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

Bruno Gomes

25 de março de 2020

1 INTRODUCAO

Qualquer estudante ou profissional que esteja envolvido no ambiente da tecnologia da informação certamente já teve contato com alguma espécie de linguagem de programação em algum ponto de sua jornada. De acordo com uma pesquisa feita pelo StackOverflow, as 5 linguagens de programação mais usadas são: JavaScript, Python, Java, Linguagens de script (Bash, Powershell, Shell) e C# (**stack-overflow**). Embora essa lista de linguagens possa parecer como um conjunto heterogêneo de tecnologias, divergindo fortemente em convençoes e nichos de uso, todas essas linguagens fazem parte da família de linguagens conhecidas como imperativas. Inquestionavelmente, as liguagens imperativas são muito importantes, pois as mesmas compõem a maioria do código sendo produzido diariamente, porém não são a única família de linguagens existentes. Nesse trabalho irá ser discutido o paradigma de programação funcional, uma alternativa ao paradigma imperativo que domina o mercado.

O foco desse trabalho será a criação de um módulo em Haskell para realizar buscas em textos usando expressões regulares. Durante essa jornada serão feitas comparações entre algoritmos escritos de maneira imperativa e funcional, usando as linguagens Python e Haskell, respectivamente. As expressões regulares partem da teoria da computação, mais especificamente da teoria das automatas. Será definida a teoria as automatas, como elas são capazes de processar expressões regulares e finalemente será abordado a implementação do módulo em Haskell para a busca em texto.

Embora o paradigma funcional seja muito menos disseminado, ele é de extrema importância e possuem um grande impacto fora de seu nicho. as descobertas e inovações Diferentes inovações e descobertas no paradgima funcional, de certo modo infecta as linguagens imperativas. Como exemplo disso temos que a partir da versão 8 do Java, foram introduzidas interfaces funcionais e *arrow functions*, conceitos esses que surgiram a programação funcional. Na linguagem Python, outro gigante da programação imperativa, existem as *list comprehenssions*, uma maneira idiomática de se construir listas em Python. Esse recurso muito amado da linguagem foi inspirado em um recurso muito similar que existe na linguagem Haskell.

Em conclusão, embora as linguagem funcionais sejam muito menos comum, elas definitivamente deixaram e continuam deixando marcas nos gigantes da pro-

gramação. Elas transcendem seu pequeno nicho de usuários e afetam a grande maiorias das pessoas que produzem código regularmente, mesmo que muitos não tenham ciência disso. Sendo assim, esse trabalho tem como objetivo introduzir o paradigma funcional, focando na linguagem Haskell, realizando comparações entre os dois paradigmas e discutindo a maneira funcional de resolver certos problemas computacionais.

2 EXPRESSÕES REGULARES E PROGRAMAÇÃO FUNCIONAL

Como foi abordado na introdução, esse trabalho une dois temas: expressões regulares e programação funcional. Esses temas serão, primeiramente, discutidos separadamente, e em seguida será feito uma previa de como será feita a construção do motor de processamento de expressões regulares.

2.1 Expressão Regular

2.1.1 Introdução as expressões regulares

Expressões regulares, tambem conhecidas como *regex* (da junção do nome em inglês, *regular expression*) são utilizadas para realizar buscas complexas sobre strings. Para Cox, "expressões regulares são uma notação que descreve um conjunto de strings. Quando alguma string está no conjunto descrito pela expressão regular, pode-se dizer que essa expressão regular corresponde a esse string."(**cox**).

As regexes são utilizadas frequentemente, tanto para extrair informações que seguem um padrão ou para realizar buscas mais flexíveis ou parciais. Como exemplo, suponha o problema de extrair todas os strings que correspondem a um horario em um texto. A escrita de um horario segue uma estrutura padrão, HH:MM:SS onde HH delimita as horas, MM delimita os minutos e SS delimita os segundos. Sem ter que construir todas as possíveis combinações de horas que seguem esse formato, uma simples varredura de texto é incapaz de extrair essa informação. Esse problema pode ser resolvido tranquilamente usando regexes.

Uma expressão regular que realiza esta busca é,

$$[0-9]2:[0-9]2:[0-9]2. \tag{1}$$

Em palavras, os símbolos [0-9] representa qualquer carácter numerico entre 0 e 9. O token 2 indica uma repetição, sendo equivalente à regex [0-9][0-9], ou seja dois carácteres numericos. O carácter : é interpretado como o símbolo dois pontos literal. Fazendo a união, a regex acima equivale a qualquer string que tenha o formato DD:DD:DD onde D indica qualquer digito de 0-9. Podemos ver que esse formato é exatamente o formato definido anteriormente.

É importante ressaltar que existem inumeras variações e implementações de regexes, onde existem diferentes meta-characteres para descrever operações. Em (**mastering**), o autor discute as diferenças em regex entre as linguagens: PHP, .NET, Java e Perl. Na documentação oficial da linguagem Python (**python-re**) é dito que o dialeto usado é basedo nas expressões regulares da linguagem Perl com alguns adicionais. Usuários UNIX também estão familiarizados com os *wildcards*

presentes nos shells. Em resumo, existem vários dialetos porém os o objetivo das regexes não se altera, buscar por padrões. A regex acima e todas as regexes subsequentes nesse texto serão escritos no dialeto da linguagem Perl.

2.1.2 Computando regexes

2.2 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.

2.2.1 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.

2.3 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.