Beschreibung regulärer Sprachen mit regulären Ausdrücken

Reguläre Sprachen können außer durch eine Grammatik auch durch sogenannten **reguläre Ausdrücke** beschrieben werden.

Grundlegende reguläre Operatoren

Reguläre Ausdrücke formuliert man mit regulären Operatoren. Die wichtigsten regulären Operatoren sind die folgenden:

- a* - "a-Stern": a beliebig oft (auch 0 mal) hintereinander
- ab - Verkettung: ein a, dann ein b
- a|b - Oder: entweder ein a oder ein b
- Mit runden Klammern ( ) können Teilausdrücke gruppiert werden

Beispiele

(1) α_1 = c*h* beschreibt alle Zeichenketten, die zunächst aus beliebig vielen Buchstaben c bestehen und anschließend aus beliebig vielen Buchstaben h. [1)]. L_1={ε,c,h,ccch,ccchhh,hhhh,cc,...}.

(2) α_2 = c*|h* beschreibt die alle Zeichenketten, die entweder nur aus c's oder nur aus h's bestehen. L_2={ε,c,h,cc,hh,ccc,hhh,...}.

(3) α_3 = (c|h)* beschreibt die alle Zeichenketten, die aus beliebig vielen c's und h's bestehen. L_3={ε,c,h,hc,ch,ccc,hchh,...}.

(A1)

Welche Sprachen werden durch die folgenden regulären Ausdrücke beschrieben? Liste eine Beispielmenge von akzeptierten Wörtern auf, aus der hervorgeht, welche Regeln gelten.

(A) α_4 = 01*0 L= {00, 010, 0110, 01110, 011110, 0111110, ...}

(B) α_5 = (01)* L= {ε, 01, 0101, 010101, 01010101, 0101010101,... }

(C) α_6 = 0|(10*)* L= {ε, 0, 1, 10, 100, 1000, 11, 101, 10100, 100100, ... }

(D) α_7 = (1(110)*)* L= {ε, 1, 1110, 1110110, 11, 11101110, ... }

(E) α_8 = `1(1*|10*)` L= {1, 11, 111, 1111, 11111, 110, 1100, 11000, ... }

(F) α_9 = `0|1*|10*` L= {ε, 0, 1, 11, 111, 1111, 10, 100, 1000, 10000, ... }

(G) α_{10} = `1*0*(0|1)` L= { 0, 1, 10, 11, 110, 111, 11001, 1110001, ... }

Erweiterte reguläre Ausdrücke

Mit der "Minimalausstattung" von drei Operatoren (ohne die Klammern) lassen sich im Prinzip schon alle regulären Ausdrücke schreiben. Bequemer, kürzer und übersichtlicher geht es, wenn man weitere Operatoren hinzunimmt, hier ist allerdings zu beachten, dass es unterschiedliche "Dialekte" solcher Erweiterungen gibt.

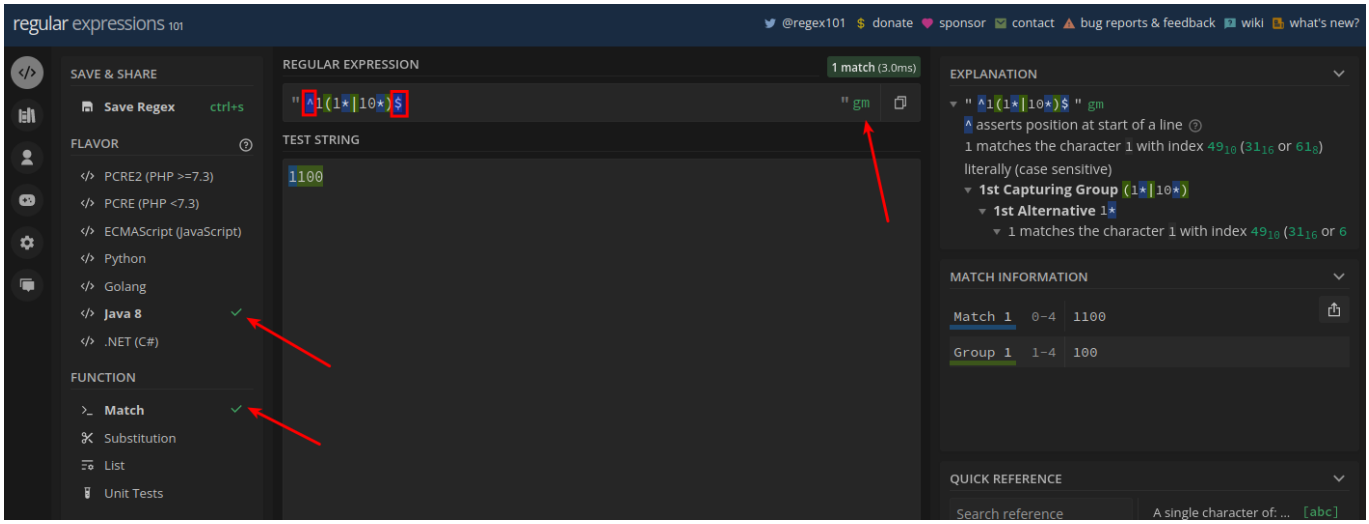
Die meisten Programmiersprachen verfügen über Funktionen oder Bibliotheken, um Strings anhand von regulären Ausdrücken zu finden. Meist gibt es auch weitere Funktionen wie Rückbezüge, mit denen sich komplexe Ersetzungen und Ähnliches realisieren lassen.

Typische Erweiterungen umfassen die folgenden Elemente:

- Eckige Klammern fassen Zeichengruppen zusammen. Der Ausdruck `[0-9]` bedeutet "die Ziffern von 0 bis 9", `[0-9]*` bedeutet entsprechend "beliebig oft eines der Zeichen von 0-9. Bislang hätten wir dafür `(0|1|2|3|4|5|6|7|8|9)*` schreiben müssen.
- `[abc]` – der Buchstabe a, b oder c (anstatt `(a|b|c)`).
- `[a-zA-Z]` – die Buchstaben A bis Z, groß und klein.
- `.`[2] – ein beliebiges Zeichen.
- `[^abc]` – jedes Zeichen außer den Buchstaben a, b und c [3]
- Wenn man ein Zeichen suchen will, das eigentlich eine Spezialbedeutung hat, muss man es escapen, meist mit einem Backslash: `[\[a-z\]]` - erst ein [, dann ein Buchstabe a bis z dann ein].
- Der Anfang einer Zeile kann durch durch ein ^ gematcht werden, das Ende durch ein $.
- Ein * bedeutet wie bisher, dass beliebig viele Zeichen der vorherigen Gruppe gesucht werden `^[aeiou]*n$` – findet n, an, en, in, on, un, aan, aaan, aen, ein, ... sofern sie in einer Zeile stehen.
- `a{2,5}` - 2 bis 5 Vorkommen des Buchstabens a [4]
- `b+` - mindestens ein Vorkommen des Buchstabens b [5].

Sandkasten bei regex101.com

Auf der Seite https://regex101.com/ kannst du reguläre Ausdrücke testen. Außerdem werden die Ausdrücke und Matches erklärt.



(A2)

Öffne die Seite https://regex101.com/ und sorge dafür, dass die Einstellungen wie im Screenshot zusehen angepasst sind[6)]

Überprüfe jetzt deine Ergebnisse aus Aufgabe 1, indem du den regulären Ausdruck in die Eingabezeile "regular Expression" schreibst und die zu testenden Wörter der regulären Sprache in das Feld "Test String", pro Zeile ein Wort.

> Achtung: Weil die regulären Ausdrücke nach der Greedy Strategie angewandt werden, musst du deine regulärer Ausdruck inklusive Zeilenanfang (^) und Zeilenende ($) testen, sonst klappts nicht - siehe Screenshot. Probiere einfach aus, was passiert, wenn du diese beiden Zeichen weglässt.

(A3) Gültige Telefonnummern

Es soll erkannt werden, ob ein String das korrekte Format für eine Telefonnummer hat.

Für uns soll eine korrekte Telefonnummer zunächst folgende Kriterien aus folgenden Bestandteilen zusammengesetzt sein:

- Einer Vorwahl, die mit einer 0 beginnt und danach mindestens eine andere Ziffer enthält.
- Einem der folgenden Zeichen zur Abtrennung der Vorwahl: -, / oder Leerzeichen
- Einer mindestens einstelligen Zahl, die nicht mit 0 beginnt.

Erstelle einen regulären Ausdruck, der in der folgenden Liste von Ziffernfolgen alle korrekten Telefonnummern erkennt.

Liste

```
07071-12345
17071-12345
07071 12345
07071 02345
0721  1654
171-8867524
07473/138652
0049-12345
```

[1)]

"Erst beliebig viele c's, dann beliebig viele h's" - wobei "beliebig viele auch "keines" bedeuten kann.

[2)]

Punkt

[3)]

Hier bedeutet das ^ gewissermaßen "nicht"

[4)]

Bisher: `(aa|aaa|aaaa|aaaaa)`

[5)]

Unterschied zu *: Beliebig oft, aber **nicht** kein mal

[6)]

Regex Flavor `Java 8`, Funktion `Match`, regex Optionen `gm`