Linguagens Formais e Autômatos

Aula 10 - Expressões regulares na prática

Prof. Dr. Daniel Lucrédio
Departamento de Computação / UFSCar
Última revisão: ago/2015

Referências bibliográficas

- **Introdução à teoria dos autômatos, linguagens e computação / John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman** ; tradução da 2.ed. original de Vandenberg D. de Souza. - Rio de Janeiro : Elsevier, 2002 (Tradução de: Introduction to automata theory, languages, and computation - ISBN 85-352-1072-5)
  - Capítulo 3 - Seção 3.3
  - Capítulo 4 - Seção 4.3

ER na prática

- Exemplos até agora são muito simples
- Alfabeto binário {0,1} ou limitado
- Na vida real, o alfabeto é maior
  - Existem classes de símbolos
- Pode ser inconveniente escrever expressões assim
- Ex: identificadores em uma linguagem de programação
  - (a+b+c+d+e+f+g+h+i+j+k+l+m+n+o+p+q+r+s+t+u+v+w+x+y+z+A+B+C+D+E+F+G+H+I+J+K+L+M+N+O+P+Q+R+S+T+U+V+W+X+Y+Z+_)(0+1+2+3+4+5+6+7+8+9+a+b+c+d+e+f+g+h+i+j+k+l+m+n+o+p+q+r+s+t+u+v+w+x+y+z+A+B+C+D+E+F+G+H+I+J+K+L+M+N+O+P+Q+R+S+T+U+V+W+X+Y+Z+_)*

ER na prática

- Existem outras notações melhores
- Exs:
  - UNIX (POSIX – IEEE Std 1003.1-2008)
  - Java Regex API

Java Regex

- Caracteres (Extraído da API do Java):

x	The character *x*
\\	The backslash character
\0*n*	The character with octal value 0*n* (0 <= *n* <= 7)
\0*nn*	The character with octal value 0*nn* (0 <= *n* <= 7)
\0*mnn*	The character with octal value 0*mnn* (0 <= *m* <= 3, 0 <= *n* <= 7)
\x*hh*	The character with hexadecimal value 0x*hh*
\u*hhhh*	The character with hexadecimal value 0x*hhhh*
\t	The tab character ('\u0009')
\n	The newline (line feed) character ('\u000A')
\r	The carriage-return character ('\u000D')
\f	The form-feed character ('\u000C')
\a	The alert (bell) character ('\u0007')
\e	The escape character ('\u001B')
\c*x*	The control character corresponding to *x*

Java Regex

- Classes de caracteres (Extraído da API do Java)

[abc]	a, b, or c (simple class)
[^abc]	Any character except a, b, or c (negation)
[a-zA-Z]	a through z or A through Z, inclusive (range)
[a-d[m-p]]	a through d, or m through p: [a-dm-p] (union)
[a-z&&[def]]	d, e, or f (intersection)
[a-z&&[^bc]]	a through z, except for b and c: [ad-z] (subtraction)
[a-z&&[^m-p]]	a through z, and not m through p: [a-lq-z](subtraction)

Java Regex

- Classes predefinidas de caracteres (Extraído da API do Java)

.	Any character (may or may not match line terminators)
\d	A digit: [0-9]
\D	A non-digit: [^0-9]
\s	A whitespace character: [\t\n\x0B\f\r]
\S	A non-whitespace character: [^\s]
\w	A word character: [a-zA-Z_0-9]
\W	A non-word character: [^\w]

ER na prática

- Normalmente ERs são usadas em busca textual
  - Ex: entrada “abracadabra”, ER = “cad”
  - Resultado = 1 correspondência (ou match) da posição 4 até 6 (começando em 0)
- Ou como forma de aceitação de entrada
  - Ex: [:digit:]+(\.[:digit:]+)?
- Se a expressão é determinística, tudo bem
  - Mas se não for, há um problema!

Java Regex

- Quantificadores (extraído do “The Java Tutorial”)
  - http://docs.oracle.com/javase/tutorial/essential/regex/quant.html

Quantifiers			Meaning
Greedy	**Reluctant**	**Possessive**	
X?	X??	X?+	*X*, once or not at all
X*	X*?	X*+	*X*, zero or more times
X+	X+?	X++	*X*, one or more times
X{n}	X{n}?	X{n}+	*X*, exactly *n* times
X{n,}	X{n,}?	X{n,}+	*X*, at least *n* times
X{n,m}	X{n,m}?	X{n,m}+	*X*, at least *n* but not more than *m* times

Java Regex

- Quantificadores
  - Ex: entrada vazia “”
  - Expressão a?
    - Reconhece a entrada
    - Correspondência “” na posição 0-0
  - Expressão a*
    - Reconhece a entrada
    - Correspondência “” na posição 0-0
  - Expressão a+
    - Não reconhece a entrada

Java Regex

- Quantificadores
  - Ex: entrada “a”
  - Expressão a?
    - Reconhece a entrada
    - Correspondência “a” na posição 0-1
    - Correspondência “” na posição 1-1
  - Expressão a*
    - Reconhece a entrada
    - Correspondência “a” na posição 0-1
    - Correspondência “” na posição 1-1
  - Expressão a+
    - Reconhece a entrada
    - Correspondência “a” na posição 0-1

Java Regex

- Quantificadores
  - Ex: entrada “aaaaa”
  - Expressão a?
    - Não reconhece a entrada
    - Correspondências “a” nas posições 0-1, 1-2, 2-3, 3-4, 4-5
    - Correspondência “” na posição 5-5
  - Expressão a*
    - Reconhece a entrada
    - Correspondência “aaaaa” na posição 0-5
    - Correspondência “” na posição 5-5
  - Expressão a+
    - Reconhece a entrada
    - Correspondência “aaaaa” na posição 0-5

Java Regex

- Quantificadores
  - Ex: entrada “abababaab”
  - Expressão a?
    - Não reconhece a entrada
    - Correspondências “a” nas posições 0-1, 2-3, 4-5, 5-6
    - Correspondência “” nas posições 1-1, 3-3, 6-6, 7-7
  - Expressão a*
    - Não reconhece a entrada
    - Correspondência “a” nas posições 0-1, 2-3
    - Correspondência “aa” na posição 4-5
    - Correspondência “” nas posições 1-1, 3-3, 6-6, 7-7
  - Expressão a+
    - Não reconhece a entrada
    - Correspondência “a” nas posições 0-1, 2-3
    - Correspondência “aa” na posição 4-5

Java Regex

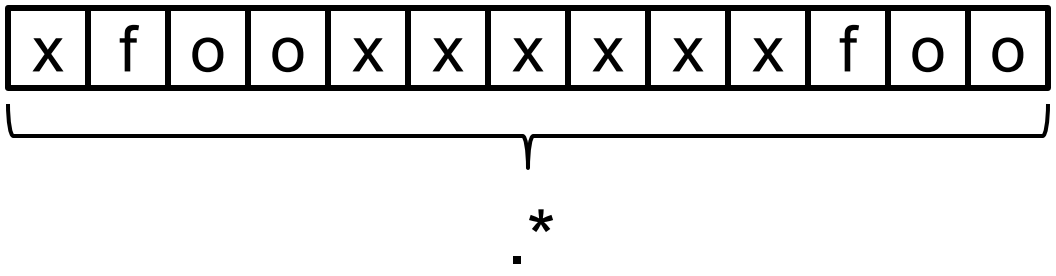
- Pode-se usar quantificadores com classes:
  - Ex: [abc]+
  - Encontra: accccabab
- Pode-se usar quantificadores com grupos:
  - Ex: (abc)+
  - Encontra: abcabcabc
- Pode-se definir o número de repetições
  - Ex: (dog){3}
  - Encontra: dogdogdog

Java Regex

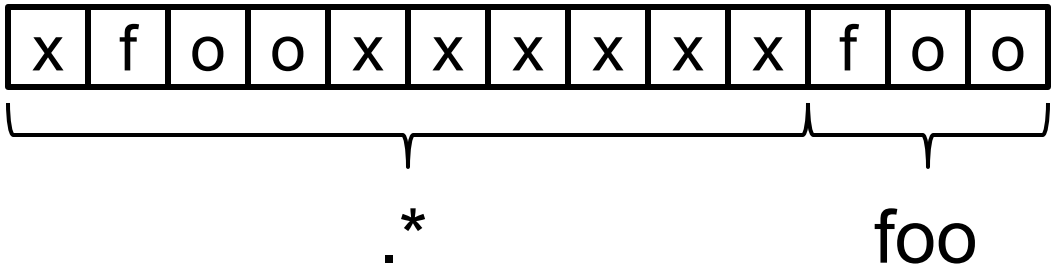
- Diferença entre os quantificadores
- Gananciosos
  - Consomem a entrada toda, buscando sempre a maior correspondência
    - Depois de consumir o máximo de caracteres, volta analisando os caracteres da direita para a esquerda
- Relutantes
  - Consomem a entrada necessária somente, buscando sempre a primeira correspondência
- Possessivos
  - Consomem a entrada toda
    - Se conseguirem uma correspondência, ótimo
    - Se não conseguirem, não voltam atrás (como os gananciosos)

Java Regex

- Operador ganancioso (ER=.*foo)
  - Num primeiro momento, “.*” consome tudo

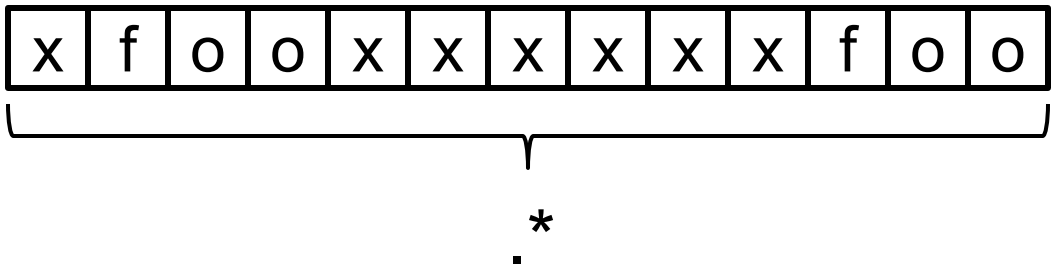


  - Em seguida, ele verifica se a ER foi reconhecida
    - Como não foi, ele volta da direita para a esquerda, um caractere por vez, até conseguir uma correspondência



Java Regex

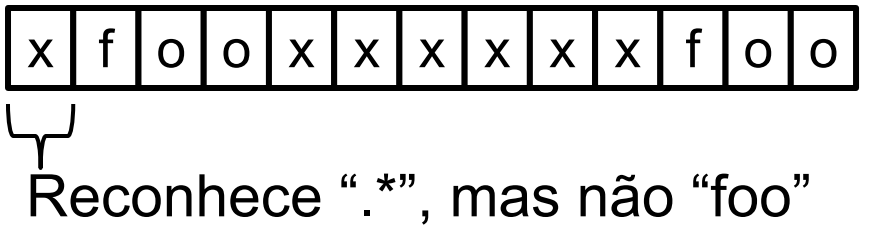
- Operador possessivo (ER=.*+foo)
  - Num primeiro momento, “.*” consome tudo

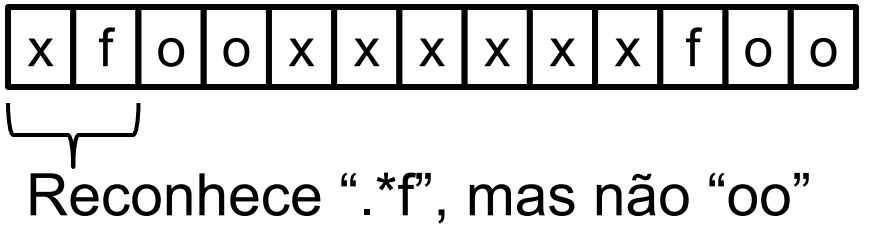


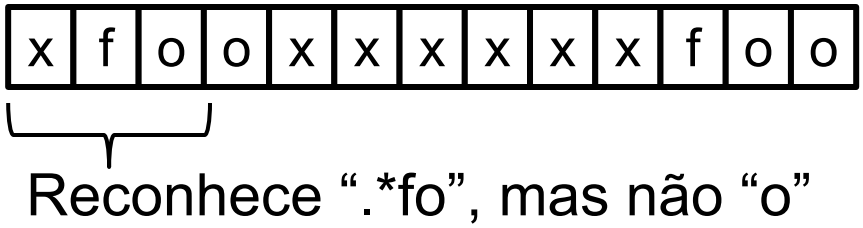
  - Em seguida, ele verifica se a ER foi reconhecida
    - Como não foi, ele não volta da direita para a esquerda (como o ganancioso). Como resultado, nenhuma correspondência é encontrada!

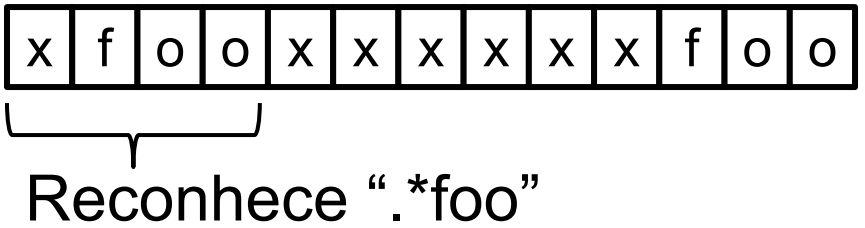
Java Regex

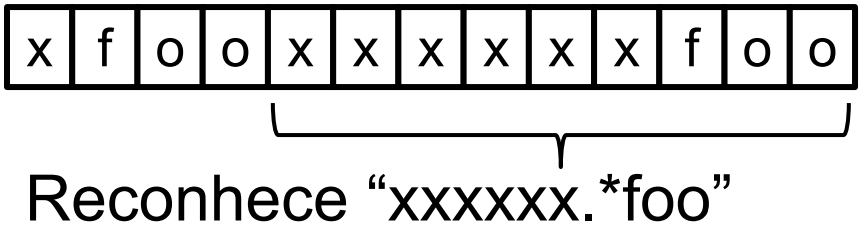
- Operador relutante (ER=.*?foo)
  - A leitura é feita da esquerda para a direita, até encontrar uma correspondência da ER toda











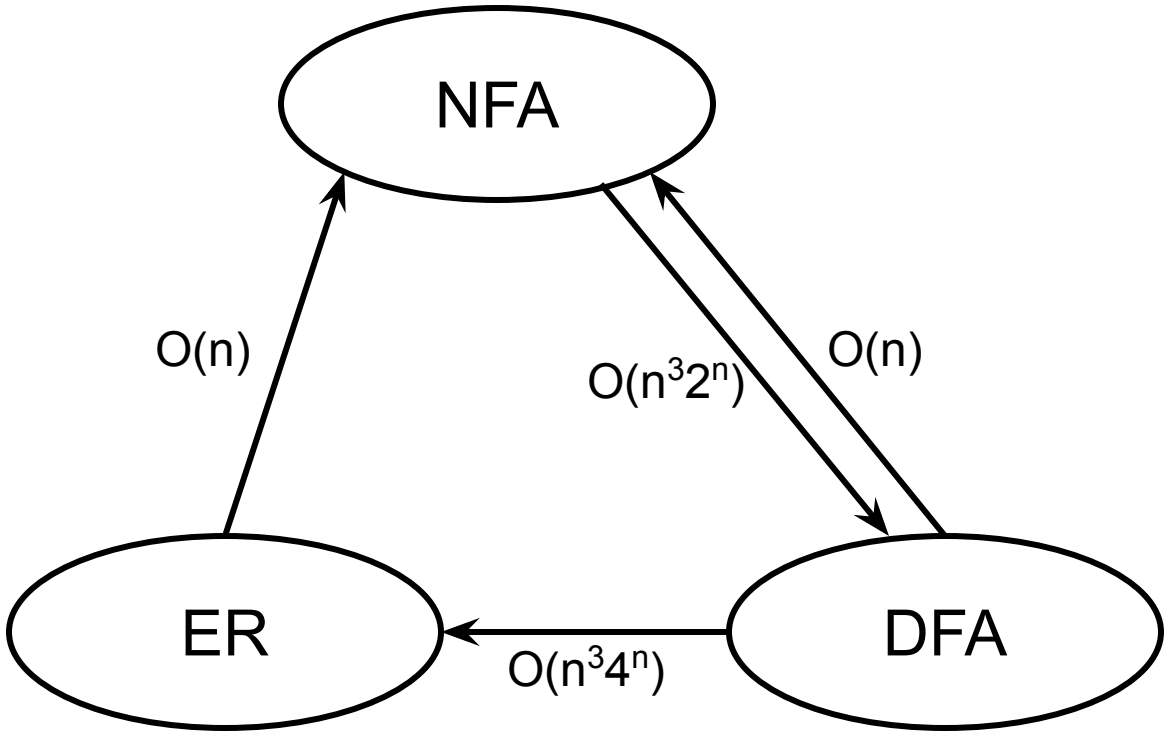
Expressões regulares na prática

- Exemplos:
  - http://regexlib.com/
- Identificadores simples (sem outros caracteres):
  - `[a-bA-B][0-9a-bA-B]*`
- Comentários:
  - Uma linha: `//.*$` ($ indica fim de linha)
  - Múltiplas linhas (desde que o . esteja configurado para reconhecer fim de linha): `/\*.*?\*/`
- Strings:
  - Simples (aspas não podem aparecer dentro): `"[^"\r\n]*"`
  - Com aspas (ou outros caracteres especiais precedidos por barras: \") dentro: `"[^"\\\r\n]*(\\.[^"\\\r\n]*)*"`

Questões sobre linguagens regulares

- Linguagens regulares existem sob muitas formas
  - DFA
  - NFA
  - ε-NFA
  - Expressões regulares
- Cada uma tem suas características
  - DFA = rápida execução
  - NFA (e ε-NFA) = mais fácil projeto, porém execução mais lenta
  - ER = boa legibilidade e projeto fácil
- É possível passar de uma para outra

Conversão entre representações

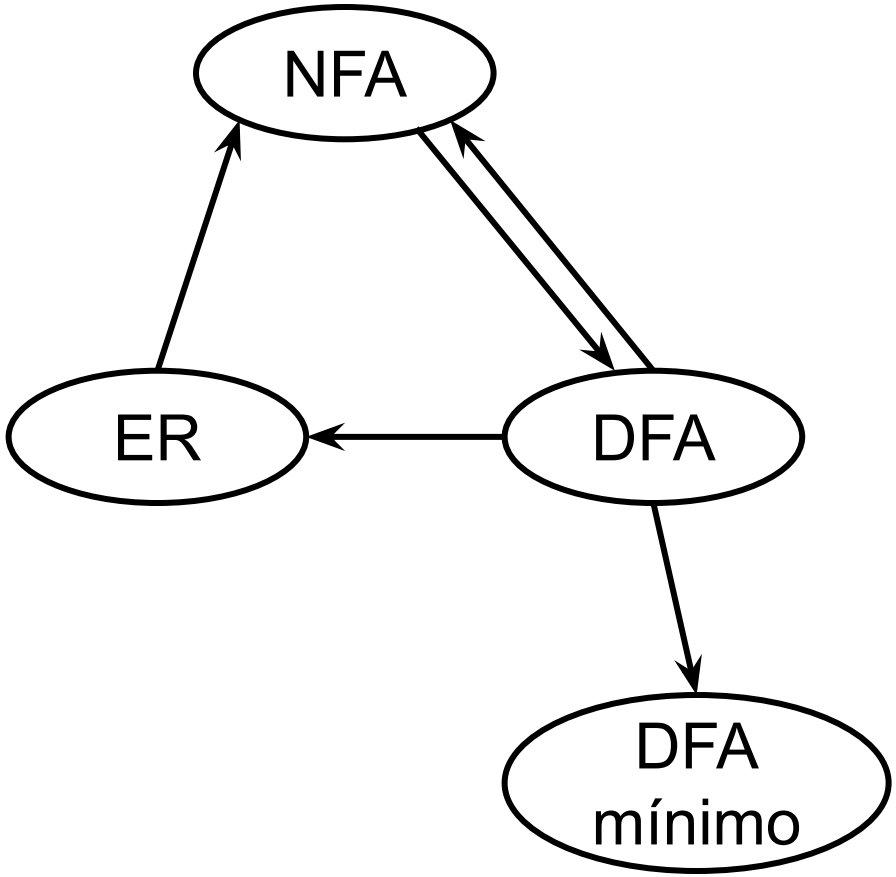


Conversão entre representações

- Normalmente especifica-se uma ER, pois é a versão mais legível
- Decisão sobre implementação depende do uso
  - Em alguns casos é preferível “executar” expressões regulares diretamente
    - Ex: comandos de busca textual, pois a cada nova busca, a expressão muda
  - Em outros casos, é preferível converter para um DFA
    - Ex: análise léxica, pois as expressões são fixas, e são executadas inúmeras vezes
    - Neste caso, uma otimização adicional é possível

Conversão entre representações

- Minimização proporciona execução mais rápida
- Especialmente útil em compiladores, pois uma vez implementado, o DFA não irá mudar
  - Vale a pena o esforço extra



Fim

Aula 10 - Expressões regulares na prática