regex-tdfa Cheat Sheet

by williamyaoh via cheatography.com/78742/cs/19172/

Importing and Using

```
dependencies:
  - regex-tdfa
  - regex-tdfa-text
import Text.Regex.TDFA
import Text.Regex.TDFA.Text ()
```

Basic Usage

```
<to-match-against> =~ <regex> -- non-monadic, gives some reasonable 'default' if no match
<to-match-against> =~~ <regex> -- monadic, calls `fail' on no match
"my email is email@email.com" =~ "[a-zA-Z0-9_\\-]+@[a-zA-Z0-9-\\-]+\\.[a-zA-Z0-9]+"
```

`(=~)` and `(=~~)` are polymorphic in their return type, so you may need to specify the type explicitly if GHC can't infer it. This is a little inconvenient sometimes, but allows the matching operators to be used in a lot of different situations. For example, it can return a `Bool`, if all you need is to check whether the regex matched; it can return a list of the matched strings; or it can return a list of the match indices and lengths, depending on what you need.

Basic Usage

```
a =~ b -- a and b can both be any of String, Text, or ByteString
"foo-bar" =~ "[a-z]+" :: String -- or Text, ByteString...
>>> "foo"
```

`regex-tdfa` only supports String and ByteString by default; `regex-tdfa-text` provides the instances for Text.

Common use cases

```
a =~ b :: Bool -- did it match at all?
a =~ b :: (String, String, String)
  -- the text before the match, the match itself, and the text after the match
a =~ b :: (String, String, String, [String])
  -- same as above, plus a list of only submatches
```

Advanced usage

```
getAllTextMatches (a =~ b) :: [String]
getAllMatches (a =~ b) :: [(Int, Int)]
getAllTextSubmatches (a =~ b) :: [String]
  -- the first element of this list will be the match of the whole regex
getAllSubmatches (a =~ b) :: [(Int, Int)]
```

For these functions, we can also request an Array as the return value instead of a List (again, through polymorphism).