Compilerbau Blatt 01:

A1.1

a + a (a+b)* a

Es wird eine Sprache beschrieben, bei der das erste und letzte Zeichen immer ein ‚a' ist. Die Zeichen zwischen den ‚a's sind dann ein oder mehrere ‚a's oder ‚b's

Mögliche ‚Wörter' sind z.B.:

- L(aa, aaba, aaaba, ababa, aaabaaba, ...)

A1.2

Regex: [a-zA-Z]([a-zA-Z_0-9])*[a-zA-Z0-9]

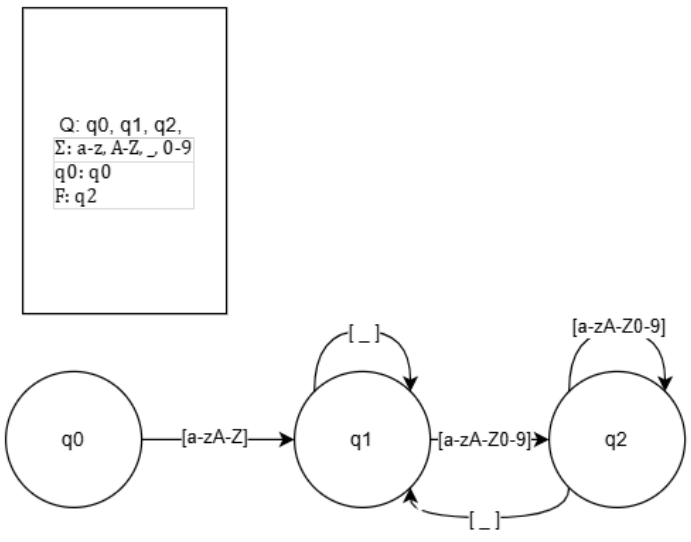
Beispiel1: Test_

1. T: -> [a-zA-Z] ist gültig
2. e: -> [a-zA-Z_0-9]* ist gültig
3. s: -> [a-zA-Z_0-9]* ist gültig
4. t: -> [a-zA-Z_0-9] ist gültig
5. _: -> [a-zA-Z0-9] ist nicht gültig

Beispiel2: Test_2

1. T -> [a-zA-Z] ist gültig
2. e: -> [a-zA-Z_0-9]* ist gültig
3. s: -> [a-zA-Z_0-9]* ist gültig
4. t: -> [a-zA-Z_0-9]* ist gültig
5. _: -> [a-zA-Z_0-9]* ist gültig
6. 2: -> [a-zA-Z0-9] ist gültig

DFA:



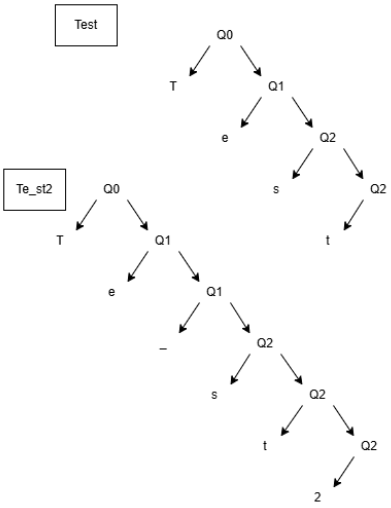
DFA Beispiel:

- Beispiel1: Test
    1. q0 -- T --> q1
    2. q1 -- e --> q2
    3. q2 -- s --> q2
    4. q2 -- t --> q2
- Beispiel2: T_s1t
    1. q0 -- T --> q1
    2. q1 -- _ --> q1
    3. q1 -- s --> q2
    4. q2 -- 1 --> q2
    5. q2 -- t --> q2

Grammatik:

- Q0 -> [a-zA-z]Q1
- Q1 -> [a-zA-Z0-9]Q2 | _Q1
- Q2 -> [a-zA-Z0-9]Q2 | _Q1 | e

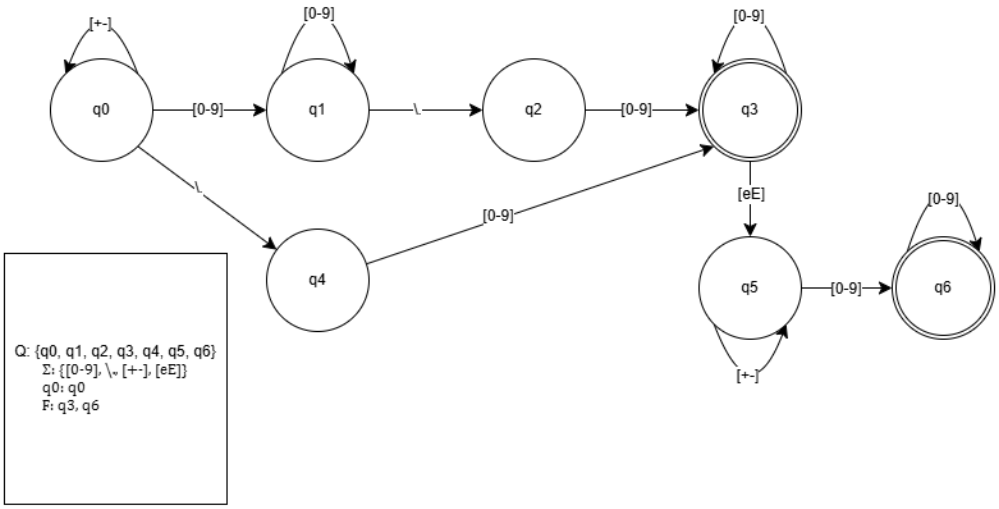
Ableitungsbäume:



A1.3

Python:

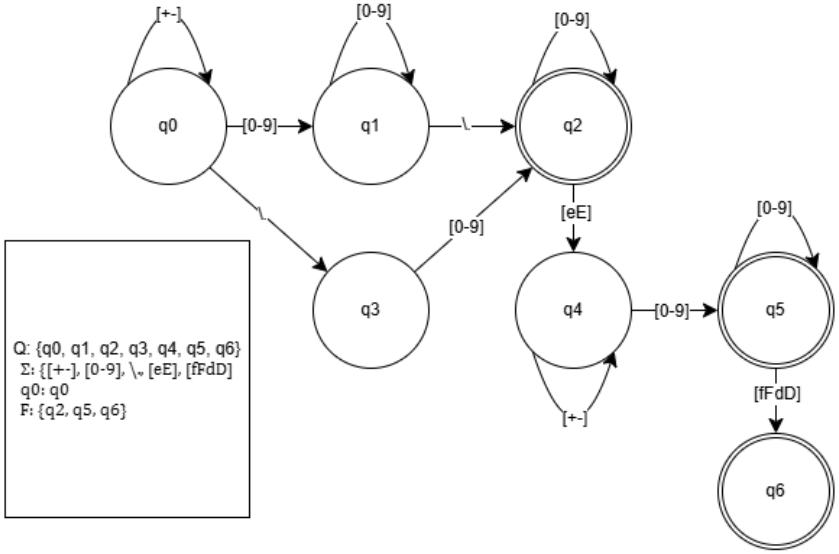
- Regex: [+-]?(([0-9]+(\.[0-9]+)?)|(\.[0-9]+))([eE][+-]?[0-9]+)?
- DFA:



- Grammatik:
    1. Q0 -> [+-]Q0 | [0-9]Q1 | \.Q4
    2. Q1 -> [0-9]Q1 | \.Q2
    3. Q2 -> [0-9]Q3
    4. Q3 -> [0-9]Q3 | [eE]Q5 | e
    5. Q4 -> [0-9]Q3
    6. Q5 -> [+-]Q5 | [0-9]Q6
    7. Q6 -> [0-9]Q6 | e

Java:

- Regex: [+-]?([0-9]+(\.[0-9]*)?|\.[0-9]+)([eE][+-]?[0-9]+)?[fFdD]?
- DFA:



-

- Grammatik:
    1. Q0 -> [+-]Q0 | [0-9]Q1 | \.Q3
    2. Q1 -> [0-9]Q1 | \.Q2
    3. Q2 -> [0-9]Q2 | [eE]Q4 | e
    4. Q3 -> [0-9]Q2
    5. Q4 -> [+-]Q4 | [0-9]Q5
    6. Q5 -> [0-9]Q5 | [fFdD]Q6 | e
    7. Q6 -> e

A1.4

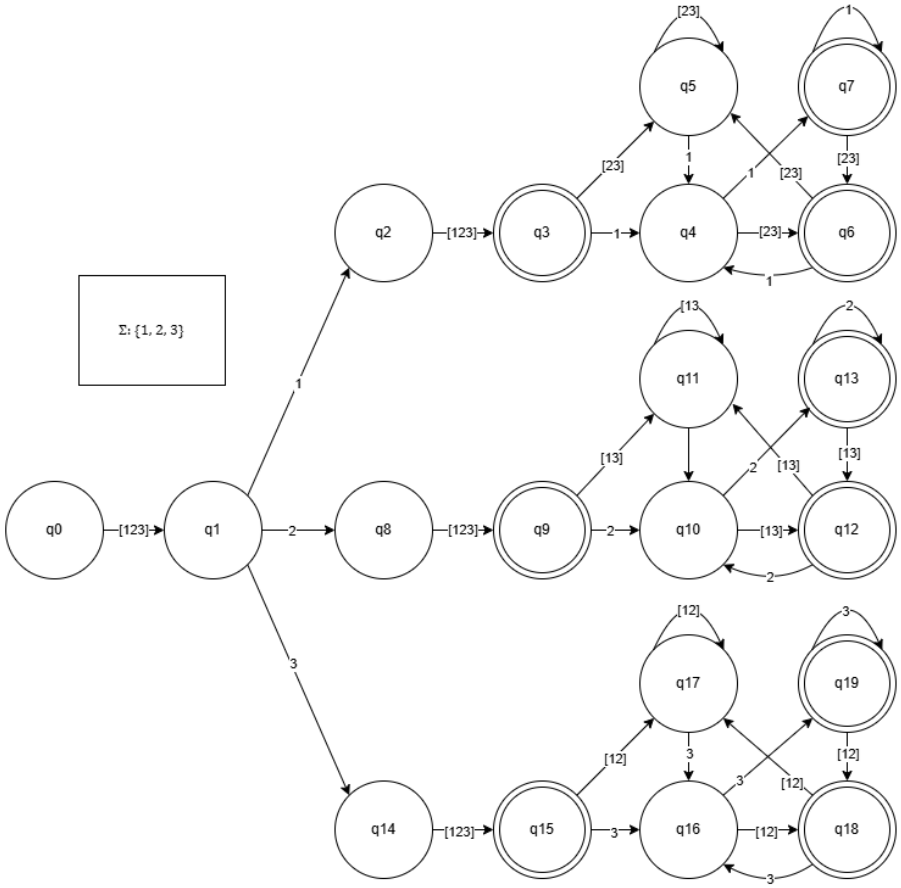
Regex: (a-z)+ @ (a-z).(a-z)

Der reguläre Ausdruck ist ungeeignet für E-Mail-Adressen da, zum Beispiel am Ende nur ein Zeichen stehen kann und deutsche Adressen die mit ‚.de' Enden könnten damit nicht abgebildet werden. Dazu könnten bei den Zeichen vor dem ‚@' auch Zahlen oder andere Zeichen wie z.B.: ‚-‘ oder ‚.' stehen und nicht nur Buchstaben.

Verbesserter regex: [a-z0-9!#\$&'*\+\-/=\^_{\|}~]([a-z0-9!#\$&'*\+\-/=\^_{\|}~]*|\.+[a-z0-9!#\$&'*\+\-/=\^_{\|}~]+ @ [a-z0-9][a-z0-9\-]*(\.[a-z0-9]+|\-[a-z0-9]+)+

A1.5

DFA:



A1.6

Regex: a[bc]*d(c[bc]*d)*[ab]

Reguläre Sprache: es wird eine Sprache beschrieben, bei der am Anfang jedes Wortes ein ‚a' steht, dann eine beliebige Anzahl an ‚b' oder ‚c', dann ein ‚d' und dann optional ein ‚c' gefolgt von wieder einer beliebigen Anzahl an ‚b' oder ‚c' und wieder ein ‚d' und zum Schluss ein ‚a' oder ‚b' wo dann das Wort terminiert.

Mögliche Wörter: L(adab, abcdab, abcbcdab, abcdcdab, adcbcbcdcbcdab, ...)

DFA: