Komplexübung: Grammatiken/Automaten und Realisierung mit Python

Einfache Maschinensprachen (*z.B. zur Programmierung von Werkzeugmaschinen*), kann man auf zweierlei Weise erzeugen.

Variante 1: Alle Maschinenbefehle haben die gleiche Zeichenlänge. Der Zeichenvorrat ist eindeutig definiert (und meistens sehr klein).

Beispiel: 4-Bit-Binärcode

Variante 2: Die Maschinenbefehle sind mindestens 1 Zeichen und höchstens n Zeichen lang. Der Zeichenvorrat ist eindeutig definiert. Zwischen zwei Befehlen folgt ein fest definiertes Trennzeichen, welches nicht im Zeichenvorrat der Befehle liegt.

Beispiel: Alle Befehle bestehen aus den Zeichen a, b und c und sind höchstens 3 Zeichen lang. Trennzeichen zwischen 2 Befehlen ist das Zeichen #.

Bemerkung: Java orientiert sich z.B. an Variante 2 - das Trennzeichen ist hier das Semikolon.

Aufgaben:

(1) Gib je eine vollständige Grammatik für die Beispielsprachen von Variante 1 und Variante 2 an! Schreibe die Produktionsregeln sowohl als Syntaxdiagramm, in Erweiterter Backus-Naur-Form (EBNF) und in Backus-Naur-Form (BNF) auf!

(2) Erstelle ausgehend von der BNF von Variante 2 je einen Ableitungsbaum für das Programm:

(a) ab#cc#a##bca
(b) c#c#abc

und entscheide anhand des Baumes auf Richtigkeit!

(3) Setze die Grammatik für Variante 2 mit dem bekannten Grammatikeditor um und teste ausgiebig!

(4) Erzeuge mit dem bekannten Automateneditor einen der Grammatik aus Aufgabe (3) äquivalenten Deterministischen endlichen Automaten (DEA) und teste diesen hinreichend!

(5) Schreibe in Python auf der Grundlage der im Anhang dargestellten Programmstruktur eine Umsetzung des DEA aus Aufgabe (4) und teste ausgiebig!

Beschäftige dich dabei wiederholend anhand des Programms mit den Begriffen: Zerlegungsprinzip, prozedurales Programmieren, Funktionen, Parameter, Rückgaben, globale und lokale Variablen, Casting, fehlerfreie Eingaben, Kontrollstrukturen.

(6) Konstruiere einen einfacheren Nichtdeterministischen endlichen Automaten (NEA) mit gleicher Erkennungsfunktion und teste diesen im Automateneditor.!

(7) Gib für den NEA aus Aufgabe (6) die vollständige Definition an. Die Überführungsfunktion soll dabei als Tabelle angegeben werden!

Anhang: *Mögliche Vorlage zu Aufgabe (5)*

```
# Funktionen

def Eingabe() :
    text=input("Gib einen Satz ein!  > ")
    return text

def Delta(zustand,zeichen) :
    pass


def Auswertung(text,zustand) :
    pass

# Hauptprogramm

text=Eingabe()
zustand="Start"
for zeichen in text :
    zustand=Delta(zustand,zeichen)
Auswertung(text,zustand)
```