haskell/introduction-to-haskell/5-type-classes