Institut für Theoretische Informatik
Lehrstuhl Prof. Dr. D. Wagner

Übungsblatt 6

Vorlesung Theoretische Grundlagen der Informatik im WS 19/20

Ausgabe: 8. Dezember 2019
Abgabe: 21. Januar 2020, 11:00 Uhr (im Kasten im UG von Gebäude 50.34)

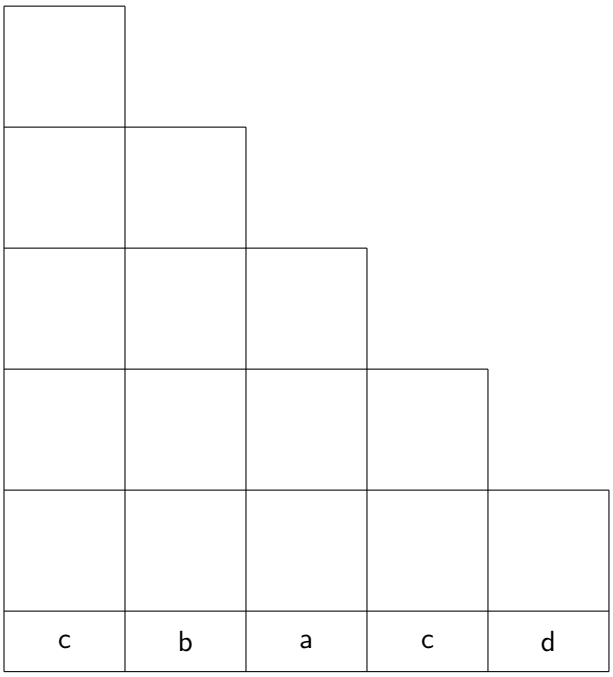
Aufgabe 1

(2 + 2 + 3 + 1 = 8 Punkte)

Gegeben sei die Grammatik $G = (\Sigma, V, S, R)$ mit Terminalen $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}\}$, Nichtterminalen $V = \{S, A, B, C, D\}$, Startsymbol S und Produktionen

$$\begin{aligned} R = \{ & S \to AD \\ & A \to CB \mid \mathsf{a} \mid \mathsf{c} \\ & B \to AD \mid \mathsf{b} \mid \mathsf{d} \\ & C \to DB \mid \mathsf{c} \\ & D \to AC \mid \mathsf{c} \mid \mathsf{d}\} \end{aligned}$$

(a) Überprüfen Sie mit dem CYK-Algorithmus, ob das Wort cbacd in der Sprache $L(G)$ enthalten ist.



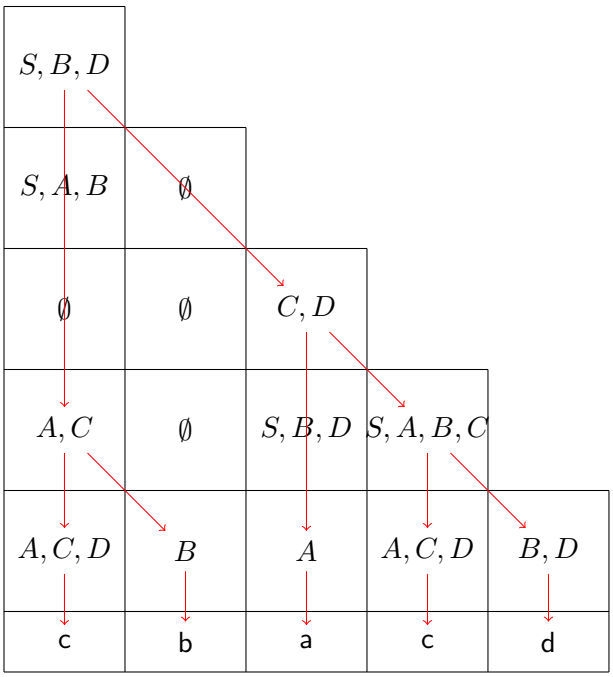
(b) Geben Sie einen Syntaxbaum für das Wort cbacd an.

(c) Erweitern Sie den CYK-Algorithmus so, dass falls das überprüfte Wort tatsächlich von der Grammatik erzeugt wird, ein Syntaxbaum ausgegeben wird.

(d) Wie viel zusätzliche Laufzeit und wie viel zusätzlichen Speicher benötigt Ihr Algorithmus aus Teilaufgabe (c)? Begründen Sie!

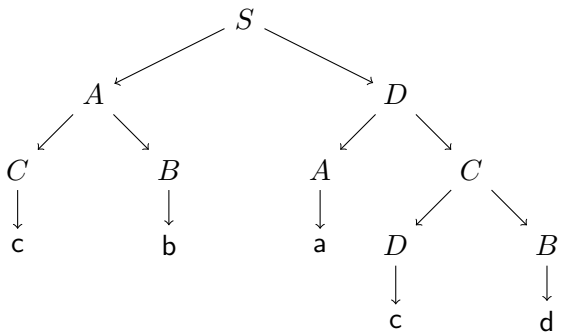
Lösung:

(a) Vergleiche folgende Tabelle:



Das Wort ist also in der Sprache enthalten.

(b) Vergleiche folgenden Syntaxbaum:



(c) Zu jedem Symbol in V_{ij} werden außerdem die ein oder zwei Vorgängersymbole gespeichert: Für $i \neq j$ sind das die zwei Symbole aus V_{ik} und V_{kj}, aus dem es berechnet wurde. Für $i = j$ ist es das eine Terminalsymbol, auf das es abgeleitet wurde. In der obigen Tabelle sind diese Vorgänger durch rote Pfeile markiert. Wird das Wort von der Grammatik erzeugt, kann der Syntaxbaum über die Vorgängersymbole rekursiv ausgegeben werden.

(d) Die Laufzeit zum Speichern der Vorgängersymbole erhöht die Laufzeit um einen konstanten Faktor. Der benötigte Speicherplatz steigt ebenfalls um einen konstanten Faktor. Die Ausgabe des Syntaxbaums eines Wortes der Länge n ist in $\mathcal{O}(n)$ Zeit möglich, da der Syntaxbaum höchstens n Knoten besitzt.

Aufgabe 2

(2 + 2 = 4 Punkte)

Betrachten Sie eine Relaxierung der Chomsky-Normalform, bei der die rechten Seiten von Ableitungsregeln auch aus drei aufeinanderfolgenden Nichtterminalsymbolen bestehen dürfen.

(a) Passen Sie den CYK-Algorithmus an, um mit solchen Grammatiken umgehen zu können, d.h., geben Sie einen Algorithmus an, der das Wortproblem für derartige Grammatiken in Polynomialzeit löst.

(b) Was ist das Laufzeitverhalten Ihres Algorithmus? Beweisen Sie!

Lösung:

(a) Erweitere den CYK-Algorithmus so, dass beim Rekursionsschritt zusätzlich überprüft wird, ob sich das Wort w in *drei* Teile $w = w_1w_2w_3$ zerlegen lässt und eine Regel $V \to WXY$ existiert, sodass w_1 von W, w_2 von X und w_3 von Y ableitbar ist.

(b) Da es von einem Wort quadratisch viele Zerteilungen in drei Teile gibt, dauert ein einzelner Rekursionsschritt nun möglicherweise quadratisch viel Zeit (anstatt linear viel Zeit beim unveränderten CYK-Algorithmus). Dadurch steigt die Gesamtlaufzeit auf $\mathcal{O}(n^4)$.

Aufgabe 3

(2 + 2 + 2 = 6 Punkte)

Beim CYK-Algorithmus wird ausgenutzt, dass ein Wort w von einem Nichtterminalsymbol V genau dann ableitbar ist, wenn eine der folgenden Aussagen wahr ist:

- w ist ein einzelnes Terminalsymbol ist und es existiert eine Regel $V \to w$.
- w lässt sich in zwei Teile w_1w_2 zerlegen, und es existiert eine Regel $V \to WX$ sodass w_1 von W und w_2 von X ableitbar ist.

Der CYK-Algorithmus benutzt diesen Zusammenhang und dynamische Programmierung, um das Wortproblem für Grammatiken von Typ 2 in kubischer Zeit zu lösen.

Betrachten Sie nun Typ-3-Grammatiken.

(a) Wann lässt sich in diesem Fall ein Wort w aus einem Symbol V ableiten?

(b) Beschreiben Sie einen möglichst effizienten Algorithmus, der – analog zum CYK-Algorithmus für Grammatiken von Typ 2 – das Wortproblem für Grammatiken von Typ 3 löst.

(c) Nennen und begründen Sie Laufzeit- und Speicherkomplexität Ihres Algorithmus.

Lösung:

(a) Ein Wort w ist von einem Nichtterminalsymbol V genau dann ableitbar, wenn eine der folgenden Aussagen wahr ist:

- w ist ein einzelnes Terminalsymbol ist und es existiert eine Regel $V \to w$.
- w lässt sich in zwei Teile $w = aw'$ zerlegen, sodass a ein Terminalsymbol ist und eine Regel $V \to aW$ existiert, wobei w' von W ableitbar ist.

(b) Im Gegensatz zum CYK-Algorithmus müssen nicht alle Zerlegungen $w = w_1 w_2$ in Betracht gezogen werden, sondern nur die eine mit $|w_1| = 1$. Es reicht also, nur das oberste Kästchen in jeder Spalte des Dreiecks zu betrachten.

(c) Um ein Kästchen auszufüllen, braucht es nun konstant viel Zeit (im Gegensatz zu linear viel Zeit). Außerdem werden nur linear viele Kästchen ausgefüllt (im Gegensatz zu quadratisch vielen). Dadurch sinkt die Laufzeit von $\mathcal{O}(n^3)$ auf $\mathcal{O}(n)$. Da nur linear viele Kästchen ausgefüllt werden, sinkt der Speicherverbrauch von $\mathcal{O}(n^2)$ auf $\mathcal{O}(n)$. Tatsächlich ist sogar immer nur das (nach rechts unten diagonal) benachbarte Kästchen für das Befüllen des nächsten Kästchens wichtig. Also kann der Speicherverbrauch weiter auf $\mathcal{O}(1)$ reduziert werden, in dem der Inhalt von nicht mehr benötigten Kästchen überschrieben wird.

Aufgabe 4

(3 + 1 = 4 Punkte)

Sei eine Grammatik G durch $V = \{S, X, Y\}, \Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}\}$ und folgende Regelmenge R gegeben:

$$\begin{aligned} S &\to \mathsf{a}S\mathsf{c} \mid X \mid \mathsf{d}Y \\ X &\to \mathsf{b}X\mathsf{c} \mid \varepsilon \\ Y &\to \mathsf{d}S \mid \varepsilon \end{aligned}$$

(a) Verwenden Sie das Verfahren aus der Vorlesung, um G in eine äquivalente Grammatik in erweiterter Chomsky-Normalform zu überführen. Es genügt dabei, wenn sie nach jedem Schritt bzw. jeder Phase das jeweilige Ergebnis angeben.

(b) In Schritt 4, Phase 2 wird beim Ersetzen von Kettenregeln der Form $A \to B$ in umgekehrter topologischer Sortierung vorgegangen. Warum ist dies nötig? Geben Sie eine Grammatik und eine Sortierung der Variablen an, für die das Verfahren nicht zum korrekten Ergebnis führt.

Lösung:

(a) **Schritt 1:** Ersetze gemischte Produktionen mit Terminal- und Nichtterminalsymbolen auf der rechten Seite.

$$\begin{aligned} S &\to ASC \mid X \mid DY \\ X &\to BXC \mid \varepsilon \\ Y &\to DS \mid \varepsilon \\ A &\to \mathsf{a} \\ B &\to \mathsf{b} \\ C &\to \mathsf{c} \\ D &\to \mathsf{d} \end{aligned}$$

Schritt 2: Ersetze Produktionen mit Länge ≥ 3.

$$
\begin{aligned}
S &\to AZ_1 \mid X \mid DY \\
X &\to BZ_2 \mid \varepsilon \\
Y &\to DS \mid \varepsilon \\
Z_1 &\to SC \\
Z_2 &\to XC \\
A &\to \mathsf{a} \\
B &\to \mathsf{b} \\
C &\to \mathsf{c} \\
D &\to \mathsf{d}
\end{aligned}
$$

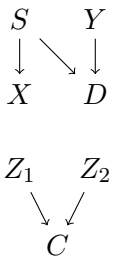
Schritt 3: Ersetze Regeln der Form $A \to \varepsilon$.

Die Menge der Variablen, die sich auf ε ableiten lassen, ist $V' = \{S, X, Y\}$. Für X und Y ist das offensichtlich. Für S lässt sich die Ableitungsfolge $S \to X \to Y$ finden. A, B, C, D sind offensichtlich nicht in V', während Z_1 und Z_2 sind wegen des Vorkommens von C auf der rechten Seite nur in Wörter ableiten lassen, die mindestens ein c enthalten.

$$
\begin{aligned}
S &\to AZ_1 \mid X \mid DY \mid D \\
X &\to BZ_2 \\
Y &\to DS \mid D \\
Z_1 &\to SC \mid C \\
Z_2 &\to XC \mid C \\
A &\to \mathsf{a} \\
B &\to \mathsf{b} \\
C &\to \mathsf{c} \\
D &\to \mathsf{d}
\end{aligned}
$$

Schritt 4: Ersetze Kettenregeln der Form $A \to B$.

Abhängigkeitsgraph:



Zyklische Abhängigkeiten der Form $A_1 \to A_2 \to \cdots \to A_n \to A_1$ gibt es in diesem Produktionssystem nicht. Eine mögliche topologische Sortierung ist S, Y, X, D, Z_1, Z_2, C.

$$
\begin{aligned}
S &\to AZ_1 \mid BZ_2 \mid DY \mid \mathsf{d} \\
X &\to BZ_2 \\
Y &\to DS \mid \mathsf{d} \\
Z_1 &\to SC \mid \mathsf{c} \\
Z_2 &\to XC \mid \mathsf{c} \\
A &\to \mathsf{a} \\
B &\to \mathsf{b} \\
C &\to \mathsf{c} \\
D &\to \mathsf{d}
\end{aligned}
$$

Schritt 5: Stelle die Ableitung $S \xrightarrow{\star} \varepsilon$ wieder her.

Das Endergebnis ist eine Grammatik G' mit $V' = \{S', S, X, Y, Z_1, Z_2, A, B, C, D\}, \Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}\}$ und Regelmenge R':

$$
\begin{aligned}
S' &\to S \mid \varepsilon \\
S &\to AZ_1 \mid BZ_2 \mid DY \mid \mathsf{d} \\
X &\to BZ_2 \\
Y &\to DS \mid \mathsf{d} \\
Z_1 &\to SC \mid \mathsf{c} \\
Z_2 &\to XC \mid \mathsf{c} \\
A &\to \mathsf{a} \\
B &\to \mathsf{b} \\
C &\to \mathsf{c} \\
D &\to \mathsf{d}
\end{aligned}
$$

(b) Betrachte z.B. die Grammatik G mit $V = \{A, B, C\}$, $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$, Startsymbol A und Regelmenge R:

$$
\begin{aligned}
A &\to B \mid \mathsf{a} \\
B &\to C \mid \mathsf{b} \\
C &\to AB \mid \mathsf{c}
\end{aligned}
$$

Schritt 1 bis Schritt 4, Phase 1 lassen diese Grammatik unverändert. In Schritt 4, Phase 2 ist die einzige mögliche topologische Sortierung A, B, C. Wird beim Ersetzen der Kettenregeln aber z.B. in der Reihenfolge A, B, C vorgegangen, erhalten wir folgendes Ergebnis:

$$
\begin{aligned}
A &\to C \mid \mathsf{b} \mid \mathsf{a} \\
B &\to AB \mid \mathsf{c} \mid \mathsf{b} \\
C &\to AB \mid \mathsf{c}
\end{aligned}
$$

Es gibt danach also immer noch eine Kettenregel $A \to C$, bei der C noch durch AB ersetzt werden müsste. Das wurde aber nicht gemacht, weil A vor C abgearbeitet wurde.

Aufgabe 5 (2 + 4 = 6 Punkte)

Zeigen Sie, dass die kontextsensitiven Sprachen den Sprachen der Klasse $\mathsf{NTAPE}(n)$ entsprechen[1], indem Sie in zwei Schritten vorgehen.

(a) Sei L eine kontextsensitive Sprache. Dann existiert eine nichtdeterministische Turingmaschine M, die L mit linearem Platzbedarf akzeptiert.

(b) Sei M eine nichtdeterministische Turingmaschine, die die Sprache $L(M)$ mit linearem Platzbedarf akzeptiert. Dann existiert eine kontextsensitive Grammatik G, so dass $L(G) = L(M)$ gilt.

Lösung:

(a) Sei L eine kontextsensitive Sprache, und G eine Typ-1-Grammatik, die L erzeugt. Zeige, dass L von einer nichtdeterministischen Turingmaschine T mit linearem Platzbedarf entschieden werden kann. Die Grammatik G hat konstante Größe und lässt sich damit in $\mathcal{O}(1)$ Platz auf das Band von T schreiben. Sei nun w eine Eingabe für T. Es gilt zu entscheiden, ob $w \in L$ ist. Im Falle $w = \varepsilon$ gilt $w \in L$ genau dann, wenn G die Ableitungsregel $S \to \varepsilon$ enthält. Andernfalls wird wie folgt vorgegangen. Enthält w ein Teilwort v, so dass G eine Regel $u \to v$ enthält, ist $u \to v$ eine *mögliche* Ableitung. Die NTM T sucht in jedem Schritt erst alle möglichen Ableitungen und wählt danach eine nichtdeterministisch aus. Dann wird v durch u ersetzt. Steht irgendwann nur noch S auf dem Band, hält T und akzeptiert. Da T nur Ableitungen vornimmt, die durch G kodiert werden, gilt $w \in L$, falls T akzeptiert. Umgekehrt akzeptiert T, falls $w \in L$ gilt, z.B. indem die Ableitungen, die von S zu w führen, in umgekehrter Reihenfolge vorgenommen werden. Man beachte, dass die Eingabe beim umgekehrten Anwenden der Ableitungen immer kürzer wird. Zusammen mit der in $\mathcal{O}(1)$ kodierten Grammatik wird so also höchstens linear viel Platz benötigt.

(b) Wir verwenden für diese Aufgabe das alternative NTM-Modell, bei der kein Orakel benutzt wird, sondern in jedem Schritt Wahlmöglichkeiten bei der Überführungsfunktion existieren (vgl. Übung 5 sowie Übungsblatt 4, Aufgabe 7). Außerdem gehen wir davon aus, dass die NTM statt einem separaten Read-Only-Eingabeband eine separate Eingabe*spur* auf dem Arbeitsband hat. Die NTM hat also nur einen Kopf und ein zweispuriges Band, wobei auf Spur 1 die Eingabe steht und der Kopf nur auf Spur 2 schreiben darf.

Sei nun $L \in \mathsf{NTAPE}(n)$ und $T = (Q, \Sigma, \Sigma \times \Gamma, s, \delta, F)$ eine NTM, die L akzeptiert. Das Bandalphabet besteht aus Tupeln (σ, γ), wobei $\sigma \in \Sigma$ das Eingabesymbol auf Spur 1 ist und $\gamma \in \Gamma$ das „Arbeitssymbol“ auf Spur 2. Übergänge haben die Form $(q, \sigma, \gamma) \mapsto (\hat{q}, \hat{\gamma}, d)$. Dabei ist q der aktuelle Zustand, σ das aktuell gelesene Eingabesymbol, γ das aktuell gelesene Arbeitssymbol. Diese werden abgebildet auf einen neuen Zustand $\hat{q}$, ein neues Arbeitssymbol $\hat{\gamma}$ und eine Kopfbewegung $d \in \{L, N, R\}$.

Wir konstruieren eine Grammatik G, die die Sprache L erzeugt. Das Terminalalphabet von G sei gerade Σ. Die Idee ist es, die Kopfbewegungen und Zustandsübergänge von T als Ableitungen in G zu kodieren. Dazu kodieren wir die aktuelle Bandkonfiguration als Kette von Nichtterminalsymbolen. Jedes Nichtterminalsymbol ist ein Tripel $(q, \sigma, \gamma) \in (Q \cup \{\bot\}) \times (\Sigma \cup \{\sqcup\}) \times (\Gamma \cup \{\sqcup\})$, das ein Feld auf dem Band repräsentiert. Dabei steht q für den Zustand der Kontrolleinheit bzw. $\bot$, falls der Kopf nicht auf diesem Feld steht. Das Eingabesymbol auf Spur 1 wird durch σ repräsentiert und das Arbeitssymbol auf Spur 2 durch γ.

[1] Vergleiche Vorlesung 13, Folie 18.

Durch folgende Regeln lässt sich die Anfangskonfiguration für jedes mögliche Eingabewort erzeugen:

$$\begin{aligned} & S \;\to\; (\bot, \sqcup, \sqcup)\ A\ (\bot, \sqcup, \sqcup) \\ \forall \sigma \in \Sigma: \quad & A \;\to\; (s, \sigma, \sqcup) \mid (s, \sigma, \sqcup)\ B \\ \forall \sigma \in \Sigma: \quad & B \;\to\; BB \mid (\bot, \sigma, \sqcup) \end{aligned}$$

Man beachte, dass der Kopf von T zunächst im Zustand s auf dem linkesten Symbol der Eingabe steht. Die Eingabe wird links und rechts von je einem leeren Feld der Form $(\bot, \sqcup, \sqcup)$ begrenzt.

Die Idee ist nun, dass sich diese Anfangskonfiguration genau dann zum Eingabewort w ableiten lässt, wenn w von M akzeptiert wird. Füge deshalb für jeden Übergang $(q, \sigma, \gamma) \mapsto (\hat{q}, \hat{\gamma}, d)$ und alle $\sigma' \in \Sigma$,$\gamma' \in \Gamma$ folgende Regeln hinzu:

$$\begin{aligned} (\bot, \sigma', \gamma')\ (q, \sigma, \gamma) \;&\to\; (\hat{q}, \sigma', \gamma')\ (\bot, \sigma, \hat{\gamma}) && \text{falls } d = L \\ (q, \sigma, \gamma) \;&\to\; (\hat{q}, \sigma, \hat{\gamma}) && \text{falls } d = N \\ (q, \sigma, \gamma)\ (\bot, \sigma', \gamma') \;&\to\; (\bot, \sigma, \hat{\gamma})\ (\hat{q}, \sigma', \gamma') && \text{falls } d = R \end{aligned}$$

Sobald ein akzeptierender Zustand $f \in F$ erreicht ist, muss das ursprüngliche Wort entpackt werden.

$$\begin{aligned} (f, \sigma, \gamma) \;&\to\; \sigma \\ (\bot, \sigma, \gamma)\ \sigma' \;&\to\; \sigma\ \sigma' \\ \sigma'\ (\bot, \sigma, \gamma) \;&\to\; \sigma'\ \sigma \end{aligned}$$

Ist $\varepsilon \in L$, fügen wir außerdem noch die Regel $S \to \varepsilon$ hinzu.

Aufgabe 6 (4 Punkte)

Zeigen Sie, dass folgende Sprachen nicht kontextfrei sind[2]:

(a) $L_1 = \{\mathtt{0}^n \mathtt{1}^{2n} \mathtt{0}^n \mid n \in \mathbb{N}\}$

(b) $L_2 = \{aa \mid a \in \{\mathtt{0}, \mathtt{1}\}^*\}$

(c) $L_3 = \{a\#b\#c \mid a, b, c \in \{\mathtt{0}, \mathtt{1}\}^*, a + b = c \text{ als Binärzahlen aufgefasst}\}$

Lösung:

(a) Wende das Pumping-Lemma an und wähle $z = \mathtt{0}^n \mathtt{1}^{2n} \mathtt{0}^n$. Wegen $|vwx| \leq n$ kann vx nicht sowohl Nullen aus dem vorderen Teil des Wortes als auch aus dem hinteren Teil des Wortes enthalten. Falls vx überhaupt keine Nullen enthält, gilt $uv^2wx^2y = \mathtt{0}^n \mathtt{1}^m \mathtt{0}^n$ mit $m > 2n$, also $uv^2wx^2y \notin L_1$. Falls vx nur Nullen aus einem Teil des Wortes enthält, enthält in uv^2wx^2y dieser Teil mehr Nullen als der andere.

[2]Dazu können Sie z.B. das Pumping-Lemma für kontextfreie Sprachen verwenden, das am 09.01. in der Vorlesung behandelt wird.

(b) Wende das Pumping-Lemma an und wähle $z = \mathtt{0}^n\mathtt{1}^n\mathtt{0}^n\mathtt{1}^n$. Damit $uv^2wx^2y \in L_2$ gilt, muss v in der vorderen Hälfte von z liegen, x in der hinteren Hälfte, und es muss $v = x$ gelten. Wegen $|vwx| \leq n$ kann nur dann $v = x$ gelten, wenn vwx komplett aus Nullen oder komplett aus Einsen besteht. Dann befinden sich v und x aber in derselben Hälfte des Wortes.

(c) Wende das Pumping-Lemma an und wähle $z = \mathtt{10}^n\#\mathtt{10}^n\#\mathtt{10}^{n+1}$. Als Binärzahlen aufgefasst gilt also $a = b = 2^n$ und $c = 2^{n+1}$. Damit $uwy \in L_3$ gilt, muss v ein Teilwort von a oder b sein und x ein Teilwort von c. Wegen $|vwx| \leq n$ kann x nur dann ein Teilwort von c sein, wenn v nur aus Nullen von b besteht. Außerdem darf x nicht die Eins von c enthalten, weil c sonst in uwy zu 0 wird. Es gilt also $v = \mathtt{0}^i$ und $x = \mathtt{0}^j$ mit $i, j > 0$. Damit $uv^2wx^2y \in L_3$ gilt, muss $2^n + 2^{n+i} = 2^{n+j+1}$ gelten. Das ist äquivalent zu $2^i + 1 = 2^{j+1}$, was als ganzzahlige Lösung nur $i = j = 0$ besitzt.