

# Theoretische Informatik 1

## Übungsblatt 6

Thomas Haas  
Prof. Dr. Roland Meyer

TU Braunschweig  
Wintersemester 2019/20

Ausgabe: 21.01.2020

Abgabe: 30.01.2020, 15:00

Geben Sie Ihre Lösungen bis Donnerstag, 30.01.2020, 15:00 Uhr, durch Einwerfen in die Übungskästen neben Büro IZ 343 ab. Geben Sie in Gruppen von 4 Personen ab.

Es handelt sich hierbei um das letzte Übungsblatt, das abgegeben werden muss und bepunktet wird. Der weitere Stoff der Vorlesung ist dennoch klausurrelevant.

### Aufgabe 1: Abschlusseigenschaften [9 Punkte]

a) [4 Punkte] Es ist bekannt, dass die Sprache  $\mathcal{L} = \{a^n b^n c^n \mid n \in \mathbb{N}\}$  nicht kontextfrei ist. Zeigen Sie nun, dass aber das Komplement  $\overline{\mathcal{L}}$  von  $\mathcal{L}$  kontextfrei ist. Damit ist  $\overline{\mathcal{L}}$  ein Beispiel für eine kontextfreie Sprache deren Komplement nicht kontextfrei ist.

**Hinweis:** Es gibt mehrere Gründe, warum ein Wort  $w$  nicht in  $\mathcal{L}$  enthalten sein könnte. Da kontextfreie Sprachen unter Vereinigung abgeschlossen sind, reicht es also jeden dieser Gründe einzeln zu betrachten und als kontextfrei zu erkennen.

b) Zu einem Wort  $w = w_1 w_2 \dots w_{n-1} w_n$  definieren wir  $\text{reverse}(w) = w_n w_{n-1} \dots w_2 w_1$ . Zu einer Sprache  $\mathcal{L}$  sei  $\text{reverse}(\mathcal{L}) = \{\text{reverse}(w) \mid w \in \mathcal{L}\}$ .

- [2 Punkte] Zeigen Sie, wie man aus einem NFA  $A$  einen NFA  $A'$  mit  $\mathcal{L}(A') = \text{reverse}(\mathcal{L}(A))$  konstruieren kann.
- [2 Punkte] Zeigen Sie, wie man aus einer kontextfreien Grammatik  $G$  eine kontextfreie Grammatik  $G'$  mit  $\mathcal{L}(G') = \text{reverse}(\mathcal{L}(G))$  konstruieren kann.
- [1 Punkt] Folgern Sie aus den beiden obigen Punkten und der Tatsache, dass rechtslineare Grammatiken reguläre Sprachen erzeugen (siehe letztes Hausaufgabenblatt), dass linkslineare Grammatiken ebenso reguläre Sprachen erzeugen.

### Aufgabe 2: Pushdown-Automaten [6 Punkte]

Konstruieren Sie Pushdown-Automaten für folgende Sprachen und geben Sie jeweils an, welche Akzeptanzbedingung Sie annehmen (leerer Stack oder Finalzustände):

1. [2 Punkte]  $\mathcal{L}_1 = \{w \in \{a, b, (, )\}^* \mid w \text{ ist korrekt geklammert}\}$ .
2. [2 Punkte]  $\mathcal{L}_2 = \{w \in \{a, b, (, )\}^* \mid |w|_a = 2|w|_b\}$ .
3. [2 Punkte] Können Sie auch einen PDA bauen, der  $\mathcal{L}_1 \cap \mathcal{L}_2 = \{w \in \{a, b, (, )\}^* \mid |w|_a = 2|w|_b \text{ und } w \text{ ist korrekt geklammert}\}$  akzeptiert? Wenn nein, was ist intuitiv das Problem hier?

### Aufgabe 3: CFG, CNF, CYK [8 Punkte]

Der CYK-Algorithmus erwartet als Eingabe eine kontextfreie Grammatik (CFG) in Chomsky-Normalform (CNF). Dies bedeutet, dass alle Produktionsregeln von der Form  $X \rightarrow YZ$  (für Nicht-terminale  $Y, Z$ ) oder von der Form  $X \rightarrow a$  (für ein Terminal  $a$ ) sind.

- a) [4 Punkte] Verwenden Sie das Verfahren aus der Vorlesung, um eine zur CFG  $G = (\{S, X, Y\}, \{a, b, c\}, P, S)$  sprachäquivalente Grammatik in CNF zu berechnen, wobei  $P$  durch die folgenden Regeln definiert ist:

$$\begin{aligned} S &\rightarrow aXbXc, \\ X &\rightarrow Y \mid YYY \mid a, \\ Y &\rightarrow bc \mid cb. \end{aligned}$$

Benutzen Sie ihre gefundene CNF und den Cocke-Younger-Kasami-Algorithmus (CYK-Algorithmus) um zu entscheiden, ob das Wort  $abccbc$  von der obigen Grammatik  $G$  erzeugt wird.

- b) [4 Punkte] Entscheiden Sie mit Hilfe des CYK-Algorithmus, ob die Wörter  $babaa$  und  $baba$  von der Grammatik mit den folgenden Regeln erzeugt werden:

$$\begin{aligned} S &\rightarrow AB \mid BC, \\ A &\rightarrow CC \mid b, \\ B &\rightarrow BA \mid a, \\ C &\rightarrow AB \mid a. \end{aligned}$$

#### Aufgabe 4: Die Syntax einer Programmiersprache als Grammatik [8 Punkte]

In dieser Aufgabe sollen Sie eine Grammatik konstruieren, welche die Syntax einer einfachen Programmiersprache beschreibt.

a) [2 Punkte] Geben Sie eine kontextfreie Grammatik  $G$  an, so dass  $\mathcal{L}(G)$  die Menge der gemäß der weiter unten erklärten Regeln syntaktisch korrekten Programme ist.

- Verwenden Sie als Terminale `id`, `num`, `var`, `if`, `then`, `else`, `end`, `while`, `do`, `;`, `+`, `-`, `*`, `/`, `<`, `>`, `=`, `(`, `)`.

Hierbei ist „`id`“ ein Platzhalter für mögliche Variablennamen und „`num`“ ein Platzhalter für natürliche Zahlen. Die anderen Symbole sind selbsterklärend.

- Ein **Ausdruck** in der Programmiersprache besteht aus Variablen, Zahlen und Operationen, die diese verknüpfen, z.B. `(x+2)`, `(z<500)`, `(x*(y/3))`, `(x==(y+1))`.
- Ein **Programm** ist leer, oder eine Variablendeklaration (z.B. `var x;`), oder eine Zuweisung eines Ausdrucks an eine Variable (z.B. `x=(x+5);`), oder eine bedingte Anweisung (z.B. `if x then y=(z/x); end`), oder eine Fallunterscheidung (z.B. `if x then y=(z/x); else y=z; end`), oder eine Schleife (z.B. `while x do x=(x-1); end`), oder eine Verkettung von zwei Programmen (z.B. `var x; x=500;`).

b) [2 Punkte] Geben Sie die vollständige Ableitung in Ihrer Grammatik aus Teil a) vom Startsymbol zum Programm

```
var x; x=10; var y; y=(x-9); while x do x=(x-1); y=(y+1); end
an.
```

(Sie müssen hierzu zunächst die Variablen durch `id` und die Zahlen durch `num` ersetzen.)

c) [2 Punkte] Beweisen Sie, dass  $\mathcal{L}(G)$  nicht regulär ist.

d) [2 Punkte] Beschreiben Sie, wie die Grammatik aus Teil a) modifiziert werden muss, damit die Programmiersprache **Funktionen** unterstützt.

Funktionen haben einen Namen, eine Parameterliste (potentiell leer) und einen Funktionsrumpf. In diesem dürfen `return;` sowie `return AUSDRUCK;` benutzt werden. Funktionsaufrufe dürfen als Anweisungen und Ausdrücke benutzt werden.

Beispielsweise soll folgendes Wort ein valides Programm sein:

```
function f (var x) return (x+1); end
function g () var y; y=2; y=f(y); return; end
g();
```