.

Gustavo Lopes, Homenique Vieira, Lucas Santiago, Rafael Amauri,
Thiago Henriques

Um estudo estatístico sobre ataques cardíacos e seu prognóstico

Belo Horizonte

2021

Resumo

Este é o manual de uso da classe abntex2. Trata-se de um conjunto de customizações da classe memoir para elaboração de documentos técnicos e científicos condizentes com as normas da Associação Brasileira de Normas Técnicas, especialmente a ABNT NBR 6022:2018, ABNT NBR 10719:2015, ABNT NBR 14724:2011 e a ABNT NBR 6024:2012, além de outras referentes a "Informação e documentação"

Palavras-chave: Ataques cardiácos.

Sumário

Introdução

Em 1930, Alonzo Church, matemático estadunidense apresentou o Cálculo Lambda, como parte da investigação dos fundamentos da matemática. O Cálculo Lambda é um sistema que estuda funções recursivas computáveis e foi utilizada como base para as teorias e fundamentos matemáticos por trás do paradigma da Programação Funcional. Ele também pode ser considerado a primeira linguagem programação funcional, todavia, não foi projetada para ser executada em computadores, sendo apenas um modelo que descreve relações entre funções simples, permitindo criar funções mais complexas.

Com o passar dos anos, varias linguagens funcionais foram criadas, sendo alguns exemplos a linguagem LISP em 1955 e a ML no final da década de 70. Porém, não havia um padrão para as linguagens desse paradigma, e quando chegou a segunda metade da década de 80, havia uma necessidade de criar uma única linguagem, que englobasse as melhores práticas de projeto, além de implementar as técnicas funcionais que estavam em alta na época.

Nesse contexto Haskell foi criado em 1987, por Peyton Jones e Paul Hudak. Sendo assim a The Yale Meeting foi a primeira reunião presencial, no qual foi decidido os principais objetivos que a linguagem proporcionaria, como também a escolha do nome. Segue as metas estabelecidas na reunião:

- Ser viável para o ensino, pesquisa e aplicações, incluindo sistema de larga escala;
- Ser completamente descritiva via publicação no tocante à sua sintaxe e sua semântica;
- Não ser proprietária, tal que qualquer um pudesse implementá-la e distribuí-la;
- Basear-se em ideias que envolvessem o senso comum;
- Reduzir a diversidade desnecessária de outras linguagens funcionais.

A implementação de Haskell começou do zero, desenvolvendo funções únicas e tendo inspiração na linguagem Miranda que estava desempenhando um papel importante na época. Em geral, essa linguagem passou por algumas versões que ajudou muito no desempenho e na adição de novas funções.

1 Histórico sobre a linguagem, com sua cronologia

Depois desse evento, outras reuniões se sucederam e sendo assim no dia 01/04/1990, foi publicado primeiro relatório da versão 1.0 do Haskell. Durante os proximos 15 anos, Haskell teve o lançamento de diferentes versões, tranzendo outras funcionalidades para linguagem, entre elas se encontra Haskell'98 e Haskell 2010 que é sua versão mais recente.

O relatório da versão do Haskell'98 foi lançado em fevereiro de 1999. Ela surgiu no intuito de estabeler uma versão mais estável, para ser possível realizar a documentação mais profunda da lingua em livros, já que o Haskell estava evoluindo muito rápido e ganhando notoriedade.

Em 2006 a equipe começou a planejar revisões anuais para adicionar o progresso do desenvolvimento da linguagem. Sendo assim a primeira revisão, publicada em julho de 2010, foi nomeada Haskell 2010 e nela foi incrementada uma serie recursos que antes não estavam disponiveis. Segue abaixo uma serie dos recursos que foram disponibilizados:

- Do and If Then Else
- Hierarchical Modules
- Empty Data Declarations
- Fixity Resolution
- Foreign Function Interface
- Line Comment Syntax
- Pattern Guards
- Relaxed Dependency Analysis
- Language Pragma
- Remove n+k patterns

Atualmente existem 3930 programadores com contas registradas que utilizam Haskell, sendo que 2566 dessas contas são públicas e 1364 são privadas. Por causa do número de programadores, existem muitas oportunidades nessa área, fazendo os empregos terem um sálario médio de $80.000 à $160.000.

Por mais que Haskell não seja tão utilizado como antes, muitas aplicações foram feitas com a linguagem. Algumas aplicações são:

- A biblioteca open-source Semantic, implementada pelo GitHub, foi puramente escrito em Haskell;

- O Facebook implementa programas anti-spam que são escritos em Haskell e são de código aberto;
- O Snap e Yesod, ambos frameworks para aplicações na web, foram feitos para suportar Haskell;
- O Linspire, sistema operacional comercial, tem Haskell como linguagem escolhida para o desenvolvimento das aplicações do sistema;
- Xmonad, totalmente escrito em Haskell, é um gerenciador de janela para o sistema X Window System.

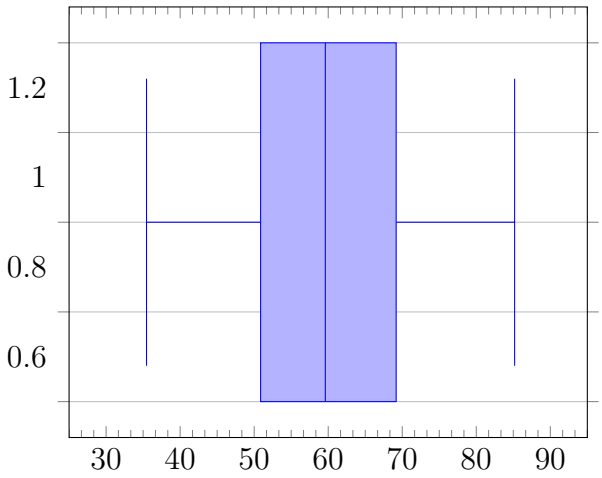

Figura 1 – Box Plot

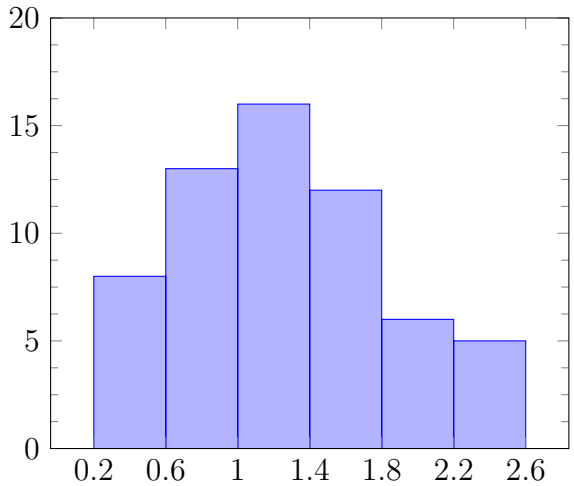

Figura 2 – Histograma

2 Paradigma a que pertence

O paradigma oferece e determina a visão que o programador possui sobre a estruturação e a execução do programa. Um exemplo bem famoso de paradigma é o POO (Programação orientada a objetos), conhecido por ser o modelo para linguagens como C++ e Java.

A Programação funcional é um paradigma que descreve uma expressão matemática a ser avaliada, mapeando dos valores de entradas nos valores de retorno, por meio de funções. Em outras palavras: a programação funcional é baseado em cima de funções que rodam no topo de outras funções. Programação funcional está englobada junto ao grupo da Programação descritiva.

Eis mais algumas características deste paradigma:

2.1 Dados imutáveis

É possível declarar valores a variáveis, mas não pode mudar o valor delas durante a compilação. Isso acontece porque, diferente de linguagens imperativas(Como C), onde você atribui um valor e pode mudá-lo em execuções, variáveis em Haskell(por exemplo) possuem tipagem forte, logo não sofrem efeitos colaterais(*side effects*).

2.2 Funções puras

Também presentes em linguagens como JavaScript e Python, Funções puras são aquelas que recebem um parâmetro *input* e sempre vão retornar o mesmo *input* sem causar efeitos colatreis ao programa.

Esse tipo de função é positiva pelos seguintes aspectos:

- facilita execução de códigos em paralelo, pois não impactam outras funcionalidades que estão em atuação;
- Maior facilidade em criar cache, já que os mesmos parâmetros são sempre esperados.

2.3 Cálculo Lambda

Como mencionado anteriormente, o Cálculo Lambda está presente na programação funcional, e em Haskell não é diferente. Nessa lingua, é possível utilizar as chamadas *Expressões lambda* que são funções anônimas(sem nome), formadas por uma sequência de padrões:

- Argumentos da função
- Corpo

Função anônima para calcular o dobro de x

$$x \rightarrow x + x$$

2.4 Análise crítica

As linguas que seguem o paradigma funcional se caracterizam em criar funcionalidade dentro de estruturas de fácil compreensão e definição. Isso ajuda a manter a confiabilidade, e leitura do código, principalmente quando integrado à forte tipagem. Além disso, táis códigos se caracterizam pelo alto nível de abstração, principalmente ao utilizar as funções, já que isso permite supressão de detalhes da programação, e garante uma menor probabilidade de erros.

Em contraste, tais ideias podem se tornar complicadas, quando utilizadas dentro de contextos onde é necessário de muitas variáveis(Exemplo: Banco de Dados). Além disso, por conta que as funções possuem a característica de serem recursivas por padrão, elas tendem a ser mais demoradas para computar, quando comparado as linguagens imperativas. Ainda nesse ponto, funções recursivas podem gerar maior uso de memória.

3 Características mais marcantes da linguagem

A linguagem Haskell pode nos apresentar características peculiares que a destaca, entre elas o Lazy Evaluation ou também conhecido como *laziness*. Como exemplo, se ecrevermos um pequeno filtro ou parser em Haskell, não é necessário preocupar com os detalhes de ler input linha-por-linha ou bloco-por-bloco porque é possível simplesmente utilizar *getContents* e então encadear uma série de funções. Uma vez que se tenha o *laziness* na entrada, se tem o *laziness* na saída porque imprimir algo não requer que tudo seja processado de uma só vez. Sendo assim, o programa fica com um design muito simples pois não é preciso lidar com buffers ou iterar sobre as linhas e ainda roda em um espaço constante.

Além da técnica descrita acima, Haskell também apresenta benefícios em termos de expressividade, alcançado por *matching the patterns* e o fato de que funções podem expressar coisas de forma sucinta, mas legível, tornando mais fácil representar um problema e então, pensar na lógica, que é claramente identificável. Com Haskell, manipula-se funções tão fácil quanto a linguagem Pear manipula strings.

A linguagem Haskell, assim como todas as outras linguagens, tem suas sintaxes e estruturação próprias. O centro da organização de um programa, seja ele feito em Haskell ou não, está definido nas funções que são criadas, porém, existem outros fatores importantes a serem observados. Nas primeiras linhas do programa são descritos os módulos que serão utilizados pelo mesmo. Para utilizar um módulo dentro de um programa basta colocar a seguinte declaração:

Já a criação de módulos é diferente, o cabeçalho deve ser escrito da seguinte forma:

Podemos importar outros módulos dentro de um módulo que estamos criando. O corpo do módulo é onde estão as funções criadas para serem utilizadas. Mesmo podendo haver vários módulos dentro de um mesmo arquivo, mesmo não sendo recomendado. Dentro da linguagem Haskell temos vários tipos de dados primitivos, operadores lógicos e alguns caracteres especiais. Quanto aos operadores lógicos, os mais comuns são: &&, || e not. Para os tipos de dados primitivos, temos: Bool, Int, Char, String, Float e Double - tipos comuns da maioria das linguagens de programação. Além dos operadores lógicos, Haskell proporciona os operadores aritméticos:

+	Soma
-	Subtração
*	Multiplicação
∧	Potência
div	Divisão inteira
mod	Resto de divisão
abs	Valor absoluto de um inteiro
negate	Troca o sinal do valor

Tabela 1 – Operadores aritméticos

Para finalizar os tipos primitivos em Haskell, os caracteres especiais possuem as seguintes representações:

'\t'	Marca de tabulação
'\n'	Nova linhas
'\\'	Barra invertida
'\"	Aspas simples
'\"'	Aspas duplas

Tabela 2 – Caracteres especiais

Em Haskell, é possível criar tipos abstratos de dados, como tuplas e listas. Em uma tupla, podemos inserir um número determinado de dados com tipos predefinidos, já em uma lista os dados são todos do mesmo tipo (possibilitando a criação de uma lista que contenha tuplas, desde que os tipos de dados dentro das tuplas sejam iguais entre si). Apesar de serem tipos de dados abstratos, tuplas e listas não são as únicas dentro da linguagem, Haskell nos permite criar classes dentro dele. As classes em Haskell, são semelhantes às classes na orientação de objetos, com algumas implementações diferentes. Para começar uma classe, temos uma expressão como mostrada abaixo:

$$class < nomeDaClasse >< tipoDeEntrada > where < assinaturaDaClasse >$$

Definindo uma classe

O nome da classe é um identificador e o tipo de entrada é para representar o tipo de dado, ou uma variável que atuaria como variável de tipo - possibilitando que

a classe aceite qualquer tipo de dado na entrada. Sendo assim, o tipo de entrada pode ser tanto um inteiro, um char ou até uma string, dependendo da intenção do usuário.

Por fim, não podemos deixar de falar no tratamento de exceção em Haskell, que é feito, não surpreendentemente, por meio de expressões. O comando *catch* é uma função cujo primeiro parâmetro uma expressão a executar e o segundo parâmetro é o que deve ser feito caso ocorra uma exceção. Além do comando *catch*, temos as palavras chaves *raise* e *throw*, que são utilizadas para definirmos exceções definidas e especificadas pelo programador.

- Como dito anteriormente, a linguagem só faz utilização de funções e funções dentro de funções. Por isso Haskell é descrito como puramente funcional;
- Haskell possui uma sintaxe simples, elegante e concisa. Como resultado, programas em Haskell possuem poucas linhas;
- Além disso, a linguagem usa avaliação preguiçosa (*Lazy evaluation*), que é uma técnica para atrasar a computação até um ponto em que o resultado da computação é considerado necessário;
- Tipagem estática: Verificação dos tipos usados em dados e variáveis para garantir que sempre está sendo usado um tipo que é esperado em todas as situações;
- Função de ordem superior: Função que tem como argumento uma outra função, ou que produz uma função como resultado;
- Antes da execução de um programa, os compiladores e interpretadores realizam uma checagem forte de tipos de dados, verificação monomórfica e verificação polimórfica.

4 Linguagem similares ou confrontantes

- Prolog

  * Haskell eh uma linguagem funcional, que requer uma descrição matemática e lógica para resolver um problema. Prolog é uma linguagem declarativa, que declara um conjunto de regras sobre qual a saída que deve ser resultada de qual entrada e assim deduz qual a saída deve ser o resultado de qual entrada;

  * Prolog é especialmente fácil para a resolução de problemas lógicos e Haskell é melhor para a resolução de problemas que podem ser computados com algoritmos.

- LISP

  * Ambos Haskell e LISP têm uma sintaxe minimalista comparada a C++, C# e Java;

  * Em Haskell são escritos pequenos pedaços de código que são combinados pela linguagem;

  * LISP é uma linguagem homiconica, enquanto Haskell usa computação monadica.

- Scheme

  * Scheme, assim como Haskell, também é uma linguagem funcional;

  * A sintaxe de Scheme é extremamente simples e a linguagem se apoia em ter um núcleo com poucas funcionalidades que pode ser facilmente expandida com diversas ferramentas de extensão;

  * Scheme é uma linguagem interpretada, que se difere de Haskell, já que Haskell tem a opção de ser tanto uma linguagem compilada ou interpretada;

  * Scheme é uma linguagem que utiliza tipagem dinâmica, o que entra em constraste com Haskell, que utiliza tipagem estática.

- ML

  * ML é uma linguagem que usa declaração implícita, o que garante *type-safety*. Haskell não possui declaração implícita;

  * Ambas possuem coletor de lixo;

  * ML é uma linguagem que utiliza chamada por valor, enquanto Haskell possui tanto chamada por valor quanto por referência.

- Miranda

  * Ambas são linguagens puramente funcional. Linguagens puramente funcionais executam todo o código em funções;

  * Miranda utiliza tipagem estática, assim como Haskell, C e outras linguagem da época;

* Miranda tem um algoritmo de parsing implementado que faz uso de indentação ao invés de chaves - - para indentação. Essa característica viria a ser popularizada na linguagem Python muitos anos no futuro. Haskell, assim como muitas outras linguagens da época, utiliza caracteres para indentação;

* Por ser uma linguagem puramente funcional, é impossível existir *side-effects* em ambas as línguas.

- Elixir

  * Elixir é compilada para bytecode que será rodado na Erlang Virtual Machine. Haskell é compilada para código de máquina;

  * Elixir consegue chamar funções de Erlang e vice-versa, isso sem qualquer impacto em *runtime*. Haskell não é capaz de chamar funções de outras linguagens, pois ela é compilada;

  * Elixir é uma linguagem *concurrent*, o que significa que o código é rodado ao mesmo tempo ao invés de sequencialmente. Haskell, pelo contrário, é uma linguagem sequencial, podendo ser paralelizada manualmente;

  * Assim como Haskell, Elixir é uma linguagem funcional.

5 Exemplo(s) de programa(s)

5.1 Hello World em Haskell

5.2 Quicksort

5.2.1 Exemplo de Quicksort em C

5.2.2 Exemplo de Quicksort em Haskell usando compreensão de lista

5.2.3 Exemplo de Quicksort em Haskell usando a função filter

5.3 Explicação dos algoritmos

Os algoritmos em Haskell são extremamente compactos em relação à C. Sua sintaxe não tem foco em programar diretamente a memória, como acontece em C. Começando pelo *Hello World* simples ou evoluindo para um quicksort, a lingua mostra-se bem compacta e direta na resolução do problema.

No primeiro exemplo, foi usado *list comprehension* para resolver o problema. Na primeira linha, foi declarado uma lista vazia para ser retornada caso o quicksort receba uma lista sem valores. A segunda linha cria uma função recursiva, caso tenha valores na lista para serem ordenados.

5.3.1 Exemplo de Quicksort em Haskell usando a função filter

Esse é um modelo traduzido diretamente do exemplo do quicksort em C para Haskell. Modelo resumido usando recursos da linguagem abaixo:

5.3.2 Exemplo de Quicksort em Haskell usando a função filter

Mesmo resumindo o código, a forma imperativa para o quicksort não parece ser uma boa opção, usar o quicksort de forma recursiva dentro do Haskell torna o código muito mais simples e intuitivo.

Conclusão

Após trabalhar nessa linguagem por quase um mês, ficou claro para toda a equipe que Haskell é uma linguagem diferenciada. Possui uma história que foi essencial para sua formação, sem esquecer, é claro, que foi uma lingua desenvolvida de forma comunitária. Há grandes obstáculos em ingressar nessa lingua, principalmente por documentação excassa e a documentação oficial em maioria ser paga ou extremamente complexa para iniciantes.

Por muito tempo, Haskell não foi uma lingua unificada, cada pessoa que entrava no projeto criava uma versão diferente sem que um líder principal coordenasse como ela estava evoluindo. Falta de uma unidade atrasou um pouco a lingua ser adotada pela comunidade. Por conta da falta de uma documentação única, a dificuldade de aprendizado foi outro grande ponto negativo que impactou diretamente na falta de profissionais que a utilização, ficando apenas fechada em um ambiente científico como universidades.

Entretanto depois de tudo que vimos, a linguagem se apresenta de forma bem mais positiva do que todas essas ideias citadas acima. Ela apresenta vários recursivos interessantes, como cálculos lambda, recursões simples e amarrações de funções em variáveis de forma simples. Há vários tutoriais distribuidos pela internet. Mesmo que poucas pessoas programem nessa lingua, possui uma comunidade bem forte que a mantém.

Por fim, o grupo entendeu que Haskell desempenhou seu papel na história da computação. Além disso, várias empresas ainda o adotam por entenderem a importância de seu uso, feito para cálculos científicos e precisos. Ainda dentro do contexto de programação funcional e uso de cálculos lambda, Haskell ainda é uma, se não a melhor, lingua para ser utilizado.

Referências

ARAUJO, L. C. *Como customizar o abnTeX2*. 2015. Wiki do abnTeX2. Disponível em: <https://github.com/abntex/abntex2/wiki/ComoCustomizar>. Acesso em: 27 abr 2015. Nenhuma citação no texto.

BERTO, H. B. *O que são funções puras*. Disponível em: <https://tableless.com.br/o-que-sao-funcoes-puras/>. Acesso em: 25 march 2021. Nenhuma citação no texto.

BOSCAGLIA, B. S.; SATLER, P. S.; MATOS, R. P. de. *Haskell*. Disponível em: <http://www.inf.ufes.br/~vitorsouza/archive/2020/wp-content/uploads/teaching-lp-20132-seminario-haskell.pdf>. Acesso em: 25 march 2021. Nenhuma citação no texto.

DESCONHECIDO. *Haskell Jobs - March 2021 | Functional Works*. Disponível em: <https://functional.works-hub.com/haskell-jobs>. Acesso em: 25 march 2021. Nenhuma citação no texto.

DESCONHECIDO. *Haskell (linguagem de programação)*. Disponível em: <https://pt.wikipedia.org/wiki/Haskell_(linguagem_de_programa%C3%A7%C3%A3o)#Paradigma>. Acesso em: 25 march 2021. Nenhuma citação no texto.

FRAYZE, M. *Funções puras*. Disponível em: <https://marciofrayze.medium.com/fun%C3%A7%C3%B5es-puras-1ed312f75763>. Acesso em: 25 march 2021. Nenhuma citação no texto.

HUDAK, P. et al. A history of haskell: Being lazy with class. *HOPL III: Proceedings of the third ACM SIGPLAN conference on History of programming languages*, v. 1, n. 1, p. 12–1–12–55, 2007. Nenhuma citação no texto.

JONES, S. P. *Haskell 98 report*. Disponível em: <https://www.haskell.org/definition/haskell98-report.pdf>. Acesso em: 25 march 2021. Nenhuma citação no texto.

MALAQUIAS, J. R. *Programação Funcional em Haskell*. Disponível em: <http://www.decom.ufop.br/romildo/2018-1/bcc222/slides/progfunc.pdf>. Acesso em: 25 march 2021. Nenhuma citação no texto.

MARLOW, S. *Haskell 2010 report*. Disponível em: <https://www.haskell.org/definition/haskell2010.pdf>. Acesso em: 25 march 2021. Nenhuma citação no texto.

QASTACK. *Haskell minimalist quicksort*. Disponível em: <https://qastack.com.br/programming/7717691/why-is-the-minimalist-example-haskell-quicksort-not-a-true-quicksort>. Acesso em: 25 march 2021. Nenhuma citação no texto.

SINGH, C. *Quicksort program in c*. Disponível em: <https://beginnersbook.com/2015/02/quicksort-program-in-c/>. Acesso em: 17 march 2021. Nenhuma citação no texto.

SNOYMAN, M. *The meeting place for professional Haskell programmers*. Disponível em: <https://www.haskellers.com/>. Acesso em: 25 march 2021. Nenhuma citação no texto.

Apêndices

APÊNDICE A – Gustavo Lopes

O que eu achei de Haskell? Como um usuário de longa data de linguagens orientadas a objeto(OOP), como Java, C++ e mais recentemente Dart, minha experiência inicial com Haskell foi... um pouco estranha. Fiquei muito curioso e até encucado em ver Haskell executar códigos, que em outras linguagens precisariam ocupar 50 linhas, em apenas 2 ou 3 linhas, como foi o Quicksort.

Além disso, lendo sua história, achei extremamente fascinante toda a concepção do Haskell, mas fiquei triste vendo que é muito difícil encontrar muitas pessoas falando sobre essa linguagem, que nasceu do esforço coletivo de vários programadores, mas que hoje não recebe tanto apoio como outras linguagens.

Por fim, como curiosidade, eu estava procurando sobre como fazer interfaces em C++ quando descobri que a *WxWidgets*, uma biblioteca para criar interface cross-platform, também possui sua versão para Haskell, chamada *WxHaskell*.

APÊNDICE B – Thiago Henriques

Bom, como posso expressar o que eu senti aprendendo sobre essa nova linguagem? Eu gosto muito das linguagens de alto nível, principalmente Python, então no começo devo admitir que senti um certo desafio. Porém, sinto que Haskell conseguiu cumprir o seu propósito e possibilitou a criação de novos.

Em geral fiquei fascinado como a linguagem foi implementada e a tomada de decisão da equipe com o passar do tempo. A implementação possibilitou a adição de diversas funcionalidades únicas da linguagem e facilitou o desenvolvimento mais fluido da mesma. Além disso, achei interresante como em alguns códigos de Haskell eram necessário poucas linhas para se executar uma ação, assim me lembrando um pouco de Python.

Eu observei que Haskell possui diversas implementações no mercado e muitos projetos de código aberto que utilizam da linguagem. Um dos projetos que eu achei muito interessante foi o Detexify, feito totalmente em Haskell, que tem como objetivo simplificar a busca de símbolos especiais para aqueles que trabalham em LaTeX. O usuário só precisa desenhar o símbolo em uma caixa e o comando será exibido em LaTeX.

APÊNDICE C – Lucas Santiago

Na minha opinião, Haskell se apresenta de uma forma muito diferente da maioria da outras línguas. Parafraseando Fábio Akita: cada lingua de programação não é mais do que uma ferramenta para resolver um problema.

Os problemas que Haskell se propôs a resolver são cálculos matemáticos funcionais. O paradigma funcional dessa linguagem torna ela bem resistente a efeitos colaterais. Além de ser uma lingua *cross-platform*, extremamente importante para todas aquelas pessoas que precisem de uma lingua que funcione em qualquer sistema operacional.

Como um programador de C/C++ e Python, vejo que Haskell está em um nível intermediário entre essas línguas. É interessante ver uma lingua de alto nível com tipagem estática, torna o código bem estável - chances baixas de resultar em erros -, tipagem é fundamental quando se quer garantir precisão de valores nos cálculos.

Concluindo, não foi tão intuitivo aprender Haskell, ele possui várias propriedades que não existem em nenhuma das línguas que conhecia. Entretando, isso foi extremamente positivo para mim, aprendi várias coisas novas. Abaixo há mais um vídeo do Fabio Akita, dessa vez citando que nem sempre é fácil aprender algo novo. Em programação, quando você aprende da forma correta, várias novas possibilidades de resolução de problemas começam a aparecer.

APÊNDICE D – Pedro Souza

Haskell definitivamente é uma linguagem um tanto quanto diferente das outras que são consideradas mais comuns Java, C++, C#, Javascript, Python, etc. Apesar de não ter uma sintaxe muito complexa, sua implementação pode ser um pouco estranhada para quem está começando a conhecer a linguagem.

O fato de ser baseada em expressões, Haskell se torna um tanto quanto amedrontador para programadores novos, com pouca experiência e que estão mais acostumados com as linguagens mais usuais. Apesar de ter estudado e descoberto um pouco mais sobre a linguagem, admito que continuo relutante quanto a usá-la. Talvez por ser um programador ainda em formação e com pouca experiência, não consigo ainda pensar em aplicações que necessariamente teriam de ser feitas com Haskell.

Como muitas outras linguagens por aí, Haskell continua muito oculta, sem muito reconhecimento e fama, um grande exemplo disso é que se pesquisarmos "Haskell" no google, encontraremos algum resultado sobre a linguagem apenas na segunda página e no último item encontrado. E aqui entre nós, quantas vezes vamos até a segunda página do google em alguma pesquisa?