Case Study on Regular Expression

Regular expressions, or simply regex, have been widely used as a powerful pattern matching and text extractor tool through decades. Although they provide a powerful and flexible notation to define and retrieve patterns from text, the syntax and the grammatical rules of these regex notations are not easy to use, and even to understand. Any regex can be represented as a Deterministic or Non-Deterministic Finite Automata; so it is possible to design a representation to automatically build a regex, and a optimization algorithm able to find the best regex in terms of complexity. This paper introduces both, a graph-based representation for regex, and a particular heuristic-based evolutionary computing algorithm based on grammatical features from this language in a particular data extraction problem.

regexpal.com and gskinner.com are good websites which provide the regex test services. You can try your regex on them. Especially, there are many examples on gskinner.com. They are good materials for the beginner to read and understand the regex patterns.

2.1 Email Validator

RegExp: ^[w-.+%]+@(?:[w-]+.)+[w]{2,6}$

This pattern can matches the emails like

- 521+me@company.com
- jeffrey.sun@example.net
- Chris.Liu@731.110.org

Description:

^[w-.+%]+ matches the user account which is before @. It matches a beginning string contains word character, slash sign, dot sign, cross sign and percent sign.

^ start of the line.

[begin of the character set

w any word character [A-Za-z0-9_].

– the slash sign.

. the dot sign.

+ the cross sign.

% it is not meta-character. It keeps its normal meaning.

] end of the character set. Any of element inside it is acceptable.

+ repeat the preceding element one or more times.

(?:[w-]+.)+ matches the sub-domain names between @ and the top domain name.

(?: begin of the group. But tell the engine not to capture it, since we don't reference it later.

[begin of the character set

w any word character [A-Za-z0-9_].

– the slash sign.

] end of the character set. Any of element inside it is acceptable.

+ repeat the preceding element one or more times.

. the dot sign.

) end of the group.

+ repeat the preceding element one or more times. It matches multiple subdomains, such as *731.110.*

[w]{2,6}$ matches top domain name. It is an ending string contains two to six word characters. $ is an anchor operator, which means the end of the string.

3 Meta-characters

Depends on the regex processor, there are about 14 meta-characters, which have special meaning. They are **{}[]()^$.|*+?**. Remembering them are very important to master the regex.

The meta-characters don't have their literal character meaning. If you want to use their literal meaning, you must add the backslash before them.

3.1 Character matching

Metacharacter	Description	Example
.	Match any single character.	b**.**g matches b**a**g or b**e**g.
[ ]	Match a single character that is contained with the character set.	b**[ai]**g matches b**a**g or b**i**g, but not b**e**g.
[^]	Match a single character that is NOT contained with the character set. This is	b**[^ai]**g matches b**e**g, but not

	opposite to the previous one. The meaning of ^ inside the brackets is different from that outside the brackets (anchor).	b**a**g or b**i**g.

( )	Groups a series of pattern elements to a single element. When you match a pattern within parentheses, you can use any of $1, $2, … later to refer to the previously matched pattern.	$string1 = “Hello Worldn”;if ($string1 =~ m/(H..).(o..)/) {
		
		print “We matched ‘$1’ and ‘$2’n”;
		
		}
		
		Output:
		
		We matched ‘Hel’ and ‘o W’

(?:expr)	Non-capturing group. You can’t use $ index to refer to the matched pattern. This group will be discarded by the regex engine after matching.	
|	Match either the element before or the element after this operator	**cat|dog** will match **cat** in **About cats and dogs**.
n, r and t	Match an LF character, CR character and a tab character respectively	_
d,w,s	A digit, word character [A-Za-z0-9_], or whitespace.	_
D,W,S	Anything except a digit, word character, or whitespace.	_
xFF where FF are 2 hexadecimal digits	Match the character with the specified ASCII/ANSI value, which depends on the code page used.	**xA9** matches the copyright symbol in the Latin-1 character set.
uFFFF where FFFF are 4 hexadecimal digits	Match a Unicode character. [3]	**u20AC** matches the euro currency sign

3.2 Quantification

This character indicates how often of the preceding element is allowed to occur. [2]

Metacharacter	**Description**	**Example**
?	Repeats the previous item ***zero or one*** times	**abc?** matches **ab** or **abc**
*****	Repeats the previous item ***zero or more*** times. Greedy, match as many items as possible before stopping.	**“.*”** matches **“def” “ghi”** in **abc “def” “ghi” jkl**
***?** (lazy star)	Repeats the previous item ***zero or more*** times. Lazy, match as few items as possible before stopping.	**“.*?”** matches **“def”** in **abc “def” “ghi” jkl**
+	Repeats the previous item ***one or more*** times. Greedy, match as many items as possible before stopping.	**“.+”** matches **“def” “ghi”** in **abc “def” “ghi” jkl**
+? (lazy plus)	Repeats the previous item ***one or more*** times. Lazy, match as few items as possible before stopping.	**“.+?”** matches **“def”** in **abc “def” “ghi” jkl**
{n} where n is an integer >= 1	Repeats the previous item exactly n times.	**a{3}** matches **aaa**
{m,n} where n >= 0 and m >= n	Repeats the previous item **between n and m** times.	**b[1-9][0-9]{2,4}b** matches a number between 100 and 99999
{n,m}? where n >= 0 and m >= n	Repeats the previous item **between n and m** times. Lazy, so repeating n times is tried before increasing the repetition to m times.	

3.3 Anchor

An anchor doesn’t match any characters. It matches a position.

Metacharacter	**Description**	**Example**
^	Match the start of the string	**^.** matches **a** in **a**bcndef.

$	Match the end of the string	**.$** matches **f** in abcnde**f**.
b	Matches at the position between a word character and a non-word character.	**.b** matches **c** in ab**c**

3.4 Look-around

Look-around is special kind of the group. The expression inside the group must be matched, but the matched result isn’t added to the final result of the whole regex.

Metacharacter	Description	Example
(?=expr)	Positive lookahead. The expr much be matched.	**q(?=u)** matches the **q** in **question**, but not in **Iraq**.The u is not part of the overall regex match. But it must exist.
(?!expr)	Negative lookahead.	**q(?!u)** matches **q** in **Iraq** but not in **question**