**Expressões regular em linguagem funcional: Módulo de busca usando expressões regulares implementado em Haskell**

GOMES,Bruno

LIMA, José Airton G. de

**1 INTRODUÇÃO**

1.1 Objetivos

Objetivo geral: Implementar uma biblioteca de processamento de expressões regulares para a linguagem Haskell.

Objetivo especifico:

- Entender o paradigma funcional e os componentes da

linguagem Haskell

- Estudar a teoria de autômatos, variantes das expressões regulares e algoritmos de processadores de expressões regulares

- Implementar biblioteca de busca de Strings em Haskell

1.2 Justificativa

Embora o objetivo deste trabalho seja a construção de uma biblioteca para processamento de expressões regulares, a motivação para isso ´e de realizar um estudo sobre o paradigma funcional. Dentre as inúmeras linguagens de programação já inventadas, as que realmente se destacam e são utilizadas em grandes aplicações são

poucas. Dentre elas, podemos destacar as linguages Java, Python, C, C++, C#,

Go, Shell Scripting e JavaScript. Embora a diferenca entre essas linguagems serem

gritantes para pessoas com um mínimo conhecimento sobre elas, todas podem ser

classificadas como linguagens imperativas. Por mais diferentes que essas linguagens

sejam, elas compartilham um paradigma para a resolução do problema computacional em questão. Linguagens imperativas focam em dizer como o problema deverá

ser resolvido.

Embora ser muito utilizada, a programação imperativa não é o único paradigma

computacional existente. Um dos paradigmas que contrasta diretamente e o paradigma de programação funcional. Em poucas palavras, a grande diferença entre esses paradigmas ´e que enquanto a programação imperativa foca no como resolver

o problema, a programacao funcional foca no que o programa faz. O ponto de contraste são as palavras como é que, embora seja uma definição simplista, consegue

capturar a essencia dresses paradigmas.

Embora o paradigma funcional seja muito menos disseminado, ele eh de extrema importância. Grandes descobertas nesse campo de estudo transcende o paradigma funcional e infeta as linguagens procedurais. Alguns exemplos sao as list comprehensions da linguagem Python e as lambda expressions e streams introduzidos

na versão 8 da linguagem Java, um dos gigantes do mercado. Sendo assim, esse

trabalho tem como objetivo introduzir o paradigma funcional, focado na linguagem

Haskell. Serão introduzidos os principais conceitos e ferramentas das linguagens

funcionais com um foco empírico. Esses conceitos serão utilizados para implementar

uma engine de processamento de texto usando expressões regulares.

Em conclusão, o paradigma funcional faz parte do dia a dia da grande maioria dos profissionais da área, embora muitas vezes não saibam disso. Elementos

de sucesso de linguagens funcionais acabam sendo implementados nas grandes linguagens imperativas pois os mesmos se mostraram úteis e atraente. Sendo assim,

desenvolvedores devem conhecer um mínimo sobre o paradigma funcional, nao so

pois ele está presente em grande parte das linguages mas porque alguns problemas

podem ser resolvidos de maneira mais breve.

**2 FUNDAMENTACAO TEORICA**.

Como visto nos objetivos, esse trabalho pode ser quebrado em 3 partes: expressões regulares (regex), programação funcional e algoritmos para processamento de regex.

**2.1.1 Expressoes Regulares**

Uma expressao regular (regex) eh um string de caracteres que define um padrão de busca. Essas strings são construídas a fim de encontrar/extrair informações em texto que seguem uma estrutura pré conhecida com elementos dinâmicos.

Um exemplo, é possível escrever uma regex para encontrar em um texto todas as datas escritas no formato usual de DD/MM/AAAA, onde DD indica o dia, MM o mês e AAAA o ano. Uma regex capaz de encontrar esse padrão é ”[0-9]2 /[0-9]2 /[0-9]4”. Essa regex indica que para um texto ser aceito ele deve contar os dígitos de 0 a 9, repetidos duas vezes (indicado pelo substring [0-9]{2}), seguido por uma barra, outro grupo de dois dígitos, outra barra e finalmente um grupo de 4 dígitos.

A notação de regexes pode variar dependendo da implementação, porém seguindo o modelo Perl de regexes, a notação ”[]” indica um grupo de caracteres que são aceitos para aquela posição de texto, [012] indica o conjunto de números 1,2,3 que deve ser aceito como correto. Em algumas situações, é possível indicar um intervalo de caracteres, como foi feito acima. O grupo ”[0-9]” aceita os números 0,1,2,3,4,6,7,8,9 como corretos para fins de procura, é possível também definir um conjunto de letras. O conjunto ”[a-z]” aceita as letras de ”a” ate ”z”, minusculas como corretas para o padrao. O caráter ”.” é definido como um caractere reservado em diversas implementações de regexes diferentes, ”.” simboliza um grupo onde qualquer caracter eh aceito.

É possível enumerar o número de vezes que um caractere, ou grupo de caracteres, podem aparecer em um regex, isso eh indicado pela notação ””. Ainda usando o exemplo de datas, o substring ”2” indica que o caracter que antecede o enumerador será repetido duas vezes. Note que para fins de processamento, um grupo é considerado como um único caracter pois somente um único caracter irá dar match por vez, logo a regex ”[0-9]2” irá dar match em qualquer dois dígitos adjacentes. Enumeradores podem indicar um intervalo com comprimento unidade ou um intervalo arbitrário, como o enumerador 2,5, que define um enumerador que aceita de 2 a 5 caracteres. É possível omitir o número a direita ou esquerda da vírgula para definir um intervalo flexível, o intervalo 0, aceitá 0 ou qualquer número de ocorrências do grupo que o precede. Alguns caracteres são reservados para indicar intervalos especiais em regexes. O caracter ”?” eh equivalente ao intervalo ”0,1”, ou seja, o grupo anterior pode ocorrer 0 ou 1 vezes, usado para indicar quando uma condição eh opcional. Outro caracter especial eh ”?” que indica o intervalo ”1,”, ou seja um ou mais caracteres. Finalmente, o caracter ”\*” eh equivalente ao intervalo ”0,” que aceita qualquer número de ocorrências do grupo a sua esquerda.

Essa eh uma breve introdução sobre expressões regulares, existem diversos caracteres especiais que podem ser usados para indicar posições no texto. Para uma referência completa sobre as funcionalidades avançadas de expressões regulares consulte (python-re), (jeffrey). do conteúdo deve possuir uma ordem sequencial.

**2.1.2 Programacao Funcional**

Os conceitos de programação funcional surgiram junto do cálculo lambda, inventado por Alonzo Church. Embora existam várias linguagens funcionais (ML, Lisp, Rackett, Haskell), todas elas compartilham essa origem. Tal como em programação imperativa que diferencia linguagens entre procedural e orientada a objetos, existem diferenças entre as linguagens funcionais, mas primeiramente, o que é programação funcional. Segundo (Bird), ”programação funcional é: um método para construção de programas que enfatiza funções e suas aplicações ao invés de comandos e suas execuções; programação funcional faz uso de notação matemática simples que permite que problemas sejam descritos de maneira clara e concisa. [...]”.

Esse trabalho foca na linguagem Haskell como fonte de exemplos para o paradigma funcional. Haskell, como toda linguagem, possui suas próprias particularidades, porém grande partes dos conceitos apresentados será geral para linguagens funcionais.

Haskell faz parte do conjunto de linguagens conhecidas como ”linguagens funcionais puras”. Uma linguagem pura permite a definição de símbolos uma única vez, exemplo, ao definir ”let a = 4” em um programa, tentar mudar o valor da variável ”a” resulta em um erro. O valor de um símbolo não pode mudar durante a execução do programa, de certa forma isso é equivalente as keywords ”final” do java e ”const” da linguagem C. Embora não poder atualizar o valor de uma variável soe como uma grande desvantagem, isso permite o que pode ser chamado de ”funções sem efeito-colateral”. Isso significa que uma função jamais irá alterar o estado do programa fora dela, toda vez que uma função for chamada com os mesmos atributos, ela irá retornar o mesmo valor, isso não é o caso em linguagens orientadas a objeto, por exemplo, onde métodos alteram o estado do objeto (setters). O conceito de funções sem efeito colateral é uma vantagem pois isso facilita o entendimento do programador sobre a sequência de eventos do programa, não existe preocupação de que uma função altere alguma variável global ou o estado de um objeto. Lipovaca expressa o conceito de linguagem funcional pura de maneira instrutiva, ele diz que: ”Embora isso parece limitante quando você está vindo do mundo imperativo, nós vimos que isso eh algo realmente legal. Em uma linguagem imperativa voce nao tem garantia de que uma simples função que deveria somente processar alguns números não irá queimar sua casa, sequestrar seu cachorro e riscar seu carro com uma batata enquanto processa aqueles números.”.

Como nao podemos alterar o estado de uma variável em um programa funcional, isso nos força a procurar diferentes maneiras de resolver problemas computacionais. Um exemplo clássico disso é o problema de calcular o fatorial de um número. O fatorial de n (n!) é definido como n! = (1) ∗ (2) ∗ ... ∗ (n − 1) ∗ (n). Em uma linguagem imperativa, esse problema pode ser resolvido com um for, uma variável acumuladora e uma variável contadora; como nao podemos mutar o valor de definições em linguagens funcionais, podemos fazer uso de recursão. O programa abaixo define uma funcao ”factorial” que recebe um valor do tipo int retorna um de valor int. A funcao ”factorial” retorna o valor 1 quando seu argumento é 0 e retorna o valor n \* (n-1)! para qualquer outro valor. Essas curiosa sintaxe faz uso de um construtor muito útil do Haskell chamado de pattern matching. Segundo ??, ”pattern matching consiste de especificar padrões para o qual algum dado deve tomar forma, verificar se o dado acorda com esses padrões”, é possível definir vários padrões para uma única função. Padrões são executados de cima para baixo e por isso padrões mais gerais (dão match em um maior número de situações) devem ser posicionados por último.

factorial :: Int -> Int

factorial 1 = 1

factorial n = n \* factorial (n-1)

Tipos de dados diferentes possuem diferentes possibilidades de pattern matching. Em Haskell, valores contidos dentro de colchetes ”[]” define uma lista, e o padrão (x:xs) define que x é o primeiro elemento da lista e xs o restante da lista. Por exemplo, dado a lista [1,2,3] os valores de (x:xs) em um pattern matching terá o valor de x = 1 e xs = [2,3]. Usando esse construtor, podemos definir a função sum que soma todos os número em uma lista.

sum :: [Int] -> Int

sum [] = 0

sum (x:xs) = x + sum xs

O código acima define uma função sum que recebe uma lista de itens (”[Int]”) e retorna um int, a função sum faz uso de dois pattern matchings. O primeiro padrão ”[]” faz match com uma lista vazia, o valor da função sum para a lista vazia é definido como o valor 0. Caso a lista não seja vazia, o segundo padrão é executado, e para aquele caso a soma de uma lista é definida como o primeiro elemento mais a soma do restante da lista. Novamente, isso é uma função que faz uso de recursão para resolver o problema computacional. A função sum chama a ela mesma até o valor de xs ser a lista vazia, para o qual definimos que isso seria igual a 0. Caso omitirmos o primeiro padrão, a função iria chamar a ela mesma indefinitivamente. Pattern matching eh muito util em funções recursivas pois nos permite definir a condição de saída da recursão.

**2.1.3 Algoritmos de regex**

Nessa seção será abordado um pouco sobre máquinas de estado, como elas podem ser usadas para representar expressões regulares. Talvez abordar a diferença entre máquinas determinísticas e não determinísticas, e como uma máquina não determinística pode ser transformada em determinística. Ainda estou em duvida qual algoritmo será utilizado para a construção da biblioteca, irei pesquisar mais sobre isso.da lógica inerente a qualquer assunto, que uma vez

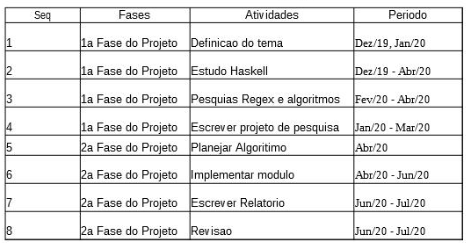
**3 METODOLOGIA**

Utilizando os algoritmos abordados na seção de algoritmos, será criado um módulo escrito funcionalmente em Haskell para realizar a pesquisa de regexes em um texto. Tenho duvida a respeito dessa seção. O que faz parte do escopo de metodologia? Práticas de desenvolvimento aplicadas como testes unitários? Acredito que a maior parte do conteúdo abordado aqui seja estabelecido após a escolha do algoritmo(s) de regex a serem implementados. A definição do dialeto da regex, junto dos símbolos especiais, flags e etc deve ser feita nessa seção ou na fundamentação teórica?

**4 CONSIDERAÇÕES FINAIS**

O objetivo deste trabalho foi expor alguns conceitos por trás do paradigma funcional e demonstrar como esses conceitos são úteis através da construção de um módulo de processamento de expressões regulares. O módulo foi implementado usando a linguagem Haskell, uma linguagem funcional pura. Primeiramente foi abordado as origens da programação funcional, tal como as principais diferenças entre o paradigma funcional e imperativo. Ainda, foram introduzidos alguns conceitos exclusivos das linguagens funcionais, tais como Monads. Foi abordado de maneira abrangente o conceito por trás das expressões regulares (regexes) e alguns casos onde são úteis, juntamente com parte da sua história. Por último foi discutido diferentes métodos para se resolver o problema computacional referente a pesquisa de expressões regulares. Usando os conceitos apresentados foi demonstrado o processo de construção do módulo de regex.

**4 CRONOGRAMA**



**REFERÊNCIAS**

Referencias temporarias sem formatacao

<https://books.google.com.br/books?id=R2RbBAAAQBAJ>

<https://books.google.com.br/books?id=B4RxBAAAQBAJ>

<https://books.google.com.br/books?id=NLngYyWFl_YC>

<https://books.google.com.br/books?id=olp7lAtpRX0C>

<https://books.google.com.br/books?id=e5ahCgAAQBAJ>

<https://docs.python.org/3/library/re.html>

<https://wiki.haskell.org/Introduction>

<https://books.google.com.br/books?id=NeTqCAAAQBAJ>

<https://books.google.com.br/books?id=GX3w_18-JegC>