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

Bruno Gomes

16 de fevereiro de 2020

1 INTRODUCAO

1.1 Objetivos

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

Objectivo especifico: - Entender o paradigma funcional e os componentes da linguagem Haskell - Estudar a teoria de automatas, variantes das expressoes regulares e algoritimos de processadores de expressoes regulares - Implentar biblioteca de busca de Strings em Haskell

1.2 Justificativa

Embora o objetivo deste trabalho seja a construcao de uma biblioteca para processamento de expressões regulares, a motivacao para isso é de realizar um estudo sobre o paradigma funcional. Dentre as inumeras linguagens de programcao ja inventadas, as que realmente se destacam e sao utilizadas em grandes aplicacoes sao 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 minimo conhecimento sobre elas, todas podem ser classificas como linguagems imperativas. Por mais diferentes que essas linguagems sejam, elas compartilham um paradigma para a resolucao do problema computacional em questao. Linguagens imperativas focam em dizer *como* o problema devera ser resolvido.

Embora ser muito utilizada, a programacao imperativa nao é o unico paradigma computacional existente. Um dos paradigmas que contrasta diretamente é o paradigma de programacao funcional. Em poucas palavras, a grande diferenca entre esses paradigmas é que enquanto a programacao imperativa foca no como resolver o problema, a programacao funcional foca no *que* o programa faz. O ponto de constrate sao as palavras como e que, embora seja uma definicao simplista, consegue capturar a essencia desses paradigmas.

Embora o paradigma funcional seja muito menos dissiminado, ele eh de extrema importancia. Grandes descobertas nesse campo de estudo transcende o paradigma

funcional e 'infecta' as linguagens procedurais. Alguns exemplos são as list comprehenssions 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. Serao introduzidos os principais conceitos e ferramentas das linguagems funcionais com um foco empirico. Esses conceitos serao utilizados para implementar uma engine de processamento de texto usando expressoes regulares.

Em conclusão, o paradgima funcional faz parte do dia a dia da grande maioria dos profissionais da area, emboram muitas vezes nao saibam dissos. Elementos de sucesso de linguagens funcionais acabam sendo implementados nas grandes linguagens imperativas pois os mesmos se mostraram uteis e atraente. Sendo assim, desenvolvedores devem conhecer um minimo sobre o paradgima funcional, nao so pois ele esta presente em grande parte das linguages mas porque alguns problemas podem ser resolvidos de maneira mais breve.

2 DESENVOLVIMENTO

2.1 FUNDAMENTACAO TEORICA

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

2.1.1 Expressoes Regulares

Uma expressao regular (regex) eh um string de caracters que define um padrao de busca. Essas strings sao construidas a fim de encontrar/extrair informacoes em texto que seguem uma estrutura pre conhecida com elementos dinamicos.

Um exemplo, eh possivel 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 mes e AAAA o ano. Uma regex capaz de encontrar esse padrao e "[0-9]2
/[0-9]2
/[0-9]4". Essa regex indica que para um texto ser aceito ele deve contar os digitos de 0 a 9, repetidos duas vezes (indicado pelo substring [0-9]2), seguido por uma barra, outro grupo de dois digitos, outra barra e finalemente um groupo de 4 digitos.

A notacao de regexes pode variar dependendo da implementacao, porem seguindo o modelo Perl de regexes, a notacao "[]" indica um grupo de caracters que sao aceitos para aquela posicao de texto, [012] indica o conjunto de numeros 1,2,3 que deve ser aceito como correto. Em algumas situacoes, eh possivel indicar um intervalo de caracters, como foi feito acima. O grupo "[0-9]" aceita os numeros 0,1,2,3,4,6,7,8,9 como corretos para fins de procura, eh possivel tambem definir um conjunto de letras. O cojunto "[a-z]" aceita as letras de "a" ate "z", minusculas como corretas para o padrao. O caracter "." eh definido como um caracter reservado em diversas implementacoes de regexes diferentes, "." simboliza um grupo onde qualquer caracter eh aceito.

Eh possivel enumerar o numero de vezes que um caracter, ou grupo de caracteres, podem aparecer em um regex, isso eh indicado pela notacao "". Ainda usando o exemplo de datas, o substring "2" indica que o caracter que antecede o enumerador sera repetido duas vezes. Note que para fins de processamento, um grupo eh consi-

derado como um unico caracter pois somente um unico caracter ira dar match por vez, logo a regex "[0-9]2" ira dar match em qualquer dois digitos adjacentes. Enumeradores podem indicar um intervalo com comprimento unidade ou um intervalo arbitrario, como o enumerador 2,5, que define um enumerador que aceita de 2 a 5 caracters. Eh possivel omitor o numero a direita ou esquerda da virgula para definir um intervalo flexivel, o intervalo 0, aceita 0 ou qualquer numero de ocorrencias do grupo que o precede. Alguns caracteres sao 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 condicao eh opcional. Outro caracter especial eh "?" que indica o intervalo "1,", ou seja um ou mais caracters. Finalmente, o caracter "*" eh equivalente ao intervalo "0," que aceita qualquer numero de ocorrencias do grupo a sua esquerda.

Essa eh uma breve introducao sobre expressoes regulares, existem diversos caracters especiais que podem ser usados para indicar posicoes no texto. Para uma referencia completa sobre as funcionalidades avancadas de expressoes regulares consulte (**python-re**), (**jeffrey**).

2.1.2 Programacao Funcional

Os conceitos de programacao funcional surgiram junto do calculo lambda, inventado por Alonzo Curch. Embora existam varias linguagens funcionais (ML, Lisp, Rackett, Haskell), todas elas compartilham essa origem. Tal como em programacao imperativa que diferencia linguagens entre procedural e orientada a objetos, existem diferencas entre as linguagens funcionais, mas primeiramente, o que eh programacao funcional. Segundo (**Bird**), "programcao funcional eh: um metodo para construcao de programas que enfatiza funcoes e suas aplicacoes ao invez de commandos e suas execucoes; programacao funcional faz uso de notacao matematica 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 proprias particularidades, porem grande partes dos conceitos apresentados sera geral para linguagens funcionais.

Haskell faz parte do conjunto de linguagens conhecidas como "linguagens funcionais puras". Uma linguagem pura permite a *definicao* de simbolos uma unica vez, exemplo, ao definir "let a = 4" em um programa, tentar mudar o valor da variavel "a" resulta em um error. O valor de um simbolo nao pode mudar durante a execucao do programa, de certa forma isso eh equivalente as keywords "final" do java e "const" da linguagem C. Embora nao poder atualizar o valor de uma variavel soe como uma grande desvantagem, isso permite o que pode ser chamado de "funcoes sem efeito-colateral". Isso significa que uma funcao jamais ira alterar o estado do programa fora dela, toda vez que uma funcao for chamada com os mesmos atributos, ela ira retornar o mesmo valor, isso nao eh o caso em linguagens orientadas a objeto, por exemplo, onde metodos alteram o estado do objeto (setters). O conceito de funcoes sem efeito colateral eh uma vantagem pois isso facilita o entendimento do programador sobre a sequencia de eventos do programa, nao existe preucupacao de que uma funcao altere alguma variavel 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 voce esta vindo do mundo imperativo, nos vimos que isso eh algo realmente legal. Em uma linguagem imperativa voce nao tem

garantia de que uma simples funcao que deveria somente processar alguns numeros nao ira queimar sua casa, sequestrar seu cachorro e riscar seu carro com uma batata enquanto processa aqueles numeros.".

Como nao podemos alterar o estado de uma variavel em um programa funcional, isso nos forca a procurar diferentes maneiras de resolver problemas computacionais. Um exemplo classico disso eh o problema de calcular o fatorial de um numero. O fatorial de n (n!) eh definido como $n! = (1) * (2) * ... * (n-1) * (n)$. Em uma linguagem imperativa, esse problema pode ser resolvido com um for, uma variavel acumuladora e uma variavel contadora; como nao podemos mutar o valor de definicoes em linguagens funcionais, podemos fazer usso de recurssao.

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 eh 0 e retorna o valor n * (n-1)! para qualquer outro valor. Essas curiosa sintaxe faz uso de um construtor muti util do Haskell chamado de *patten matching*. Segundo **??**, "pattern matching consiste de especificar padroes para o qual algum dado deve tomar forma, verificar se o dado acorda com esses padroes", eh possivel definir varios padroes para uma unica funcao. Padroes sao executados de cima para baixo e por isso padroes mais gerais (dao match em um maior numero de situacoes) devem ser posicionados por ultimo.

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 padrao (x:xs) define que x eh 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 tera o valor de x = 1 e xs = [2,3]. Usando esse construtor, podemos definir a funcao sum que soma todos os numero em uma lista.

sum :: [Int] -¿ Int sum [] = 0 sum (x:xs) = x + sum xs

O codigo acima define uma funcao sum que recebe uma lista de ints ("[Int]") e retorna um int, a funcao sum faz uso de dois pattern matchings. O primeiro padrao "[]" faz match com uma lista vazia, o valor da funcao sum para a lista vazia eh definido como o valor 0. Caso a lista nao seja vazia, o segundo padrao eh executado, e para aquele caso a soma de uma lista eh definida como o primeiro elemento mais a soma do restante da lista. Novamente, isso eh uma funcao que faz uso de recursao para resolver o problema computacional. A funcao sum chama a ela mesma ate o valor de xs ser a lista vazia, para o qual definimos que isso seria igual a 0. Caso omitissimos o primeiro padrao, a funcao iria chamar a ela mesma indefinitvamente. Pattern matching eh muito util em funcoes recursivas pois nos permite definir a condicao de saida da recurssao.

2.1.3 Algoritimos de regex

Nessa secao sera abordado um pouco sobre maquinas de estado, como elas podem ser usadas para representar expressoes regulares. Talvez abordar a diferenca entre maquinas deterministicas e nao deterministicas, e como uma maquina nao deterministica pode ser transformada em deterministica.

Ainda estou em duvida qual algoritimo sera utilizado para a construcao da biblioteca, irei pesquisar mais sobre isso.

2.2 METODOLOGIA

Utilizando os algoritimos abordados na secao de algoritimos, sera criado um modulo escrito funcionalmente em Haskell para realizar a pesquisa de regexes em um texto.

Tenho duvida a respieto dessa secao. O que faz parte do escopo de metodologia? Praticas de desenvolvimento aplicadas como testes unitarios?

Acredito que a maior parte do conteudo abordado aqui seja estabelecido apos a escolha do algoritmo(s) de regex a serem implementados.

A definicao do dialeto da regex, junto dos simbolos especiais, flags e etc deve ser feita nessa secao ou na fundamentacao teorica?

3 CRONOGRAMA

Seq.	Fases	Atividades	Projeto de pesquisa
1	1a Fase do projeto de pesquisa	Definicao do tema	Dez/19, Jan/20
1	1a Fase do projeto de pesquisa	Estudo Haskell	Dez/19 - Abr/20
2	1a Fase do projeto de pesquisa	Pesquisa Regex e algoritimos	Fev/20 - Abr/20
3	1a Fase do projeto de pesquisa	Escrever projeto de pesquisa	Jan/20 - Mar/20
4	2a Fase do projeto de pesquisa	Planejar Algoritimo	Abr/20
5	2a Fase do projeto de pesquisa	Implementar modulo	Abr/20 - Jun/20
6	2a Fase do projeto de pesquisa	Escrever Relatorio	Jun/20 - Jul/20
7	2a Fase do projeto de pesquisa	Revisao	Jun/20 - Jul/20

4 CONSIDERACOES FINAIS

O objetivo desse trabalho foi expor alguns conceitos por traz do paradigma funcional e demonstrar como esses conceitos sao uteis atraves da construcao de um modulo de processamento de expressoes regulares. O modulo foi implementado usando a linguagem Haskell, uma linguagem funcional pura.

Primeiramente foi abordado as origens da programacao funcional, tal como as principais diferencas 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 traz das expressoes regulares (regexes) e alguns casos onde sao uteis, juntamente com parte da sua historia. Por ultimo foi discutido diferentes metodos para se resolver o problema computacional referente a pesquisa de expressoes regulares.

Usando os conceitos apresentados foi demonstrado o processo de construcao do modulo de regex.