Theoretische Informatik 1
Übungsblatt 6

René Maseli
Prof. Dr. Roland Meyer
TU Braunschweig
Wintersemester 2021/22

Ausgabe: 01.02.2022 Abgabe: 10.02.2022, 23:59

Geben Sie Ihre Lösungen bis Donnerstag, 10.02.2022 23:59 Uhr, per E-Mail oder über das StudIP an ihren Tutor ab. Fertigen Sie dazu ihre Hausaufgaben direkt in .pdf Form an oder scannen ihre handschriftlichen Hausaufgaben ein.

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

Hinweis: Es gibt mehrere Gründe, warum ein Wort w nicht in 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 $w^R = w_n w_{n-1} \dots w_2 w_1$. Zu einer Sprache L sei $L^R = \{w^R \mid w \in L\}$.

- [2 Punkte] Zeigen Sie, wie man aus einem NFA A einen NFA B mit $\mathcal{L}(B) = \mathcal{L}(A)^R$ konstruieren kann.
- [2 Punkte] Zeigen Sie, wie man aus einer kontextfreien Grammatik G eine kontextfreie Grammatik H mit $\mathcal{L}(H) = \mathcal{L}(G)^R$ 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: CFG, CNF, CYK [8 Punkte]

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

Betrachten Sie die zwei CFG $G = \langle \{S, X, Y\}, \{a, b\}, P_G, S \rangle$ und $H = \langle \{S, X, Y\}, \{a, b, c\}, P_H, S \rangle$.

$$
\begin{array}{lll}
P_G: & S \to XaY \mid \varepsilon & \\
 & X \to bXYb \mid Y & \\
 & Y \to aXYb \mid \varepsilon &
\end{array}
\qquad
\begin{array}{lll}
P_H: & S \to X \mid aba & \\
 & X \to YYa \mid ab & \\
 & Y \to S \mid c &
\end{array}
$$

a) [1 Punkt] Verwenden Sie das Verfahren aus der Vorlesung, um eine zu G sprachäquivalente Grammatik G_a ohne ε-Produktionen zu berechnen.

b) [1 Punkt] Verwenden Sie G_a und das Verfahren aus der Vorlesung, um eine Grammatik G_b in CNF zu berechnen, welche $\mathcal{L}(G_b) = \mathcal{L}(G)$ erfüllt.

c) [1 Punkt] Benutzen Sie G_b und den CYK-Algorithmus, um zu entscheiden, ob das Wort *bbaab* von G erzeugt wird.

d) [2 Punkte] Entscheiden Sie mit Hilfe von G_b und des CYK-Algorithmus, ob $ababbaa \in \mathcal{L}(G)$ gilt.

e) [1 Punkt] Verwenden Sie das Verfahren aus der Vorlesung, um eine zu H sprachäquivalente Grammatik H_e in CNF zu berechnen.

f) [1 Punkt] Benutzen Sie H_e und den CYK-Algorithmus, um zu entscheiden, ob das Wort *ccaba* von H erzeugt wird.

g) [1 Punkt] Zeigen oder widerlegen Sie $cabaa \in \mathcal{L}(H)$ mit Hilfe von H_e und des CYK-Algorithmus.

Aufgabe 3: Die Syntax einer Programmiersprache als Grammatik [8 Punkte]

Die Syntax von Programmiersprachen ist üblicherweise als kontextfreie Grammatik (oft dargestellt als EBNF oder Syntaxdiagramm) definiert. 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 $\Sigma := \{\texttt{id}, \texttt{num}, \texttt{var}, \texttt{if}, \texttt{then}, \texttt{else}, \texttt{end}, \texttt{while}, \texttt{do}, \texttt{;}, \texttt{op}, \texttt{=}, \texttt{(}, \texttt{)}\}$.

  Hierbei sind `id`, `num` und `op` jeweils Platzhalter für mögliche Variablennamen, natürliche Zahlen und binäre Operatoren (einschließlich == etc.). Die anderen Symbole repräsentieren jeweils nur ein Zeichen oder Schüsselwort.

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

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

```
var x; x=10; var y; y=0; while x do x=(x-1); y=((y*2)+1); end
```

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

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

d) [2 Punkte] Modifizieren Sie G, sodass die Programmiersprache Funktionen unterstützt.

Eine **Funktion** beginnt mit dem Schlüsselwort `function` und einem Funktionsnamen, gefolgt von einer durch `,` getrennten Liste von Parametern (potentiell leer), gefolgt von einem Funktionsrumpf (einem Programm), gefolgt vom Schlüsselwort `end`. Im Funktionsrumpf darf die Rückkehr-Anweisung (z.B. `return (x*x)`) benutzt werden.

Funktionsaufrufe dürfen als Anweisungen und Ausdrücke benutzt werden.

Beispielsweise soll folgendes Wort ein valides Programm sein:

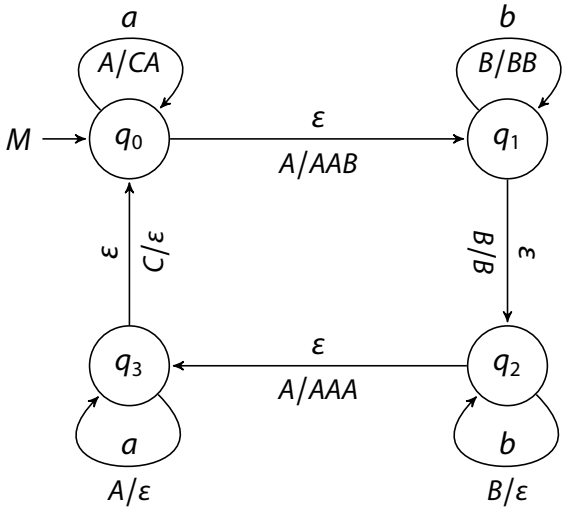
```
function f (x) var y; y=2; return g(x,y) end;

function g (x,y) if (x<y) return x else return y end end;

f(4);
```

Aufgabe 4: Pushdown-Automaten [10 Punkte]

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

1. [2 Punkte] $L_1 = \{w \in \{a, b, (,)\}^* \mid w \text{ ist korrekt geklammert }\}$.
2. [2 Punkte] $L_2 = \{w \in \{a, b, (,)\}^* \mid |w|_a = 2|w|_b\}$.
3. [1 Punkt] $L_1 \cap L_2 = \{w \in \{a, b, (,)\}^* \mid |w|_a = 2|w|_b$ und w ist korrekt geklammert $\}$

   Können Sie auch einen PDA bauen, der $L_1 \cap L_2$ akzeptiert? Wenn nein, was ist intuitiv das Problem hier?
4. [5 Punkte] Gegeben sei der folgende PDA $M = \langle\{q_0, q_1, q_2, q_3\}, \{a, b\}, \{A, B, C\}, A, q_0, \delta\rangle$, welcher bei leerem Stapel akzeptiert. Nutzen Sie die Tripel-Konstruktion, um eine Grammatik für $\mathcal{L}(M)$ zu finden. Beschränken Sie sich ausschließlich auf die nützlichen Nichtterminale.



Hinweis: Es gibt 8 nützliche Nichtterminale und darüber 10 Produktionen.