colloquium exactarum

EXPRESSÃO REGULAR NUMÉRICA

NUMERIC REGULAR EXPRESSION

Bruno Vier Hoffmeister, Marta Bez

Instituto de Ciências Exatas e Tecnológicas – Universidade - Novo Hamburgo, RS.
Correspondência para Bruno Vier Hoffmeister - bruno@hoffmeister.us

RESUMO - Este artigo apresenta a de finição formal da linguagem de programação Expressão Regular Numérica. Um conceito de linguagem inspirado na sintaxe da Expressão Regular que traz o seu poder e flexibilidade para cadeias numéricas é descrito.
Palavras-chave: linguagem de programação; expressões regulares; detecção de padrões; gramatica regular; variações numéricas; numeric regex.

ABSTRACT - This article defines the formal definition of the computer program language Numeric Regular Expression. A language concept inspired by Regular Expression syntax, applying your power and flexibility to numeric chains are describe.
Keywords: programming language; regular expressions; pattern detection; regular grammar; numerical variations; numeric regex.

Recebido em:20/11/2013
Revisado em: 15/01/2014
Aprovado em:30/01/2014

1 INTRODUÇÃO

A linguagem de programação *Perl* foi pioneira na incorporação das expressões regulares que hoje estão presentes em quase todas as linguagens de programação. Baseada na biblioteca escrita por Henry Spencer, as expressões regulares do *Perl* simplificam a busca em textos. Sua implementação utiliza o conceito de cadeias de caracteres para casar de forma dinâmica os textos. A simples expressão regular [A-Z] casará com qualquer caractere seguindo a tabela ASCII que está no intervalo de A (65) à Z (90). Ao acrescentar o modificador "+" no final da expressão, a busca se repetirá, caractere por caractere, até que o mesmo não case com a expressão ou o texto acabe. Aplicando esse conceito a sequências de números, gera-se a expressão regular [0-9]+, que casará com qualquer caractere que esteja no intervalo de 0 (48) à 9 (57), seguindo a tabela ASCII.

Se for necessário buscar números de 0 à 12 em um texto, uma expressão regular que buscaria esses valores pode ser representada por ([0-9]|10|11|12). Quanto mais específicas forem as regras para a busca de números, maior e mais complexa será a expressão regular, dessa forma, sendo impossível, em alguns casos, gerar uma expressão correspondente.

Este artigo foi desenvolvido a partir da necessidade de gerar algoritmos eficientes e de rápida construção para análise de padrões técnicos de variação na bolsa de valores. A partir da abstração dos algoritmos empregados na detecção da maioria dos padrões econômicos foi desenvolvido a linguagem, caracterizando-se portanto a natureza metodológica como aplicada. Com o procedimento técnico classifica-se como experimental, pois constitui-se na criação de uma nova linguagem que pode ser utilizada para detecção de padrões de variações numéricas nas mais diversas áreas, como bolsa de valores, análise de imagens, entre outras.

No intuito de otimizar a busca por sequencias numéricas muda-se o conceito de busca por cadeias de caracteres para cadeias numéricas. Ao invés de processar caractere a caractere, analisa-se, número a número, sendo ele composto por múltiplos caracteres ou não. Se for necessário realizar a busca de um número de 0 a 12, uma expressão regular numérica correspondente pode ser [0:12]. Nesse artigo será apresentada a definição formal da Expressão Regular Numérica. Para isso, inicia-se apresentando o conceito de Expressões Regulares e uma das formas utilizadas para definir uma linguagem, o BNF. Por conseguinte, é abordada a definição formal da Expressão Regular Numérica e os

casos práticos de uso da linguagem. Por fim, é apresentando o repositório de desenvolvimento da linguagem.

2 EXPRESSÕES REGULARES

Expressões regulares são representações de padrões de cadeias de caracteres. Uma determinada expressão regular é completamente definida pelo conjunto de cadeias de caracteres com as quais ela casa (LOUDEN, 2004, p.34). Apesar das Expressões Regulares terem sido padronizadas pelo *Institute of Electrical and Electronics Engineers* (IEEE), a implementação da linguagem *Perl* ainda é a mais utilizada pelas linguagens de programação, por possuir mais funcionalidades e recursos.

Para maiores informações a respeito de expressões regulares vide Friedl (2006), que aborda o funcionamento prático da linguagem.

3 DEFINIÇÃO FORMAL DE UMA LINGUAGEM DE PROGRAMAÇÃO

Segundo Chomsky (1956), há quatro tipos de gramática: gramáticas regulares (3), gramáticas livres de contexto (2), gramáticas sensíveis a contexto (1) e gramáticas com estrutura de frase (0). Sendo de maior nível a gramática de tipo 0, quanto maior o nível da gramática, maior é a liberdade de suas regras.

Para Cooper e Torczon (2008, p.75), as gramáticas livres de contexto predominam no mundo das linguagens de programação modernas, onde estas são definidas por um autômato determinístico finito de pilha e de tipo 2 na hierarquia de Chomsky (COOPER; TORCZON, 2008, p.75).

Existem diversas formas de descrever a gramática de uma linguagem de programação, sendo a BNF (forma de Bakus-Naur) a mais utilizada para descrever linguagens de programação. Trata-se de uma notação recursiva de formalização de sintaxe de linguagem através da produção de gramáticas. Esta é representada através de símbolos que são substituídos recursivamente por outros símbolos da gramática gerando sentenças (JOSÉ NETO, 1987, p.48).

4 DEFINIÇÃO FORMAL DA EXPRESSÃO REGULAR NUMÉRICA

Expressão regular numérica é uma linguagem de programação livre de contexto com reconhecedor determinístico descendente (*top-down*), ou seja, a árvore de estrutura é construída a partir do símbolo raiz (inicial). Sua gramática é definida na Figura 1 através da forma de Bakus-Naur.

```
<Main> : <Commands>
       | <StartAnchor> <Commands>

  <StartAnchor>: '^'
  <EndAnchor>   : '$'

  <Commands>: <Number> ' ' <Commands>
          | <Integer> ' ' <Commands>
          | <Interval> <Commands>
          | <Number> '{' <Limiter> '}' <Commands>
          | <Integer> '{' <Limiter> '}' <Commands>
          | '(' <Commands> ')' <Commands>
          | '(' <Commands> ')' '{' <Limiter> '}' <Commands>
          | <EndAnchor> <EOF>
          | <EOF>

  <Interval>: '[' <IntervaNumber> ':' <IntervaNumber> ']'
          | '[' <IntervaNumber> ':' <IntervaNumber> ']' '{' <Limiter> '}'
          | '[' <IntervaNumber> ':' <IntervaNumber> ';' <Function> ']'
          | '[' <IntervaNumber> ':' <IntervaNumber> ';' <Function> ']' '{' <Limiter> '}'

  <IntervaNumber> : <Number>
              | <Integer>
              | <Percent>

  <Limiter>: <quantifiers>
          | <NumericLimiter>
          | <PercentageLimiter>
          | <PercentageLimiter> ';' <NumericLimiter>

  <quantifiers>      : '*'
                | '+'
                | '?'

  <NumericLimiter>: <Integer>
                  | <Integer> ':' <Integer>

  <PercentageLimiter>: <Percent>
                | <Percent> ':' <Percent>

  <Function>: <FunctionName>
          : <FunctionName> <Parameters>

  <Parameters> : <Number>
           | <Integer>
           | <Number> <Parameters>
           | <Integer> <Parameters>
```

```
<FunctionName> : [a-zA-Z][a-zA-Z0-9_]*

<Integer>   : [0-9]+
            | -[0-9]+

<Number>    : [0-9]+[,.][0-9]+
            | -[0-9]+[,.][0-9]+

<Percent>   : <Number> '%'
            | <Integer> '%'
```

Figura 1. Forma Bakus-Naur da Gramática da Expressão Regular Numérica

Os símbolos definidos na Forma Bakus-Naur são:

• <Main>: Símbolo inicial, sendo a raiz da construção da árvore da linguagem.

• <*StartAnchor*>: Âncora inicial, torna obrigatório que o casamento ocorra no primeiro item da sequência numérica.

• <*EndAnchor*>: Âncora final, torna obrigatório que o casamento encerre ao final da sequência numérica.

• <*Commands*>: Bloco de comandos. Regra sintática.

• <*Integer*>: Número inteiro. Identificação direta com a sequência numérica.

• <*Number*>: Número com casas decimais, é permitido o caractere ‘,’ e o caractere ‘.’ para identificação das casas decimais. Identificação direta com a sequência numérica.

• <*Percent*>: Número com casas decimais. Utilizado para identificar sequências e limitar repetições nos símbolos <*Interval*> e <*Limiter*>.

• <*Interval*>: Intervalo numérico. Ocorre o casamento quando o número da sequência numérica se encontra entre o intervalo de números <*Number*> ou apresenta uma variação entre o intervalo de percentuais <*Percent*>. O percentual é calculado através da variação do número atual e do número anterior da sequência. Caso não haja número anterior na sequência, também ocorrerá o casamento. O símbolo <*Function*> executará uma função previamente definida da linguagem ou definida pelo usuário em tempo de execução no intuito de validar através de um retorno booleano o intervalo.

• <*Function*>: Função externa da linguagem para validação do <*Interval*>.

• <*Limiter*>: Símbolo de repetição. Irá repetir o último símbolo enquanto suas condições forem verdadeiras. A condição numérica <*NumericLimiter*> especifica o intervalo do número absoluto de vezes que a repetição deverá ocorrer. Caso o número mínimo não seja atingido, a expressão não irá

casar. A condição percentual <*PercentageLimiter*> repetirá enquanto os valores oscilarem entre o intervalo percentual informado. O percentual de variação é calculado em relação ao primeiro número do primeiro símbolo da repetição. A condição numérica e a condição percentual podem ser combinadas. Por fim, o símbolo quantificador <*quantifiers*> abrevia algumas especificações de repetição:

- *: Casa enquanto o símbolo a ser repetido for verdadeiro.
- *+*: Casa enquanto o símbolo a ser repetido for verdadeiro, porém torna obrigatória a existência de pelo menos um casamento.
- *?*: Casa com no máximo uma repetição, não sendo a mesma obrigatória.

5 CASOS DE USO

Os dois principais blocos de construção da linguagem são o intervalo (*Interval*), representando por dois colchetes [] e o limitador (*Limiter*), representado por duas chaves {}.

O intervalo casa com um número que obedece as especificações informadas, variando entre dois valores absolutos ou dois percentuais de variação em relação ao número anterior. É possível aplicar uma das funções pré-definidas da linguagem, assim como, adicionar funções customizadas. Esta estrutura, sendo a função de validação opcional, é representada dessa forma:

[valor absoluto inicial ou percentual mínimo de variação : valor absoluto final ou percentual máximo de variação ; função de validação]

O limitador repete o último bloco de operação da *pattern* da expressão regular numérica de acordo com as especificações informadas. É possível exigir um número mínimo e/ou máximo de repetições para o bloco anterior em conjunto com um intervalo de variação percentual em relação ao início das repetições, conforme a estrutura do limitador:

{variação percentual mínima : variação percentual máxima ; número mínimo de repetições : número máximo de repetições}

As expressões regulares numéricas podem ser utilizadas para operações simples como encontrar sequências crescentes de pelo menos cinco números variando de zero à mil “[0:1000;asc]{5}”. Utilizando os mesmos parâmetros, é possível aplicar uma função de média “[0:1000;avgasc]{5}” que casará com no mínimo 5 números crescentes com variação superior a média dos anteriores, conforme apresentado na Figura 2. Outras funções matemáticas podem ser aplicadas, tais como médias móveis, médias exponenciais e etc. A linguagem permite a

criação de funções personalizadas de validação.

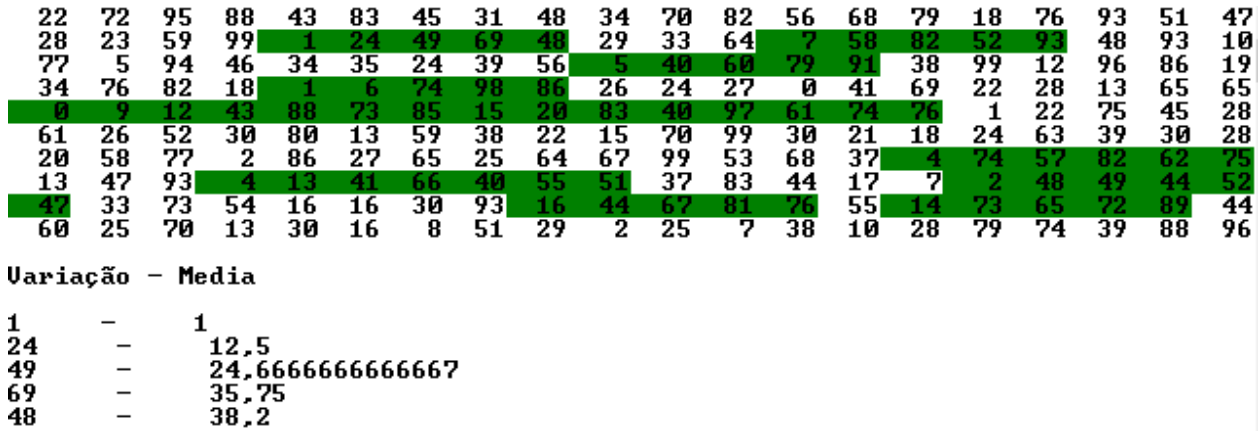

Figura 2. *Pattern* “[0:1000;avgasc]{5}” aplicada a número aleatórios

Outro exemplo de uso é na análise de imagens digitais. Uma imagem monocromática é uma função de intensidade de luz bidimensional *f(x, y)*, onde *x* e *y* denotam coordenadas espaciais e o valor de *f* no ponto (*x, y*) é proporcional ao brilho (ou nível de cinza) da imagem neste ponto (GONZALEZ; WOODS, 2008). Um exemplo de uso seria verificar se uma imagem possui o formato xadrez, variando entre as cores preto e branco utilizando a *pattern* “^([0]{5}[255]{5})+$”. A expressão irá casar sucessivamente com cinco *pixels* da cor preta seguidos de cinco *pixels* da cor branca. A Figura 3 apresenta uma imagem xadrez com as suas cores convertidas para valores numéricos (0 = preto, 255 = branco). Através dessa matriz é possível validar utilizando as expressões regulares numéricas se a imagem possui um formato xadrez. Para que seja confirmada a formação de quadrados perfeitos é necessário percorrer a matriz verticalmente e horizontalmente. Também é possível aplicar a linguagem para detectar erros de descontinuidade em imagens como um pico de variação de tonalidade superior a 10% utilizando a *pattern* “[10%:1000%]”.

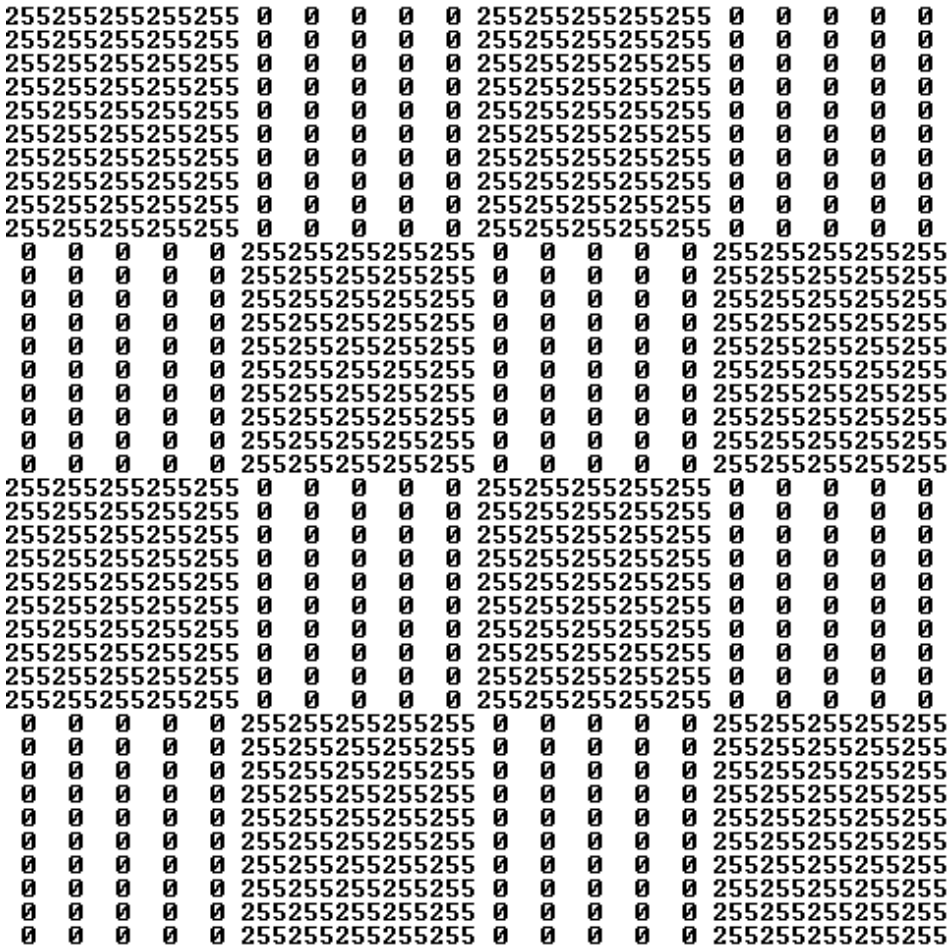

Figura 3. Imagem monocromática em formato xadrez (4x4).

6 CONSIDERAÇÕES FINAIS

A linguagem aplica o poder e a flexibilidade que as expressões regulares possuem em cadeias de caracteres em cadeias numéricas. Algoritmos complexos, extensos e confusos podem ser portados para apenas uma linha da linguagem. Um relatório avançado customizado de uma aplicação financeira pode ser rapidamente gerado e extraído com a linguagem, sendo muito simples criar identificadores de padrões, principalmente na análise técnica. As aplicações da linguagem são vastas, desde análise de imagens (procura por padrões de variações de cores) à *triggers scripts* em uma aplicativo *trader* da Bolsa de Valores.

O conceito da linguagem abordada nesse artigo está atualmente apenas implementado na linguagem C# .net framework 4.0. Sua primeira versão está disponível em https://numericregex.codeplex.com/.

REFERÊNCIAS

CHOMSKY, N. **Three models for the description of language**. IRE Transactions on Information Theory**,** n.2, p.113–124, 1956. http://dx.doi.org/10.1109/TIT.1956.1056813

COOPER, K.D.; TORCZON, L. **Engineering a compiler.** San Francisco, CA: Morgan Kaufmann, 2008.

FRIEDL, J.E.F. **Mastering Regular Expressions.** USA: O'Reilly Media, 2006.

JOSÉ NETO, J. **Introdução à compilação**. Rio de Janeiro: LTC, 1987.

LOUDEN, K.C. **Compiladores: princípios e práticas**. São Paulo, SP: Cengage Learning, 2004.

GONZALEZ, R; WOODS, R. **Digital Image Processing**. 3. ed. Prentice Hall, 2008.