Contents

1 Syntax of Regex Expression

1.1 Intro

```
\regex_replace_all:nnN { \w+ } { \c{emph}\cB\{ \0 \cE\} , } \l_my_tl
% 1. \0: denotes the full match
% 2. \c{emph}: \emph
% 3. \cB\{...\cE\}: {...}
% 4. \cB.: \begingroup
% 5. \cE.: \endgroup
```

1.2 new and store

If a regular expression is to be used several times, it can be compiled once, and stored in a regex variable using `\regex_set:Nn`. For example

```
\regex_new:N \l_foo_regex
\regex_set:Nn \l_foo_regex { \c{begin} \cB. (\c[^BE].*) \cE. }
```

1.3 regex show

`\regex_show:n { \c{begin} \cB. (\c[^BE].*) \cE. }` will show the details of the regex.

```
> Compiled regex {\c {begin}\cB .(\c [^BE].*)\cE .}:
+-branch
  control sequence \begin
  Match
    categories B, class
      any token
  ,-group begin
  | Match, repeated 0 or more times, greedy
  |   categories CMTPUDSLOA, class
  |     any token
  `-group end
  Match
    categories E, class
      any token.
<recently read> }
```

1.4 match rule

- every alphanumeric character (`A{Z, a{z, 0{9`) matches exactly itself
- non-alphanumeric printable ascii characters can (and should) always be escaped
- spaces should always be escaped (even in character classes)
- any other character may be escaped or not, without any effect: both versions match exactly that character.

1.5 Characters classes

Some predefined character classes are available:

- `.`: A single period matches any token.
- `\d`: decimal digit
- `\h`: space or tab
- `\w`: any word, alphanumerics and underscore
- The uppercae form of the above classes are negated, e.g. `\D`, `\H`, `\W`

  match exactly one token regex expression:

- `[...]`: matches any of the special tokens
- `^...`: matches any token except the special tokens
- `[x-y]`: Within a character class
- `[:<name>:]`: the posix character class
- `[^:<name>:]`: the negated form of posix character class

1.6 alternatives, groups, repetitions

For **greedy quantifiers** the regex code will first investigate matches that involve as **many** repetitions as possible, while for **lazy quantifiers** it investigates matches with as **few** repetitions as possible first.

- `?`: 0 or 1, greedy
- `??`: 0 or 1, lazy
- `*`: 0 or more, greedy.
- `*?`: 0 or more, lazy
- `+`: 1 or more, greedy
- ...
- `{n}`: exact n times
- `{n,}`: at least n times
- ...
- `{n, m}`: At least n, no more than m, greedy.
- ...

1.7 Alternation and capturing groups

- `A|B|C`: Either one of A, B, or C, investigating A first.
- `(...)`: Capturing group.
- `(?:...)`: Non-capturing group.

1.8 Matching exact tokens

- `C`: for control sequences;
- `B`: for begin-group tokens;
- `E`: for end-group tokens;
- `T`: for alignment tab tokens;
- `P`: for macro parameter tokens;
- `U`: for superscript tokens (up);
- `D`: for subscript tokens (down);
- `S`: for spaces;
- `L`: for letters;

Use as follows:

```
1. \c{<regex>} A control sequence whose csname matches the ⟨regex⟩

2. \c[XYZ] Applies to the next object, and forces it to only match tokens with category X, Y, or
```

2 Miscellaneous and replacement text

`\b`: Word boundary, replacement text refered as.

- `\0`: is the whole match
- `\1`: is the submatch that was matched by the first (capturing) group (...);
- `\a, \e, \f, \n, \r, \t, \xhh, \x{hhh}`: correspond to single characters as in regular expressions;
- `\c{<cs name>}`: inserts a control sequence;
- `\u{<tl var name>}`: inserts the contents of the `tl var`

Most of the features described in regular expressions do not make sense within the replacement text.

2.1 match and further item

Finds the first match of the ⟨regular expression⟩ in the ⟨token list⟩. If it exists, the **match** is stored as the **first item** of the ⟨seq var⟩, and **further items** are the contents of **capturing groups**, in the order of their opening parenthesis.

An explicit example:

```
\regex_extract_once:nnN {\((.*?)\)}{1-2(30~ 6)=[]}\l_tmpa_seq
\seq_show:N \l_tmpa_seq
\seq_use:Nn \l_tmpa_seq {,~}
```

```
>  {(30 6)}
>  {30 6}.
<recently read> }
l.171 \seq_show:N \l_tmpa_seq
```

3 Practice

2083