Seminário Haskell - Tópicos / Roteiro / Referências

Caio Diniz

Giuseppe Cordeiro

Vinícius Miranda

TÓPICOS:

1. Introdução
2. Cronologia
3. Paradigmas
4. Características principais da linguagem
5. Amarrações
6. Valores
    a. Primitivos
    b. Compostos
    c. TypeClass
7. Funções
8. Linguagens Relacionadas
9. Ambiente de Desenvolvimento, Compilador e Código

ROTEIRO:

Abertura: // vinshow

Bom dia, hoje vamos apresentar sobre a linguagem Haskell!

Sumário: // vinishow

Primeiramente, abordaremos os tópicos: uma breve introdução, a cronologia ou linha do tempo, os paradigmas da linguagem, características e desenvolvimento que seria como usar.

Introdução: // vinishow

Vamos começar com uma breve introdução sobre o Haskell.

Haskell é uma linguagem funcional, o que significa que sua abordagem à programação é bem diferente das linguagens imperativas mais comuns. Ao invés de focar em como as coisas devem ser feitas, Haskell se preocupa mais em descrever o que deve

ser feito. Isso a torna uma linguagem declarativa e extremamente poderosa quando o objetivo é alcançar maior produtividade e clareza no código.

Haskell é baseada no cálculo lambda, uma teoria matemática desenvolvida nos anos 1930 por Alonzo Church, que também inspirou o símbolo da linguagem. Atualmente, Haskell é uma das linguagens funcionais mais pesquisadas e adotadas em projetos acadêmicos e comerciais.

Haskell foi criada em 1987 como uma evolução da linguagem funcional Miranda, com o intuito de fornecer uma ferramenta mais avançada e flexível. Desde o início, ela foi projetada para priorizar a produtividade, clareza e a manutenibilidade de código.

Timeline: // vinishow

Influências:

É interessante conhecer as influências históricas que ajudaram a moldar a linguagem. Tudo começou em 1930, quando Alonzo Church desenvolveu o cálculo lambda, que viria a ser a base teórica das linguagens funcionais. Mais tarde, em 1950, John McCarthy utilizou essa teoria para criar o Lisp, a primeira linguagem de programação funcional. Em 1970, foi desenvolvida a linguagem ML, que introduziu conceitos importantes como a inferência de tipos e tipos polimórficos, características que também influenciariam diretamente o Haskell.

Timeline:

Em **1987**, quando uma comunidade de programação funcional decidiu criar uma linguagem padronizada e semântica não rígida. A linguagem foi nomeada em homenagem ao lógico Haskell Brooks Curry.

Em **1990**, a primeira versão foi definida.

Em **1999**, Haskell 98, uma versão estável e portável da linguagem, é lançada.

Em **2003**, Haskell 98 foi revisado.

Em **2006**, começou o processo de definição de um sucessor do padrão 98, informalmente conhecido como Haskell′ ("Haskell Prime").

Infográfico:

Aqui temos um infográfico, apresentando a história da linguagem.

(passar rapidamente só pontuando o que foi falado)

Paradigmas: // caio

Funcional:

Haskell segue o paradigma funcional, que é significativamente diferente do paradigma mais comum, o imperativo. No paradigma funcional, como em Haskell, não há atribuições, que são comuns em linguagens imperativas. A ausência de atribuições é um dos motivos pelos quais o paradigma funcional é considerado mais seguro, já que em linguagens imperativas, muitos bugs são causados por atribuições incorretas ou inesperadas. Em vez disso, tudo em Haskell gira em torno de funções, que recebem um número qualquer de argumentos e produzem uma saída com base nesses argumentos.

Uma característica interessante do paradigma funcional é que as funções podem ser definidas a partir de outras funções. Quando você desenvolve um programa em Haskell, você está, na verdade, criando uma grande função que recebe entradas do usuário e devolve uma saída. Essa abordagem leva a uma modularização natural, mas de uma maneira diferente do paradigma imperativo. Enquanto no imperativo você usa regras de escopo e compilações separadas para modularizar, no funcional você divide um problema em subproblemas menores, e as soluções dessas partes são combinadas para resolver o problema original.

Outro ponto, é o lazy evaluation e o high-order functions, mas explicaremos melhor depois;

Modularização:

O paradigma de modularização em Haskell é uma abordagem que enfatiza a construção de programas através da divisão de problemas complexos em partes menores e mais manejáveis, chamadas módulos. Em Haskell, a modularização acontece de maneira natural, com foco na criação de funções reutilizáveis e independentes. Cada função resolve um subproblema, e suas soluções são combinadas para formar a solução final.

Essa abordagem difere do paradigma imperativo, onde a modularização muitas vezes depende de regras de escopo e separação de blocos de código. Em Haskell, a modularização é feita através de funções que podem ser facilmente reutilizadas em diferentes contextos, promovendo maior clareza e manutenção. Além disso, o uso de módulos permite que você controle quais partes do código

devem ser exportadas ou mantidas privadas, facilitando a organização e evitando conflitos no programa.

Características principais: // caio

Lazy Evaluation:

Não existe ordem pré-definida para execução das subfunções dentro de uma função. Portanto, significa que as expressões são avaliadas apenas quando seus resultados são necessários, sem uma ordem pré-definida para a execução das subfunções.

Polimorfismo Universal:

Permite que funções sejam definidas de forma a operar em qualquer tipo de dado.

Função de Ordem Superior:

Permite que funções sejam tratadas como valores, o que significa que podem ser passadas como argumentos ou retornadas de outras funções. Isso facilita a abstração e a composição de funções, promovendo um estilo de programação mais funcional e declarativo

Estrutura de Dados de Tamanho Infinito:

A estrutura de dados de tamanho infinito é uma característica única do Haskell, possibilitada pela sua Lazy Evaluation. Isso permite que listas e outras estruturas de dados sejam definidas de forma que possam conter um número infinito de elementos, sendo que os valores são gerados apenas conforme necessário

Amarrações:// gius

Haskell apresenta um sistema de tipos estático, o que significa que todos os tipos de dados são conhecidos no momento da compilação. Isso garante maior segurança e minimiza erros durante a execução. Uma das vantagens de Haskell é a inferência de tipos, onde o compilador consegue deduzir automaticamente o tipo de um identificador, sem que o programador precise especificá-lo explicitamente.

Além disso, Haskell permite que um mesmo identificador seja utilizado em diferentes partes do programa para representar entidades distintas, desde que o contexto seja claro. Outra característica importante é que, em Haskell, as conversões de tipos são sempre

explícitas. Isso reforça a segurança, evitando conversões implícitas que possam causar comportamentos inesperados no código.

Valores: // gius

Em Haskell, os tipos de dados podem ser classificados em primitivos e compostos (ou mais complexos). Vamos explorar cada um deles:

Tipos Primitivos:

Possui os valores conhecidos na programação, integer, float, double, char e bool. Além disso, possui um tipo especial chamado de Unit que possui apenas um valor, (). Ele é usado em situações em que o valor de retorno não é importante.

Tipos Compostos:

E também tem os seguintes tipos compostos:

Listas: Haskell oferece listas homogêneas, ou seja, listas que contêm elementos de um mesmo tipo.

Tuplas (Tuples): São coleções de valores que podem ter diferentes tipos, com um número fixo de elementos.

Funções: Funções são tipos compostos em Haskell e podem ser vistas como mapeamentos entre tipos. O tipo de uma função que recebe um Int e retorna um Bool é representado como Int -> Bool.

Funções: // gius

Funções Puras

A maioria das funções em Haskell são puras, ou seja, para um dado conjunto de entradas, elas sempre retornam o mesmo resultado e não causam efeitos colaterais.

Funções de Ordem Superior

Funções que podem receber outras funções como parâmetros ou retornar funções como resultado.

Funções Parciais

Funções que são parcialmente aplicadas, ou seja, recebem menos argumentos do que o total esperado, retornando uma nova função que aguarda os argumentos restantes.

Funções Curried

Todas as funções em Haskell são curried por padrão, ou seja, podem ser aplicadas parcialmente. Cada função com múltiplos parâmetros pode ser tratada como uma sequência de funções que aceitam um único parâmetro e retornam uma função.

Funções Lambda (Funções Anônimas)

São funções sem nome definidas de maneira inline, úteis para definir pequenos comportamentos sem a necessidade de declarar uma função separada.

Funções Recursivas

Funções que se chamam a si mesmas, diretamente ou indiretamente, para resolver problemas repetitivos ou de natureza iterativa.

Funções Compostas

Funções construídas a partir da composição de outras funções usando o operador (.). O resultado de uma função serve como entrada para outra.

Funções Parcialmente Definidas

Funções que não são definidas para todos os valores possíveis de entrada.

Funções Monádicas

Funções que trabalham com tipos monádicos, como IO, Maybe, e Either. Elas lidam com efeitos colaterais, falhas ou computações alternativas de forma segura e pura.

Funções Parâmetro Polimórfico

Funções que operam sobre qualquer tipo de dado usando variáveis de tipo genéricas.

Linguagens relacionadas: // caio

Uma de suas principais influências foi a linguagem Miranda, uma linguagem funcional que introduziu muitas das ideias que vemos hoje em Haskell. Outra grande influência foi o Standard ML (SML), que ajudou a moldar a inferência de tipos e o sistema de tipos polimórficos.

Além de ser influenciada por outras linguagens, Haskell também desempenhou um papel crucial no desenvolvimento de linguagens modernas. C#, Python, e Scala, por exemplo, adotaram conceitos funcionais que se originaram ou foram aprimorados em Haskell, como o uso de funções de ordem superior e tipagem forte. Esses conceitos tornaram essas linguagens mais expressivas e eficientes, mostrando a relevância de Haskell no cenário atual de programação.

Desenvolvimento (Compilador / IDE / Código) // caio e vinishow

Quando falamos de desenvolvimento em Haskell, o principal compilador utilizado é o GHC, ou Glasgow Haskell Compiler, que é conhecido por sua eficiência e robustez. O GHC é amplamente utilizado tanto por desenvolvedores quanto pela comunidade acadêmica, e suporta a maior parte das extensões e padrões da linguagem.

Para facilitar o desenvolvimento, existem diversas ferramentas de edição de código. Uma das mais populares é o Visual Studio Code (VSCode), que, com a extensão Haskell Language Server (HLS), oferece uma integração completa com o GHC. Essa extensão proporciona funcionalidades como realce de sintaxe, autocompletar, verificação de erros em tempo real, e suporte a refatoração de código, tornando o desenvolvimento mais eficiente e agradável.

Além do VSCode, outros editores como Sublime Text, Atom, e Emacs também oferecem boas integrações com o Haskell, permitindo que os desenvolvedores escolham a ferramenta que melhor se adapta às suas necessidades

REFERÊNCIAS:

HASKELL. **Haskell: an advanced, purely functional programming language**. Disponível em: <https://www.haskell.org>. Acesso em: 15 set. 2024.

UPENN. **Lecture 01: Introduction to Haskell. University of Pennsylvania, 2013**. Disponível em: <https://www.seas.upenn.edu/~cis1940/spring13/lectures/01-intro.html>. Acesso em: 15 set. 2024.

FONTELA, Tailor. **Capítulos de Haskell. 2022**. Disponível em: <https://haskell.tailorfontela.com.br/chapters>. Acesso em: 15 set. 2024.

ANDRIANA, Maria. **Paradigmas de programação funcional. Universidade Federal de Uberlândia**. Disponível em: <https://www.facom.ufu.br/~madriana/PF_BCC.html>. Acesso em: 15 set. 2024.

IEEE. **Análise da linguagem Haskell. IEEE Xplore, 2018**. Disponível em: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8394394>. Acesso em: 15 set. 2024.

WIKIPEDIA. **Haskell (linguagem de programação)**. Wikipédia, a enciclopédia livre. Disponível em: <https://pt.wikipedia.org/wiki/Haskell_(linguagem_de_programa%C3%A7%C3%A3o)>. Acesso em: 15 set. 2024.

SOUZA, Vitor. **Seminário de Haskell - LP. Universidade Federal do Espírito Santo, 2020**. Disponível em: <http://www.inf.ufes.br/~vitorsouza/archive/2020/wp-content/uploads/teaching-lp-20142-seminario-haskell.pdf>. Acesso em: 15 set. 2024.

UFABC. **Introdução à linguagem Haskell. Universidade Federal do ABC**. Disponível em: <https://haskell.pesquisa.ufabc.edu.br/haskell/03.haskell.basico.1/#:~:text=Haskell%20tem%20como%20características%3A,menores%20que%20em%20outras%20linguagens>. Acesso em: 15 set. 2024.

WIKIPEDIA. **Haskell Programming Language**. Disponível em: <http://en.wikipedia.org/wiki/Haskell_programming_language>. Acesso em: 15 set. 2024.

HUDAK, Paul; PETERSON, John; FASEL, Joseph. **A Gentle Introduction to Haskell 98**. Disponível em: <http://www.haskell.org/tutorial/>. Acesso em: 15 set. 2024.

HUGHES, John. **Why Functional Programming Matters**. Disponível em: <http://www.cse.chalmers.se/~rjmh/Papers/whyfp.html>. Acesso em: 15 set. 2024.

SANTOS, Igor. **Twiki, Haskell**. Disponível em: <http://twiki.im.ufba.br/bin/view/MAT052/HaskellIgor>. Acesso em: 15 set. 2024.

HASKELL-PRIME. **Haskell-Prime.** Disponível em: <http://hackage.haskell.org/trac/haskell-prime>. Acesso em: 15 set. 2024.

DUTTON, C. **Introduction to Haskell**. Disponível em: <http://www.iceteks.com/articles.php/haskell>. Acesso em: 15 set. 2024.

OUTROS:

1. **Slides:**
    a. https://www.canva.com/design/DAGRNFXvaHo/dPiGZc2hQe3R0ARSWBfQuw/edit?utm_content=DAGRNFXvaHo&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton
2. **Icon:**
    a. https://www.flaticon.com/br/icone-gratis/haskell_5968259