Blatt 01: Reguläre Sprachen

A1.1 Sprache von `a + a(a + b)*a`

$$L = \{a\} \cup \{awa \mid w \in \{a, b\}^*\}$$

Es beschreibt alle Wörter, die entweder nur aus einem einzigen `a` bestehen oder mit `a` beginnen und auch enden. Dazwischen könnte auch beliebige Kombinationen aus `a` und `b` stehen.

Beispiele: `a`, `aa`, `aba`, `abba`

Nicht enthalten: `b`, `ba`, `abb`

A1.2 Bezeichner in Programmiersprachen

A) Regex:

```
(V|v|P|p|[A-Za-z])[A-Za-z0-9_]*[A-Za-z0-9]
```

- Erstes Zeichen: V, v, P, p (ist eigentlich auch in a-z enthalten) oder ein Buchstabe (a–z, A–Z)
- Danach: Buchstaben, Ziffern oder Unterstriche erlaubt
- Letztes Zeichen: kein Unterstrich
- Länge: mindestens 2 Zeichen

Beispiele: `va1`, `Vx_2`, `Pa`, `name1`

Nicht erlaubt: `v_`, `_a`, `v`

b) DFA

Alphabet: $\Sigma = \{a \dots z, A \dots Z, 0 \dots 9, \_\}$. Zustände: $Q = \{q_0, q_1, q_A, q_U, q_{\text{dead}}\}$. Start: q_0. Endzustände: $F = \{q_A\}$.

Intuition q_1: genau ein Anfangsbuchstabe. q_A: gültiges Ende (mind 2 Zeichen, letztes Zeichen Buchstabe/Ziffer). q_U: letztes Zeichen unterstrich (kein gültiges Ende). q_{dead}: Fehler.

Übergangstabelle

	Letter	Digit	_	sonst
q_0	q_1	q_{dead}	q_{dead}	q_{dead}
q_1	q_A	q_A	q_U	q_{dead}
q_A	q_A	q_A	q_U	q_{dead}
q_U	q_A	q_A	q_U	q_{dead}
q_{dead}	q_{dead}	q_{dead}	q_{dead}	q_{dead}

Beispielläufe

$$\texttt{vCount1}: q_0 \xrightarrow{v} q_1 \xrightarrow{C} q_A \xrightarrow{o} q_A \xrightarrow{u} q_A \xrightarrow{n} q_A \xrightarrow{t} q_A \xrightarrow{1} q_A \text{ (akzeptiert)}$$
$$\texttt{P\_}: q_0 \xrightarrow{P} q_1 \not\to q_U \text{ (nicht akzeptiert)}$$

B) Reguläre Grammatik:

$$\begin{aligned}
&S \to VA \mid vA \mid PA \mid pA \mid LA \\
&A \to XA \mid Y \\
&X \to a \mid b \mid ... \mid z \mid A \mid B \mid ... \mid Z \mid 0 \mid 1 \mid ... \mid 9 \mid \\
&Y \to a \mid b \mid ... \mid z \mid A \mid B \mid ... \mid Z \mid 0 \mid 1 \mid ... \mid 9
\end{aligned}$$

Ableitung für `va1`:

$$S \Rightarrow vA \Rightarrow vXA \Rightarrow vaA \Rightarrow vaY \Rightarrow va1$$

A1.3 Gleitkommazahlen in Python und Java

Regex

```
Python: ^[0-9]+(\.[0-9]+)?(e[+-]?[0-9]+)?$
Java:   ^[0-9]+(\.[0-9]+)?([eE][+-]?[0-9]+)?$
```

DFA

```
z0 --digit--> z1
z1 --digit--> z1
z1 --.--> z2
z2 --digit--> z3
z3 --digit--> z3
z1/z3 --e|E--> z4
z4 --+|--> z5
z4/z5 --digit--> z6
z6 --digit--> z6
Endzustände: z1 (Ganzzahl), z3 (mit Nachkommastellen), z6 (mit Exponent)
```

Reguläre Grammatik Terminals: Ziffern 0..9, Punkt ., e, E, +, -. Nichtterminale: S, I, F, X, Y, Z.

$$\begin{aligned}
&S \to d\,I \quad (d \text{ steht für eine Ziffer}) \\
&I \to d\,I \mid .\,F \mid e\,X \mid E\,X \mid \epsilon \\
&F \to d\,F \mid e\,X \mid E\,X \mid \epsilon \\
&X \to +\,Y \mid -\,Y \mid Y \\
&Y \to d\,Z \\
&Z \to d\,Z \mid \epsilon
\end{aligned}$$

bestatigung `3.14`, `2e10`, `1.0E-5` werden akzeptiert.

A1.4 Mailadressen

Gegebener Regex: $\texttt{(a-z)}^{+}@(a-z).(a-z)$

Problem:

- `.` steht für beliebiges Zeichen, nicht für Punkt.
- Vor und nach `@` nur je ein Buchstabe, keine Wiederholung, keine Subdomains.
- Zeichenvorrat zu klein: nur a bis z, keine Ziffern und erlaubten Sonderzeichen.
- Keine Anker `^` und `$`.

Warum `a + b + ...+ z` nicht reicht

- Beschreibt nur genau ein Zeichen; wiederholung fehlt.
- Ziffern und erlaubte Sonderzeichen fehlt.

Verbesserte Regex

```
^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$
```

Beispiele: hallo@mail.de, moin.du@uni.edu

A1.5 Der zweitletzte Buchstabe

$$\Sigma = \{1, 2, 3\},\ \texttt{Start}\ q_0,\ Q = \{q_0, r_1, r_2, r_3\} \cup \{s_{a,b,f}\},\ F = \{s_{a,b,1}\}.$$

Kurz: r_x nach erstem Zeichen x. In $s_{a,b,f}$ ist a das zweite Zeichen, b das letzte, $f = 1$ falls zweitletztes $= a$.

Beispiele Gültig: 1221 (zweites = 2, zweitletztes = 2).
Ungültig: 1213 (zweites = 2, zweitletztes = 1).

A1.6 Sprache einer regulären Grammatik

Gegeben:

$$S \to aA, \quad A \to dB \ \mid\ bA \ \mid\ cA, \quad B \to aC \ \mid\ bC \ \mid\ cA, \quad C \to \epsilon$$

Sprache:

- Start immer mit a (wegen $S \to aA$).
- In A sind nur b oder c erlaubt (aber beliebig viele), ein d führt nach B daher mindestens ein d.
- In B endet a oder b sofort, c springt zurück nach A dadurch Blöcke c(b|c)*d möglich.
- Ende ist a oder b, nie c.

also:

```
a(b|c)*d(c(b|c)*d)*(a|b)
```

Beispile: adb, acbcdb sind drin. ab, adc sind nicht drin.