Automata and Grammars (BIE-AAG)

8. Regular expressions - usage

Jan Holub

Department of Theoretical Computer Science
Faculty of Information Technology
Czech Technical University in Prague

European social fund.
Prague & EU: Investing in your future

Plan of the lecture

1. Introduction – how to describe sets of strings
2. What is and is not a regular expression
   Shell pattern matching vs. grep, perl
3. Full-featured regular expressions
4. Definitions, examples - Posix RE
   BRE, ERE
5. Definitions, examples – Perl RE
6. Match with the theory (Kleene)
7. RE support in programming languages
8. Implementation of regexp engines

1 - Introduction

How to describe sets of strings?

- Restricted regular expression (UNIX shell)
- Regular expression (RE)
- Extended regular expression (ERE)

Regular expression - how to name the type:

- Regular expression = regular expression by Kleene
- Regex, Regexp = posix or perl regular expression

1 - Introduction

How to describe and **decide** sets of strings?

- Restricted regular expression (UNIX shell): pattern matching
- Regular expression (RE): finite automaton
- Extended regular expression (ERE): matching engine with backtracking

2 - What is not RE yet

UNIX shell pattern

- Describes sets of strings over alphabet T, which are proper subsets of regular languages.
- Uses special metacharacters to express an arbitrary string, arbitrary symbol and arbitrary symbol from a given set of symbols
- Pattern is a sequence of symbols and metacharacters
- String matches pattern if it can be mapped onto the symbols of the pattern, symbol by symbol.

2 - UNIX Shell Pattern

Used metacharacters:

- $[\ ], [\hat{}\ ]$
- $?$
- $*$
- $\backslash$

2 - UNIX Shell Pattern

Used metacharacters:

- $[abc0\text{-}9]$, $[\hat{}d\text{-}z]$: describes a set of allowed and forbidden symbols (to forbid a symbol, one may also use ! instead of $\hat{}$). A range of symbols is described using a dash.
- $?$: question mark means arbitrary symbol
- $*$: asterisk means arbitrary string
- $\backslash$: “escape” symbol to allow insertion of $[, ?, *, \backslash$ into the pattern

2 - UNIX Shell Pattern

Examples of sets of strings and patterns that describe them:

strings	pattern
$a.txt$	$a.txt$
$a.txt, b.txt$	$[ab].txt$
$a.$<3 symbols>	$a.???$
a at the beginning and $.txt$ at the end	$a{*}.txt$
arbitrary string	$*$

2 - UNIX Shell Pattern

Examples of patterns and sets of strings that they describe:

pattern	strings
$main.c$	
$[a\text{-}ce].c$	
$?x?$	
$*.txt$	
$*$	

2 - UNIX Shell Pattern

Examples of patterns and sets of strings that they describe:

pattern	strings
$main.c$	$main.c$
$[a\text{-}ce].c$	$a.c, b.c, c.c, e.c$
$?x?$	$axa, axb, axc, \ldots$
$*.txt$	arbitrary beginning and $.txt$ at the end
$*$	arbitrary string

2 - POSIX symbol classes

Character classes that replace the ranges in square brackets. Contrary to ranges, posix character classes are exactly defined and are therefore useful when reliable evaluation is needed.

- Classes are desribed using $[: class\_name :]$ and can only be used inside range square brackets.
- $[[: alpha :]]$ is equivalent to $[a\text{-}zA\text{-}Z]$
- $[[: alnum :]]$ is equivalent to $[[: alpha :]01 - 9]$

$[: alpha :]$	$[: alnum :]$	$[: ascii :]$	$[: blank :]$	$[: cntrl :]$
$[: digit :]$	$[: graph :]$	$[: lower :]$	$[: print :]$	$[: punct :]$
$[: space :]$	$[: upper :]$	$[: word :]$	$[: xdigit :]$	

2 - UNIX Shell Pattern

If we want to describe a string that contains symbol(s) $*, \backslash, [, ?$, we need to prefix these symbols in the pattern with an escape symbol ’\’ or use ’”’
Moreover, shell interprets unescaped symbols in a peculiar manner:

- $ls\ [a].txt$: if there is $a.txt$, finds only $a.txt$
- $ls\ [a].txt$: finds $[a].txt$ if this exists and $a.txt$ does not exist

2 - Pattern - ending notes

- Meaning of metacharacters differs between Unix tools. For example, some tools will use $*$ to match strings containing '/' and other tools will not.
- When searching files, patterns $*$ and ?$*$ do not find files beginning with a dot. Use .$*$
- The searched pattern is commonly called **glob**

3 - Full-featured regular expressions

In practice, regular expressions are used commonly. They are based on Kleene's theory and extend it, depending on needs.
There are two main standards:

- Posix basic and extended RE (BRE and ERE)
- Perl regexp

3 - Full-featured regular expressions

Kinds of symbols in regular expressions:

- Literals (ordinary characters)
- Metasymbols (special characters)
  1. Anchors
  2. Character sets
  3. Modifiers and quantifiers
- Escape symbol

3 - Full-featured regular expressions

Meaning of symbols in regular expressions:

- Literals: mean themselves
- Metasymbols: have a special meaning
  1. Anchors: force a match to have a certain position in text
  2. Character sets: a match must use one of proposed symbols
  3. Modifiers and quantifiers: Allow operations on regular expressions like union and iteration
- Escape symbol: makes a literal out of a metacharacter

3 - Full-featured regular expressions

- String x **matches** regular expression r if x belongs to the set of string described by the regular expression r.

4 - Posix BRE

- BRE = Basic Regular Expressions
- A norm for UNIX regular expressions
- Most UNIX tools that work with regular expressions comply with this norm.

4 - BRE - Character sets

Describing a set of characters in BRE:

x	symbol x
.	arbitrary symbol
$[abc]$	a or b or c
$[a$-$c]$	a or b or c
$[\hat{}\,a$-$c]$	arbitrary symbol except a, b and c
$[[: alpha :][: space :]\#]$	arbitrary alphabet letter,
	space character or #

- BRE accepts posix character classes

4 - BRE - Quantifiers

Quantifiers follow a regular expression or its part. They correspond to iteration, but can be more fine-grained:

$*$	0 or more consecutive matches
$\backslash\{m, n\backslash\}$	from m to n consecutive matches
$\backslash\{m, \backslash\}$	m and more consecutive matches
$\backslash\{m\backslash\}$	m consecutive matches

- $\backslash$ before $\{$ and $\}$ is necessary.

4 - BRE - Quantifiers - examples

Quantifiers follow a regular expression or its part. They correspond to iteration, but can be more fine-grained:

Regular expression	described strings
$ba*$	$b, ba, baa, baaa, \ldots$
$[ba]*$	aribtrarily long string of symbols a and b
$\backslash(ba\backslash)*$	$\varepsilon, ba, baba, bababa$
$[ba]\backslash\{1,2\backslash\}$	b, a, bb, ba, ab, aa
$a\backslash\{3,\backslash\}$	$aaa, aaaa, aaaaa, \ldots$
$[ba]\backslash\{2\backslash\}$	bb, ba, ab, aa

- $\backslash$ before $\{$ and $\}$ is necessary.

4 - BRE - Anchors

Anchors are used for forcing a position of the match on the line.

Anchor	Meaning
ˆ	Beginning of the line
$\$$	End of the line
$\backslash$<	Beginning of word (non-standard, originally from vi)
$\backslash$>	End of word (non-standard, originally from vi)

- $\backslash$ before < and > is necessary.

4 - BRE - Kotvy - examples

Anchors are used for forcing a position of the match on the line.

Regular expression	Matches in	Rejects
ˆmer	**mer**ry Christmas	hopping **mer**rily
$ture\$$	long lec**ture**	imma**ture** adult
$\backslash$<sel	**sel**ect	mor**sel**
$tener\backslash$>	one swee**tener**	two swee**tener**s

4 - BRE - Special characters

Other characters in BRE:

Character	Meaning
$\backslash(\backslash)$	Group creation; groups are implicitly numbered to allow backreference
$\backslash numbern$	Repeated occurence of string matched in the n^{th} parenthesis (mechanism for remembering!)
$\backslash n$	New line character (non-standard; used e.g. by sed)

- $\backslash$ before (and) is necessary.

4 - BRE - Special characters

Other characters in BRE:

Regular expression	described strings
$\backslash(rock\backslash)$	$rock$
$\backslash(rock*\backslash)\backslash 1$	$rocroc, rockrock,$ $rockkrockk, \ldots$

- $\backslash$ before $($ and $)$ is necessary.

4 - BRE, ERE - POZOR!

All RE, unless specified otherwise, are greedy. That means they try to find the longest match possible:

Text:	$aabcaabcde$
Pattern:	$a.*b$
Match:	$aabcaab$

4 - BRE - various meaning of ’$\backslash$’

- If character ’$\backslash$’ occurs before $*, [, ., \backslash, \hat{}, \$$, it "turns off" the special meaning of these characters.
- If symbol ’$\backslash$’ occurs before $(,), <, >, \{, \}$, it "turns on" the special meaning of these characters.
- What does $[\backslash*\backslash[\,]\backslash\{2\backslash\}$ describe?

4 - Posix ERE

- BRE can describe some non-regular languages but at the same time cannot describe arbitrary regular language.
- Alternative (union) is missing
- Escape symbol has various meanings

That is why there exist Extended Regular Expressions (ERE), that try to solve problems of BRE.

4 - ERE - changes as against BRE

- Following metacharacters become literals:

  $\backslash\{\backslash\}$ $\backslash<\backslash>$

  $\backslash(\backslash)$ $\backslash number$
- New metacharacters for group:

  ( ) group parentheses

  | alternative in a group

ERE	described set of strings
$(ab\vert bc\vert cd)$	$\{ab, bc, cd\}$
$(wheat\vert barley)$	$\{what, barley\}$

4 - ERE - changes as against BRE

- New quantifiers:

quantifier	meaning
$*$	0 or more occurrences of match
$+$	1 or more occurrences of match
$?$	0 or one occurrence
$\{m, n\}$	from m to n occurences
$\{m, \}$	m and more occurences
$\{m\}$	m occurences

4 - ERE - changes as against BRE

- Quantifiers - examples:

Regular expression	described strings
$ba*$	$b, ba, baa, baaa, \dots$
$ba+$	$ba, baa, baaa, \dots$
$ba?$	b, ba
$[ba]*$	arbitrarily long string of symbols a and b
$(ba)*$	$\varepsilon, ba, baba, bababa$
$[ba]\{1, 2\}$	b, a, bb, ba, ab, aa
$a\{3, \}$	$aaa, aaaa, aaaaa, \dots$
$[ba]\{2\}$	bb, ba, ab, aa

4 - BRE and ERE - usage

- Usage in UNIX tools:

BRE	ERE
vi	awk
more	nawk
ed	egrep
grep	grep -E
sed	sed -r
...	...

4 - BRE a ERE - controversion

- Even though the posix norm tries to define the "correct" regular expressions, it does not provide an easy and ready-to-use formalism
- Almost every UNIX tool uses its own variant of regular expressions - it is important to test the RE before its use in practice.

5 - RE in Perl

- Perl is a scripting language
- It has a built-in support for regular expressions
- It builds on posix RE, extends them and facilitates work with them
- Some extensions have caused that the regexes describe a set of languages which is a proper superset of the set of all regular expressions.
- The standard is Perl 5.10 regexp

5 - RE in Perl

- Finding out whether a string $\$str$ contains an occurrence of regular expression $regex$:
  $\$str =\sim m/regex/;$
- Replacement of the first occurrence for another string:
  $\$str =\sim s/regex/replacement/;$
- Replacement of all occurrences for another string:
  $\$str =\sim s/regex/replacement/g;$

5 - RE in Perl

Perl builds on ERE and adds more functionality:

- Again, there is remembering and recalling using $\backslash number$
- Lazy quantifiers (beside greedy)
- Allows lookahead, lookbehind
- New, "shorthand" character classes
- New anchors

5 - RE in Perl

Unified metacharacters:

- $\{\ \}[\ ](\ )\hat{}\$.|*+?\backslash$ have a special meaning
- If they should be searched as literals, they need to be prefixed with $\backslash$

5 - RE in Perl

Unified metacharacters - examples:

Regular expression	described strings
$/(abc)\{1,2\}/$	$abc, abcabc$
$/[ab]?d/$	d, ad, bd
$/\hat{}.*\$/$	arbitrary full line
$/a{+}b\mid c{+}/$	$ab, aab, ..., c, cc, ...$
$/\backslash*\backslash\backslash\backslash+/$	$*\backslash+$

5 - RE in Perl

Escape sequences for ASCII characters:

- $\backslash n, \backslash r, \backslash t, \backslash x[hexa], \backslash u[Unicode],$

Match search modifiers:

- i = ignore case
- g = in case of replacements find and replace all occurrences. In case of repeated search start searching at the end of last match.
- ...

5 - RE in Perl

Examples:

Regular expression	described strings
$/hi\backslash nworld/$	hi $world$
$/entry\backslash t\backslash x61\backslash x74/$	$entry\quad at$
$/hello/i$	$hello, Hello, hEllo, ...$

5 - RE in Perl

Insertion of variables into regular expressions:

- $\$var = cas;$
- $/\$\{var\}tle/$ finds a match of string $castle$

New character classes:

- $\backslash d, \backslash D$ = $[0\text{-}9], [\hat{}0\text{-}9]$
- $\backslash s, \backslash S$ = [space characters], $[\hat{}\backslash s]$
- $\backslash w, \backslash W$ = [word characters], $[\hat{}\backslash w]$

New anchor $\backslash b$ finds boundaries of words.

5 - RE in Perl

Examples:

Regular expression	described strings
$\$x = int;$ $/\$\{x\}\backslash s{+}\backslash w{+};/$	$int\ a;, int\ b;, int\ ab;, \ldots$
$/\backslash b\backslash d{+}\backslash b/$	number (digits only)

5 - RE in Perl

Back-reference to a match enclosed in the regular expression using ( ):

- From inside the regular expression: \<serial number of the parenthesis>
- From outside the regular expression: $<serial number of the parenthesis>

Parenthesis that should not remember has the form $(? : regex)$

5 - RE in Perl

Lookahead - positive and negative (a match must be followed by something or must not be followed by something).

- $(?{=}regexp)$
- $(?!regexp)$

Lookbehind - positive and negative (a match must be preceded by something or must not be preceded by something). Regular expression used in the lookbehind pattern must be a fixed-width expression.

- $(?{<}{=}regexp)$
- $(?{<}!regexp)$

5 - RE in Perl

Examples:

Regular expression	described strings
$/([abc]+)d\backslash 1/$	$ada, bdb, cdc, aadaa, abdab, ...$ variable $\$1$: $a, b, c, aa, ab, ...$
$/(so\|ra)?ck/$	$sock, rack$ (to recall metacharacter $?$)
$/auto(?{=}maton)/$	$auto$ (after which $maton$ must follow)
$/(?{<=}sub)way/$	way (preceded by sub)

5 - RE in Perl

Greedy, lazy and possessive quantification

- All quantifiers are greedy by definition
- If we want to use a lazy quantifier, we suffix the quantifier with $?$, for possessive quantification we add suffix $+$

greedy	lazy	possessive
$*$	$*?$	$*+$
$+$	$+?$	$++$
$...$	$...?$	$...+$

5 - RE in Perl

Example: let us search in text $abcabcabcd$

regexp	match
$abc{*}d$	ano
$(abc){*}(abc){*}d$	yes, $\$1 = "abcabcabc", \$2 = ""$
$(abc){*}?(abc){*}d$	yes, $\$1 = "", \$2 = "abcabcabc"$
$(abc){*}{+}(abc){*}d$	yes, $\$1 = "abcabcabc", \$2 = ""$
$(abc){*}{+}(abc){+}d$	no

6 - Match with the theory (Kleene)

Posix and perl regular expressions are based on Kleene's thoery, but there are certain differences:

Kleene	BRE/ERE/Perl
RE describes regular languages	RE describes more (thanks to back-reference)
No context	Anchors, lookaround
Simplest possible behaviour (no $+, ?, \{\ \}, \hat{}, ..$)	Simplest possible use (Perl)

7 - RE support in languages

Built into the syntax of the language

- Perl, Ruby, Awk, Tcl, PHP, ...

Using libraries and functions

- Java, C, C++, ...

The specimen regexps are those from Perl 5.10.

8 - Regexp matching engines

- DFA (conversion of RE to DFA)
- NFA (conversion of RE to NFA, Thompson's method)
- Backtracking search (a must for non-regular languages). Be warned, the complexity of search can be exponential as opposed to FA!

9 - Conclusion - links and questions

- http://www.regular-expressions.info/quickstart.html
- http://www.gnu.org/software/sed/manual /html_node/Regular-Expressions.html
- http://www.grymoire.com/Unix/Regular.html
- http://www.grymoire.com/Unix/Sed.html#uh-0
- http://perldoc.perl.org/perlre.html
- http://regexpal.com/