2. Klausur zur Vorlesung
Theoretische Grundlagen der Informatik
Wintersemester 2020/2021

Lösung!

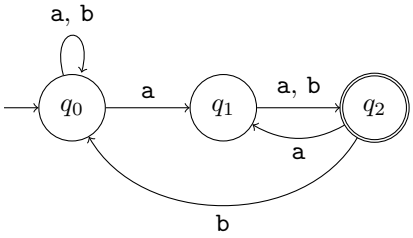
Beachten Sie:

- Bringen Sie den Aufkleber mit Ihrem Namen und Ihrer Matrikelnummer auf diesem Deckblatt an und beschriften Sie jedes Aufgabenblatt mit Ihrem Namen und Matrikelnummer.
- Schreiben Sie die Lösungen auf die Aufgabenblätter und Rückseiten. Am Ende der Klausur sind zusätzliche Leerseiten. Fordern Sie zusätzliches Papier bitte nur an, falls Sie den gesamten Platz aufgebraucht haben.
- Es sind keine Hilfsmittel zugelassen.

	Mögliche Punkte						Erreichte Punkte					
	a	b	c	d	e	Σ	a	b	c	d	e	Σ
Aufg. 1	2	4	2	1	–	9					–	
Aufg. 2	7					7						
Aufg. 3	4	3	4	–	–	11				–	–	
Aufg. 4	2	7	–	–	–	9			–	–	–	
Aufg. 5	2	1	2	3	3	11						
Aufg. 6	4	4	–	–	–	8			–	–	–	
Aufg. 7	5					5						
Σ						60						

Problem 1: Endliche Automaten $2 + 4 + 2 + 1 = 9$ Punkte

Gegeben ist folgender NEA:



(a) Geben Sie für die Sprache, die von diesem Automaten erkannt wird, einen regulären Ausdruck an.

(b) Geben Sie einen äquivalenten DEA an.

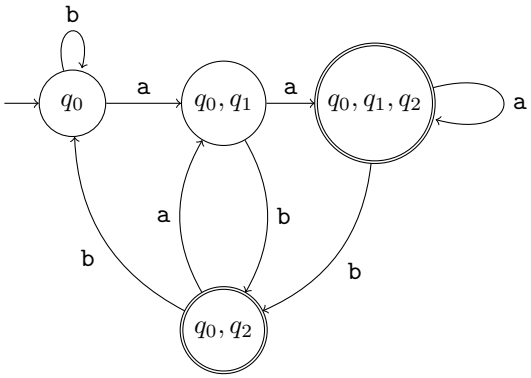
(c) Sei Σ ein beliebiges Alphabet mit 2 Zeichen. Beschreiben Sie ein *deterministisches* Verfahren, das als Eingabe einen NEA $\mathcal{A}$ mit n Zuständen und ein Wort w der Länge m bekommt und entscheidet, ob w von $\mathcal{A}$ akzeptiert wird. Ist die Laufzeit des Verfahrens polynomiell in der Eingabegröße?

(d) Das Wortproblem für reguläre Sprachen kann in Linearzeit gelöst werden. Warum widerspricht das nicht Ihrer Lösung für Aufgabenteil (c)?

Lösung:

(a) $(\mathtt{a} \cup \mathtt{b})^*\mathtt{a}(\mathtt{a} \cup \mathtt{b})$

(b) Folgender DEA ist äquivalent:



(c) Überführe $\mathcal{A}$ mit der Potenzmengenkonstruktion in einen DEA $\mathcal{A}'$ und simuliere dann $\mathcal{A}'$ mit Eingabe w. Da $\mathcal{A}'$ bis zu 2^n Zustände haben kann, ist die Potenzmengenkonstruktion (und damit das gesamte Verfahren) nicht polynomiell.

Alternativlösung: Entferne zunächst mit dem Verfahren aus der Vorlesung die ε-Übergänge aus $\mathcal{A}$. Dafür wird Zeit $\mathcal{O}(n^3)$ benötigt. Arbeite dann das Wort w Zeichen für Zeichen ab und berechne dabei in jedem Schritt die Menge aller erreichbaren Zustände. Anfangs ist nur der Startzustand erreichbar, d.h. beginne mit $Q_0 = \{s\}$. Für das Zeichen w_i berechne $Q_i = \{q \in Q \mid \exists q' \in Q_{i-1} : q \in \delta(q', w_i)\}$. Das ist die Menge der Zustände, die von Q_{i-1} durch Lesen von w_i erreichbar sind. Akzeptiere genau dann, wenn Q_m mindestens einen Endzustand enthält. Die Berechnung von Q_i benötigt Zeit $\mathcal{O}(n^2)$, da Q_{i-1} maximal n Zustände enthalten kann und jeder Zustand maximal n Nachfolgezustände haben kann. Insgesamt ergibt sich eine Laufzeit von $\mathcal{O}(n^3 + mn^2)$, also ist das Verfahren polynomiell.

(d) Beim Wortproblem ist die Sprache L fest und die Eingabe besteht nur aus dem Wort w. Die Anzahl der Zustände im DEA für L ist also eine Konstante.

Problem 2: Pumping-Lemma 7 Punkte

Wir betrachten die folgenden zwei Sprachen.

$$L_1 = \{w \in \{0,1\}^* \colon |w|_1 - |w|_0 = 1\}$$
$$L_2 = \{w \in \{0,1\}^* \colon |w|_1 - |w|_0 \equiv 1 \mod 2\},$$

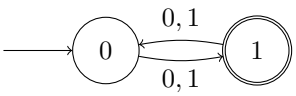
wobei $|w|_i$ für $i = 0, 1$ die Anzahl der Vorkommen von i in w bezeichnet.

Ist L_1 bzw. L_2 regulär? Zeigen bzw. widerlegen Sie jeweils.

Lösung:

L_1 ist nicht regulär, da die Aussage des Pumping-Lemmas nicht gilt. Sei $n \in \mathbb{N}$ beliebig. Wir wählen als Wort $w = 0^n 1^{n+1}$. Es gilt $|w| = 2n + 1 > n$ und $|w|_1 - |w|_0 = n + 1 - n = 1$, also $w \in L_1$. Wir betrachten eine beliebige Zerlegung von w in uvx mit $|uv| \leq n$ und $v \neq \varepsilon$. Da $|uv| \leq n$, besteht uv nur aus Nullen. Mit $v \neq \varepsilon$ folgt $v = 0^m$ für ein $m > 0$. Also enthält $w' = uv^0x$ echt weniger Nullen als w, wobei die Anzahl der Einsen gleich bleibt. Es folgt $|w'|_1 - |w'|_0 > |w|_1 - |w|_0 = 1$. Damit liegt w' nicht in L_1 und die Aussage des Pumping-Lemmas ist widerlegt.

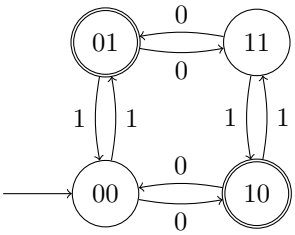
L_2 ist regulär. Hierfür betrachten wir den folgenden endlichen Automaten mit Zustandsmenge $Q = \{0, 1\}$, Startzustand 0, akzeptierenden Zustand 1 und Alphabet $\Sigma = \{0, 1\}$.



Der Automat befindet sich genau dann in Zustand i, wenn die Differenz der Anzahl an Nullen und Einsen modulo 2 gleich i ist. Der angegebene Automat akzeptiert also genau L_2 und wir folgern, dass L_2 regulär ist.

Alternativer Automat für L_2:

Wir betrachten den folgenden endlichen Automaten mit Zustandsmenge $Q = \{00, 01, 10, 11\}$, Startzustand 00, akzeptierenden Zuständen $F = \{01, 10\}$ und Alphabet $\Sigma = \{0, 1\}$.



Der Automat befindet sich genau dann in Zustand ij, wenn die Anzahl der bisher gelesenen Nullen modulo 2 kongruent zu i und die Anzahl Einsen modulo 2 kongruent zu j ist. Er akzeptiert genau dann, wenn die Anzahl von Nullen und Einsen modulo 2 verschieden ist, d.h. die Differenz ist 1. Der angegebene Automat akzeptiert also genau L_2 und wir folgern, dass L_2 regulär ist.

Problem 3: Grammatiken $4 + 3 + 4 = 11$ Punkte

Wir definieren die Sprachen L über dem Alphabet $\Sigma = \{\mathtt{0}, \mathtt{1}\}$ und L' über dem Alphabet $\Sigma' = \{\mathtt{a}, \mathtt{b}\}$ durch

$$L = \{(\mathtt{01})^n(\mathtt{10})^n \mid n > 0\} \text{ und}$$
$$L' = \{\mathtt{a}^n\mathtt{b} \mid n > 0\} \cup \{\mathtt{ba}^n \mid n > 0\}.$$

(a) Geben Sie eine Grammatik für L und eine Grammatik für L' an.

Nun betrachten wir ein Wort $w \in L$ und ersetzen jede $\mathtt{0}$ in w durch ein beliebiges Wort aus L'.

Beispiel: $w = \mathtt{01011010} \rightsquigarrow \mathtt{ab1ba11ab1ab}, \mathtt{baa1ab11aaab1ba}, \ldots$

Die Sprache aller Wörter, die so entstehen können, bezeichnen wir mit $\alpha_w(L')$. Außerdem definieren wir

$$\alpha_L(L') = \bigcup_{w \in L} \alpha_w(L').$$

Das heißt, $\alpha_L(L')$ enthält alle Wörter, die entstehen, wenn wir mit einem beliebigen Wort aus L beginnen und jede $\mathtt{0}$ durch ein beliebiges Wort aus L' ersetzen.

(b) Geben Sie eine Grammatik für $\alpha_L(L')$ an.

(c) Seien nun L und L' beliebige kontextfreie Sprachen über $\Sigma = \{\mathtt{0}, \mathtt{1}\}$ bzw. $\Sigma' = \{\mathtt{a}, \mathtt{b}\}$. Zeigen Sie, dass $\alpha_L(L')$ ebenfalls kontextfrei ist.

Lösung:

(a) $G = (\Sigma, V, S, R)$ für L mit $V = \{S, X, Y\}$, Startsymbol S und

$$R = \{S \to XSY \mid XY$$
$$X \to \mathtt{01}$$
$$Y \to \mathtt{10}\}$$

$G' = (\Sigma', V', S', R')$ für L' mit $V' = \{S', A\}$, Startsymbol S' und

$$R' = \{S' \to A\mathtt{b} \mid \mathtt{b}A$$
$$A \to \mathtt{a}A \mid \mathtt{a}\}$$

(b) $G = (\Sigma_\alpha, V_\alpha, S, R_\alpha)$ mit $\Sigma_\alpha = \{\mathtt{1}, \mathtt{a}, \mathtt{b}\}, V_\alpha = \{S, X, Y, S', A\}$, Startsymbol S und

$$R_\alpha = \{S \to XSY \mid XY$$
$$X \to S'\mathtt{1}$$
$$Y \to \mathtt{1}S'$$
$$S' \to A\mathtt{b} \mid \mathtt{b}A$$
$$A \to \mathtt{a}A \mid \mathtt{a}\}$$

(c) Seien $G = (\Sigma, V, S, R)$ und $G' = (\Sigma', V', S', R')$ kontextfreie Grammatiken für L und L' in Chomsky-Normalform. Insbesondere bildet jede Ableitungsregel, die auf Terminale abbildet, auf genau ein Terminal ab. Ohne Einschränkung sind V und V' disjunkt. Wir konstruieren eine kontextfreie Grammatik $G_\alpha = (\Sigma_\alpha, V_\alpha, S, R_\alpha)$ für $\alpha_L(L')$. Hierfür setzen wir $\Sigma_\alpha = \{\mathtt{1}, \mathtt{a}, \mathtt{b}\}$ und $V_\alpha = V \cup V'$. Die Regelmenge R_α besteht aus allen Regeln aus R' und zusätzlich allen Regeln aus R, wobei für $X \in V$ Regeln der Form $X \to \mathtt{0}$ durch $X \to S'$ ersetzt werden.

Problem 4: NP-Vollständigkeit $2 + 7 = 9$ Punkte

In der Vorlesung wurde das NP-vollständige Entscheidungsproblem 3-SAT vorgestellt:

Gegeben:	Menge U von Variablen Menge C von Klauseln über U, wobei jede Klausel höchstens drei Literale enthält
Frage:	Existiert eine Belegung der Variablen mit `wahr` und `falsch`, sodass in jeder Klausel mindestens ein Literal `wahr` ist?

Anmerkung: In der Vorlesung wurde gefordert, dass jede Klausel genau *drei Literate enthält. Hier wird nur gefordert, dass jede Klausel* höchstens *drei Literale enthält. Auch in dieser Variante ist* 3-SAT NP*-vollständig.*

Wir betrachten nun das leicht abgewandelte Entscheidungsproblem MONOTONENOTALLEQUAL (MNAE):

Gegeben:	Menge U von Variablen Menge C von Klauseln über U, wobei jede Klausel höchstens drei Literale enthält und kein Literal negiert ist
Frage:	Existiert eine Belegung der Variablen mit `wahr` und `falsch`, sodass in jeder Klausel mindestens ein Literal `wahr` ist und mindestens ein Literal `falsch` ist?

Sie dürfen voraussetzen, dass MONOTONENOTALLEQUAL NP-vollständig ist.

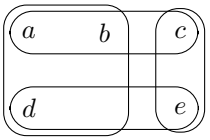
Außerdem betrachten wir das folgende MENGENFÄRBUNGSPROBLEM (MFP):

Gegeben:	Eine Menge X Eine Menge $\mathcal{F}$ von höchstens dreielementigen Teilmengen von X
Frage:	Existiert eine Färbung der Elemente aus X in rot und blau, sodass keine Menge aus $\mathcal{F}$ einfarbig ist? (Eine Färbung weist jedem Element aus X genau eine der Farben rot oder blau zu.)

(a) Zeigen Sie für die folgenden beiden Instanzen, dass es sich um Ja-Instanzen handelt. Tragen Sie hierzu die Farben bzw. Wahrheitswerte in die Tabelle unten ein.

- MENGENFÄRBUNGSPROBLEM

  $X = \{a, b, c, d, e\}, \mathcal{F} = \{\{a, b, c\}, \{a, b, d\}, \{c, e\}, \{d, e\}\}$



- MONOTONENOTALLEQUAL

  $U = \{a, b, c, d, e\}, C = \{(a, b, c), (a, b, d), (c, e), (d, e)\}$

	a	b	c	d	e
MENGENFÄRBUNGSPROBLEM	rot	rot	blau	blau	rot
MONOTONENOTALLEQUAL	`wahr`	`wahr`	`falsch`	`falsch`	`wahr`

(b) Zeigen Sie, dass das MENGENFÄRBUNGSPROBLEM NP-vollständig ist.

Lösung:
Für eine gegebene Färbung von X iterieren wir über die Mengen aus $\mathcal{F}$ und prüfen, ob sie mindestens ein rotes und mindestens ein blaues Element enthalten. Da wir jede Menge nur ein Mal betrachten, lässt sich dies in Linearzeit durchführen. Also liegt das MENGENFÄRBUNGSPROBLEM in NP.

Wir reduzieren MONOTONENOTALLEQUAL auf das MENGENFÄRBUNGSPROBLEM. Gegeben eine MONOTONENOTALLEQUAL-Instanz (U, C), konstruieren wir eine MENGENFÄRBUNGSPROBLEM-Instanz $(X, \mathcal{F})$, sodass (U, C) genau dann eine Ja-Instanz ist, wenn $(X, \mathcal{F})$ eine Ja-Instanz ist.

Sei (U, C) eine MONOTONENOTALLEQUAL-Instanz. Wir konstruieren die MENGENFÄRBUNGSPROBLEM-Instanz $(X, \mathcal{F})$, indem wir $X = U$ setzen und für jede Klausel (x, y, z) die Menge $\{x, y, z\}$ in $\mathcal{F}$ einführen. Hierfür betrachten wir jede Variable und zuätzlich jede Klausel ein Mal, weshalb die Konstruktion in Linearzeit realisierbar ist. Wir zeigen, dass es für (U, C) genau dann eine Variablenbelegung mit je einem wahren und einem falschen Literal pro Klausel gibt, wenn $(X, \mathcal{F})$ so färbbar ist, dass keine Menge einfarbig ist.

$\Longrightarrow$ Sei ϕ eine Belegung der Variablen, sodass jede Klausel ein wahres und ein falsches Literal enthält. Wir färben ein Elemente $x \in X$ rot, falls $\phi(x) = \texttt{wahr}$, und blau, falls $\phi(x) = \texttt{falsch}$. Da jede Klausel ein wahres und ein falsches Literal enthält, enthält jede Menge aus $\mathcal{F}$ ein rotes und ein blaues Element.

$\Longleftarrow$ Wir betrachten eine Färbung von X, sodass keine Menge aus $\mathcal{F}$ einfarbig ist. Wir setzen eine Variable $u \in U$ auf `wahr`, falls sie rot gefärbt ist, und auf `falsch`, falls sie blau gefärbt ist. Da jede Menge ein rotes und ein blaues Element enthält, enthält jede Klausel ein wahres und ein falsches Literal.

Problem 5: Pizza $2 + 1 + 2 + 3 + 3 = 11$ Punkte

Der ebenso gewissenhafte wie geschäftstüchtige Doktor Meta hat beschlossen, dem Verbrechen den Rücken zu kehren. Stattdessen hat er einen Pizzalieferdienst gegründet. Er hat sich eine unendlich große Armee von Robotern gebaut, die beliebig viele Pizzen gleichzeitig ausliefern können. Leider hat er aber vergessen, Köche einzustellen. Also muss er alle Pizzen selber zubereiten.

Sei S die Menge der Bestellungen, die Doktor Meta innerhalb eines Tages bekommt. Jede Bestellung j trifft zu einem Bestellzeitpunkt B_j ein und hat eine Zubereitungszeit Z_j sowie eine Lieferzeit L_j. Sobald er eine Bestellung zu einem Zeitpunkt $F_j \geq B_j + Z_j$ fertig zubereitet hat, kann sie dank der Roboterarmee direkt ausgeliefert werden und kommt zum Zeitpunkt $A_j = F_j + L_j$ beim Kunden an. Weil Doktor Meta möglichst früh Feierabend machen möchte, sucht er eine Zubereitungsreihenfolge, die $A_{\max} = \max_{j \in S} A_j$ minimiert, also die Ankunftszeit der letzten Bestellung.

Da Doktor Meta kein gelernter Koch ist, kann er immer nur eine Bestellung gleichzeitig zubereiten. Außerdem ist er nicht in der Lage, die Zubereitung einer Bestellung zu unterbrechen. Wenn er eine Bestellung fertig zubereitet hat, wählt er eine beliebige neue Bestellung aus, die schon eingetroffen ist, aber noch nicht abgearbeitet wurde, und bereitet sie zu. Sollte es gerade keine offenen Bestellungen geben, ruht er sich aus und schmiedet Weltherrschaftspläne, bis die nächste Bestellung eintrifft.

Hier noch einmal eine Übersicht der verwendeten Variablen:

Gegeben: Menge S von Bestellungen. Für jede Bestellung $j \in S$:
- Bestellzeitpunkt B_j
- Zubereitungszeit Z_j
- Lieferzeit L_j

Ergebnis: Für jede Bestellung $j \in S$:
- Fertigstellungszeitpunkt $F_j \geq B_j + Z_j$
- Auslieferungszeitpunkt $A_j = F_j + L_j$

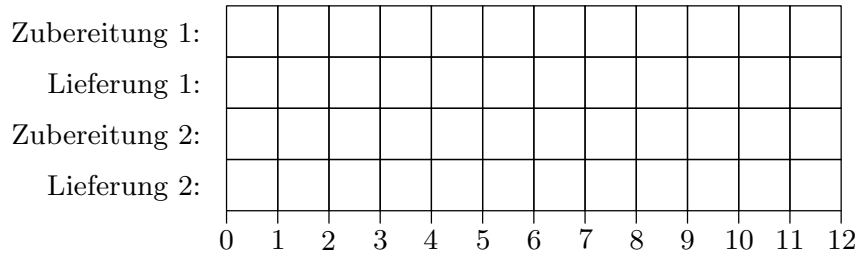
Ziel: Minimiere $A_{\max} = \max_{j \in S} A_j$

Doktor Meta ist mit der Situation unzufrieden. Er vermutet, dass er sein Vorgehen verbessern und früher Feierabend machen könnte, wenn er die Zubereitung einer Bestellung unterbrechen könnte.

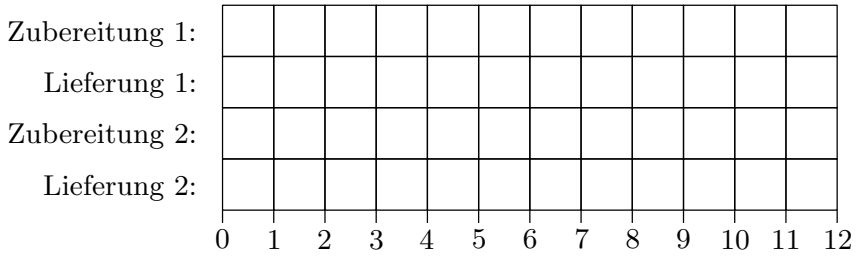
(a) Zeigen Sie, dass Doktor Metas Vermutung stimmt. Gegeben seien zwei Bestellungen:
- Bestellung 1: Bestellzeitpunkt $B_0 = 0$, Zubereitungszeit $Z_0 = 5$, Lieferzeit $L_0 = 1$
- Bestellung 2: Bestellzeitpunkt $B_1 = 1$, Zubereitungszeit $Z_1 = 1$, Lieferzeit $L_1 = 5$

Markieren Sie in den unten stehenden Tabellen die Zeitpunkte, während denen die Bestellungen zubereitet bzw. geliefert werden.

Doktor Metas Lösung (ohne Unterbrechen):



Optimale Lösung (mit Unterbrechen):



Geben Sie für beide Lösungen den Wert von $A_{\max}$ an.

Gegeben: Menge S von Bestellungen. Für jede Bestellung $j \in S$:
- Bestellzeitpunkt B_j
- Zubereitungszeit Z_j
- Lieferzeit L_j

Ergebnis: Für jede Bestellung $j \in S$:
- Fertigstellungszeitpunkt $F_j \geq B_j + Z_j$
- Auslieferungszeitpunkt $A_j = F_j + L_j$

Ziel: Minimiere $A_{\max} = \max_{j \in S} A_j$

Wir bezeichnen im Folgenden mit $A^*_{\max}$ den spätesten Ankunftszeitpunkt in einer optimalen Lösung mit Unterbrechen und mit $A_{\max}$ den spätesten Ankunftszeitpunkt in Doktor Metas Lösung ohne Unterbrechen.

(b) Zeigen Sie: $A^*_{\max} \geq \sum_{j \in S} Z_j$.

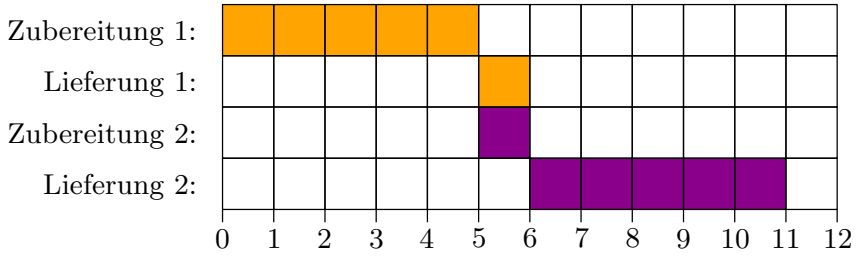
(c) Zeigen Sie, dass für jede Bestellung j gilt: $A^*_{\max} \geq B_j + Z_j + L_j$.

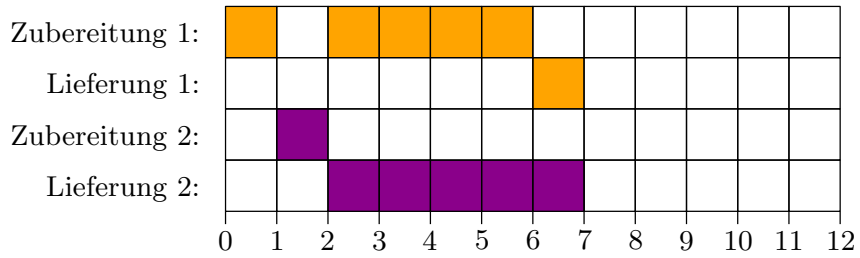
(d) Sei x die letzte Bestellung, die in Doktor Metas Lösung beim Kunden ankommt. Zeigen Sie: $F_x \leq B_x + \sum_{j \in S} Z_j$.

(e) Zeigen Sie, dass das Vorgehen von Doktor Meta ein Approximationsalgorithmus mit relativer Gütegarantie 2 ist.

Lösung:

(a) Doktor Meta hat Recht. Die Lösungswerte sind 11 für seine Lösung und 7 für eine optimale Lösung.





(b) Die letzte Bestellung kann erst beim Kunden ankommen, sobald alle zubereitet wurden.

(c) Für den Ankunftszeitpunkt A_j^* von j in der optimalen Lösung gilt: $A_j^* = F_j + L_j \geq B_j + Z_j + L_j$. Da $A_{\max}^*$ das Maximum über alle j ist, ergibt sich die Behauptung.

(d) Im Zeitraum von B_x bis F_x ist Doktor Meta durchgängig beschäftigt, weil es mit x mindestens eine offene Bestellung gibt. Insgesamt kann er höchstens für $\sum_{j \in S} Z_j$ Zeitschritte beschäftigt sein. Also ist er spätestens zum Zeitpunkt $B_x + \sum_{j \in S} Z_j$ fertig.

(e) $A_{\max} = F_x + L_x \leq B_x + L_x + \sum_{j \in S} Z_j \leq (B_x + Z_x + L_x) + \sum_{j \in S} Z_j \leq 2 \cdot A_{\max}^*$

Problem 6: 2-Kopf-Turingmaschinen $4 + 4 = 8$ Punkte

Wir erweitern das Modell einer Turingmaschine, indem wir einen weiteren Kopf hinzufügen. Die Köpfe haben eine gemeinsame Zustandsmenge Q und gemeinsame Eingabe- und Bandalphabete Σ und Γ. Beide Köpfe arbeiten auf dem selben Band, wobei abwechselnd je ein Schritt der Übergangsfunktionen für Kopf 1 und Kopf 2 ausgeführt wird. Auch wenn Kopf 2 an der Reihe ist, ist bekannt, was Kopf 1 gerade liest, und umgekehrt. Die Übergangsfunktion hat also die Form

$$\delta\colon Q \times \Gamma \times \Gamma \times \{1,2\} \to Q \times \Gamma \times \{L, N, R\},$$

wobei der vierte Parameter angibt, welcher Kopf an der Reihe ist. Die Übergangsfunktion wird also abwechselnd mit 1 bzw. 2 als viertes Argument aufgerufen. Wir nennen eine solche Turingmaschine eine *2-Kopf-Turingmaschine*.

(a) Sei $\Sigma = \{a, b, \#\}$. Beschreiben Sie eine 2-Kopf-Turingmaschine, die die Sprache $\{w\#w\# \mid w \in \{a,b\}^*\}$ in Linearzeit erkennt. Begründen Sie kurz die Laufzeit.

(b) Zeigen Sie, dass 2-Kopf-Turingmaschinen und klassische Turingmaschinen (d.h. 1 Kopf, 1 Band, wie in Vorlesung definiert) gleich mächtig sind. Simulieren Sie hierzu eine klassische Turingmaschine mit einer 2-Kopf-Turingmaschine und umgekehrt.

Lösung:

(a) Kopf 2 läuft bis zum ersten Trennzeichen # und danach noch einen Schritt weiter. Kopf 1 bleibt währenddessen stehen und ändert das Band nicht. Falls kein Trennzeichen existiert, ablehnen. Danach abwechselnd: Kopf 1 vergleicht die beiden Symbole unter den Köpfen. Bei Gleichheit, einen Schritt nach rechts; bei Ungleichheit ablehnen. Kopf 2 geht einen Schritt nach rechts. Nachdem # verglichen wurde, bleibt Kopf 1 stehen und Kopf 2 geht einen Schritt nach rechts, um zu überprüfen, ob das Wort zu Ende ist. Da beide Köpfe nur nach rechts laufen, benötigt die 2-Kopf-Turingmaschine bei Eingabelänge n höchstens $2n$ Schritte.

(b) Eine 2-Kopf-Turingmaschine kann eine klassische Turingmaschine simulieren, indem der zweite Kopf nicht verwendet wird. Umgekehrt kann eine klassische Turingmaschine eine 2-Kopf-Turingmaschine simulieren, indem sie zwischen den beiden Kopfpositionen hin- und herläuft. Dazu müssen die Kopfpositionen gespeichert werden. Dies kann durch ein größeres Bandalphabet $\Gamma' = \Gamma \times \{-, \alpha_1, \alpha_2, \alpha_{12}\}$ realisiert werden. Das Symbol $(\gamma, -)$ wird statt γ verwendet, wenn auf diesem Feld kein Kopf steht, die Symbole (γ, α_1) bzw. (γ, α_2), wenn der entsprechende Kopf auf dem Feld steht, und (γ, α_{12}), wenn beide Köpfe auf dem Feld stehen. Außerdem merken wir uns über die Zustandskontrolle (Verdoppelung der Anzahl an Zuständen), ob Kopf 1 links von Kopf 2 ist oder andersherum. Dadurch wissen wir beim Wechseln der Köpfe, in welche Richtung wir laufen müssen. Beim Laufen zum anderen Kopf merken wir uns mit Hilfe von Zuständen, welches Zeichen wir zuletzt gelesen haben.

Problem 7: Semi-Entscheidbarkeit 5 Punkte

Zeigen Sie, dass die Menge der semi-entscheidbaren Sprachen unter Konkatenation abgeschlossen ist. D.h. zeigen Sie für zwei semi-entscheidbare Sprachen L und L', dass die Sprache $L \cdot L' = \{ww' \mid w \in L, w' \in L'\}$ ebenfalls semi-entscheidbar ist.

Lösung:
Seien $\mathcal{M}$ und $\mathcal{M}'$ Turingmaschinen, die L bzw. L' akzeptieren. Wir konstruieren eine Turingmaschine $\mathcal{M}^+$, die $L \cdot L'$ akzeptiert. Wir entscheiden nicht-deterministisch, an welcher Stelle wir die Eingabe x trennen und erhalten so eine Zerlegung $x = ww'$. Anschließend prüfen wir für beide Teilwörter, ob sie in L bzw. L' sind. Dazu wird zuerst $\mathcal{M}$ mit Eingabe w simuliert, während wir uns w' merken, z.B. auf einem zusätzlichen Band oder auf einem Stück Band, das bei der Simulation von $\mathcal{M}$ übersprungen wird. Falls $\mathcal{M}$ ablehnt oder nicht hält, so tut $\mathcal{M}^+$ das Gleiche. Falls $\mathcal{M}$ akzeptiert, simuliert $\mathcal{M}^+$ die zweite Turingmaschine $\mathcal{M}'$ mit Eingabe w'. Akzeptiert $\mathcal{M}'$, so akzeptiert auch $\mathcal{M}^+$.

Falls also eine Zerlegung der Eingabe x in $x = ww'$ mit $w \in L$ und $w' \in L'$ existiert, so wird x von $\mathcal{M}^+$ akzeptiert. Andernfalls lehnt $\mathcal{M}^+$ ab bzw. hält nicht. Damit wird $L \cdot L'$ von $\mathcal{M}^+$ akzeptiert und ist semi-entscheidbar.

Anmerkung: Falls die Trennung des Wortes deterministisch gemacht wird, müssen die verschiedenen Fälle parallel ausprobiert werden. Werden sie nacheinander ausprobiert, kann es passieren, dass der erste ausprobierte Fall nicht hält, der zweite aber akzeptiert werden sollte.

Alternativlösung: Semi-entscheidbare Sprachen sind genau die Sprachen, die von Typ-0-Grammatiken erzeugt werden. Seien $G_1 = (\Sigma_1, V_1, S_1, R_1)$ und $G_2 = (\Sigma_2, V_2, S_2, R_2)$ Typ-0-Grammatiken für L bzw. L'. O.B.d.A. nehmen wir an, dass V_1 und V_2 disjunkt sind und nicht das Symbol S enthalten. Die Grammatik $G = (\Sigma, V, S, R)$ mit

$$\begin{aligned}\Sigma &= \Sigma_1 \cup \Sigma_2\\ V &= V_1 \cup V_2 \cup \{S\}\\ R &= R_1 \cup R_2 \cup \{S \to S_1 S_2\}\end{aligned}$$

erzeugt genau $L \cdot L'$.