2 – Writing Regular Expressions

Week 14 – presentation 2

Regular expressions

A regular expression (RE) is a notation for specifying a set of strings (a formal language).

An RE is either
- The empty set
- The empty string
- A single character or wildcard symbol
- An RE enclosed in parentheses
- The *concatenation* of two or more REs
- The *union* of two or more REs
- The *closure* of an RE
  (any number of occurrences)

operation	*example RE*	*matches* (IN *the set*)	*does not match* (NOT *in the set*)
concatenation	aabaab	aabaab	*every other string*
wildcard	.u.u.u.	cumulus jugulum	succubus tumultuous
union	aa | baab	aa baab	*every other string*
closure	ab*a	aa abbba	ab ababa
parentheses	a(a|b)aab	aaaab abaab	*every other string*
parentheses	(ab)*a	a ababababa	aa abbba

More examples of regular expressions

The notation is surprisingly expressive.

regular expression	*matches*	*does not match*
`.*spb.*` *contains the trigraph* spb	raspberry crispbread	subspace subspecies
`a* | (a*ba*ba*ba*)*` *multiple of three* b*'s*	bbb aaa bbbaababbaa	b bb baabbbaa
`.*0....` *fifth to last digit is* 0	1000234 98701234	111111111 403982772
`.*gcg(cgg|agg)*ctg.*` *fragile X syndrome pattern*	...gcgctg... ...gcgcggctg... ...gcgcggaggctg...	gcgcgg cggcggcggctg gcgcaggctg

Generalized regular expressions

Additional operations further extend the utility of REs.

operation	*example RE*	*matches*	*does not match*
one or more	`a(bc)+de`	abcde abcbcde	ade bcde
character class	`[A-Za-z][a-z]*`	lowercase Capitalized	camelCase 4illegal
exactly j	`[0-9]{5}-[0-9]{4}`	08540-1321 19072-5541	111111111 166-54-1111
between j and k	`a.{2,4}b`	abcb abcbcb	ab aaaaaab
negation	`[^aeiou]{6}`	rhythm	decade
whitespace	`\s`	*any whitespace char (space, tab, newline...)*	*every other character*

Note. These operations are all *shorthand*.
They are very useful but not essential.

RE: `(a|b|c|d|e)(a|b|c|d|e)*`
shorthand: `(a-e)+`

Example of describing a pattern with a generalized RE

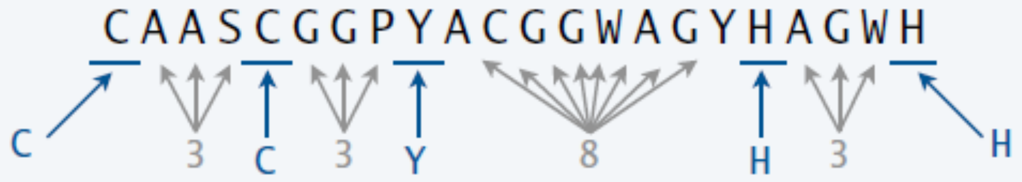
A C_2H_2-type zinc finger domain signature is

- C followed by 2, 3, or 4 amino acids, followed by
- C followed by 3 amino acids, followed by
- L, I, V, M, F, Y, W, C, or X followed by 8 amino acids, followed by
- H followed by 3, 4, or 5 amino acids, followed by

Q. Give a generalized RE for all such signatures.

A. `C.{2,4}C...[LIVMFYWCX].{8}H.{3,5}H`

"Wildcard" matches any of the letters
CAVLIMCRKHDENQSTYFWP



Constructs of the standard regular expression and meta characters

Let's get familiar with core constructs of regular expressions and some reserve meta characters that have a special meaning in regular expressions.

Symbol	Meaning	Example
. (dot or period)	Matches any character other than newline.	Matches **#**, **@**, **A**, **f**, **5**, or **.**
* (asterisk)	* matches zero or more occurrences of the preceding character or group.	`m*` matches 0 or more occurrences of the letter **m.**
+ (plus)	+ matches one or more occurrences of the preceding element.	`m+` matches one or more occurrences of the letter **m.**
? (question mark)	**?** means optional match. It is used to match zero or one occurrence of the preceding element. It is also used for lazy matching (which will be covered in the coming chapters).	`nm?` means match **n** or **nm**, as **m** is an **optional** match here.

Symbol	**Meaning**	**Example**
| (pipe)	| means alternation. It is used to match one of the elements separated by `|`	`m|n|p` means match either the letter `m` or the letter `n` or the letter `p`
^ (cap)	^ is called anchor, that matches start of the line	`^m` matches `m` only when it is the first character of the string that we are testing against the regular expression. Also, note that you do not use ^ in the middle of a regular expression.
$ (dollar)	**$** is called anchor that matches line end.	`m$` matches `m` only at line end.

Symbol	**Meaning**	**Example**
\b (backslash followed by the letter b)	Alphabets, numbers, and underscore are considered word characters. **\b** asserts word boundary, which is the position just before and after a word.	**`\bjava\b`** matches the word, `java` . So, it will not match `javascript` since the word, `javascript`, will fail to assert `\b` after `java` in the regex.
\B (backslash followed by uppercase B)	**\B** asserts true where **\b** doesn't, that is, between two word characters.	For the input text, *`abc`*, **`\B`** will be asserted at two places: 1. Between *a* and *b*. 2. Between *b* and *c*.
(...) a sub-pattern inside round parentheses	This is for grouping a part of text that can be used to capture a certain substring or for setting precedence.	**`m(ab)*t`** matches `m`, followed by zero or more occurrences of the substring, **ab**, followed by **t**.
{min,max}	A quantifier range to match the preceding element between the minimum and the maximum number.	**`mp{2,4}`** matches `m` followed **2** to **4** occurrences of the letter `p`.

Symbol	**Meaning**	**Example**
[...]	This is called a character class.	`[A-Z]` matches any uppercase English alphabet.
\d (backslash followed by the letter d)	This will match any digit.	`\d` matches any digit in the 0-9 range.
\D (backslash followed by uppercase D)	This matches any character that is not a digit.	`\D` matches `a`, `$`, or `_`.
\s (backslash followed by the letter s)	Matches any whitespace, including tab, space, or newline.	`\s` matches `[ \t\n]`.
\S (backslash followed by uppercase S)	Matches any non-whitespace.	`\s` matches the opposite of `\s`

Symbol	**Meaning**	**Example**
\w (backslash followed by the letter w)	Matches any word character that means all alphanumeric characters or underscore.	`\w` will match `[a-zA-Z0-9_]`, so it will match any of these strings: "*abc*", "*a123*", or "*pq_12_ABC*"
\W (backslash followed by the letter W)	Matches any non-word character, including whitespaces. In regex, any character that is not matched by \w can be matched using \W.	It will match any of these strings: "+/=", "*$*", or " *!~*"

Some basic regular expression examples

`ab*c`	This will match a, followed by zero or more b, followed by c.
`ab+c`	This will match a followed by one or more b, followed by c.
`ab?c`	This will match a followed by zero or one b, followed by c. Thus, it will match both abc or ac.
`^abc$`	This will match abc in a line, and the line must not have anything other than the string abc due to the use of the start and end anchors on either side of the regex.
`a(bc)*z`	This will match a, followed by zero or more occurrences of the string bc, followed by z. Thus, it will match the following strings: az, abcz, abcbcz, abcbcbcz, and so on.

```
a(bc)*z
```

This will match a, followed by zero or more occurrences of the string bc, followed by z. Thus, it will match the following strings: az, abcz, abcbcz, abcbcbcz, and so on.

```
ab{1,3}c
```

This will match a, followed by one to three occurrences of b, followed by c. Thus, it will match following strings: abc, abbc, and abbbc.

```
red|blue
```

This will match either the string red or the string blue.

```
\b(cat|dog)\b
```

This will match either the string cat or the string dog, ensuring both cat and dog must be complete words; thus, it will **fail** the match if the input is cats or dogs.

```
[0-9]
```

This is a character class with a character range. The preceding example will match a digit between 0 and 9.

```
[a-zA-Z0-9]
```

This is a character class with a character range. The preceding example will match any alpha-numeric character.

```
^\d+$
```

This regex will match an input containing only one or more digits.

```
^\d{4,8}$
```

This regex will allow an input containing four to eight digits only. For example, `1234`, `12345`, `123456`, and `12345678` are valid inputs.

```
^\d\D\d$
```

This regex will allow an input containing four to eight digits only. For example, `1234`, `12345`, `123456`, and `12345678` are valid inputs.

```
^\d\D\d$
```

This regex not only allows only one digit at the start and end but also enforces that between these two digits there must be one non-digit character. For example, `1-5`, `3:8`, `8X2`, and so on are valid inputs.

```
[a-zA-Z0-9]
```

This is a character class with a character range. The preceding example will match any alpha-numeric character.

```
^\d+$
```

This regex will match an input containing only one or more digits.

```
^\d{4,8}$
```

This regex will allow an input containing four to eight digits only. For example, `1234`, `12345`, `123456`, and `12345678` are valid inputs.

```
^\d\D\d$
```

This regex not only allows only one digit at the start and end but also enforces that between these two digits there must be one non-digit character. For example, `1-5`, `3:8`, `8X2`, and so on are valid inputs.

```
^\d+\.\d+$
```

This regex matches a floating point number. For example, `1.23`, `1548.567`, and `7876554.344` are valid inputs.

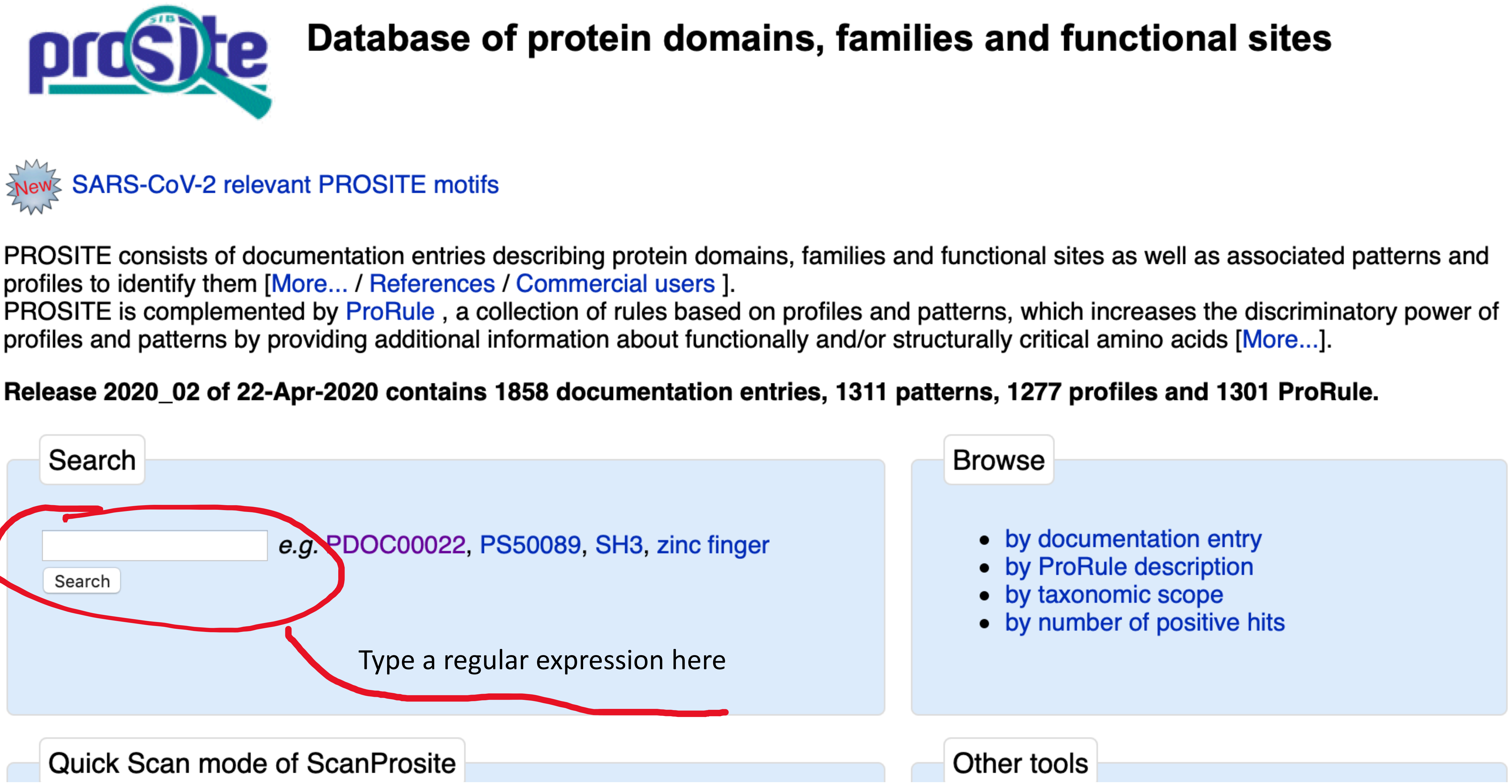
```
.+
```

This matches any character one or more times. For example, `qwqewe`, `12233`, or `f5^h_=!bg` are all valid inputs:

```
^\w+\s+\w+$
```

This matches a word, followed by one or more whitespaces, followed by another word in an input. For example, `hello word`, `John Smith`, and `United Kingdom` will be matched using this regex.

Another real world application



Database of protein domains, families and functional sites

New SARS-CoV-2 relevant PROSITE motifs

PROSITE consists of documentation entries describing protein domains, families and functional sites as well as associated patterns and profiles to identify them [More... / References / Commercial users].
PROSITE is complemented by ProRule , a collection of rules based on profiles and patterns, which increases the discriminatory power of profiles and patterns by providing additional information about functionally and/or structurally critical amino acids [More...].

Release 2020_02 of 22-Apr-2020 contains 1858 documentation entries, 1311 patterns, 1277 profiles and 1301 ProRule.

Search

e.g. PDOC00022, PS50089, SH3, zinc finger

Search

Type a regular expression here

Browse

- by documentation entry
- by ProRule description
- by taxonomic scope
- by number of positive hits

Quick Scan mode of ScanProsite

Other tools