Lösungen für die zusätzlichen Übungsaufgaben für die Vorlesung "Berechenbarkeit und Komplexität"

Die folgenden Aufgaben sind ein Angebot für zusätzliche Vorbereitung von der Vorlesung und für die Prüfung "Berechenbarkeit und Komplexität". Sie sind kein offizielles Lernmittel von den Mitarbeitern des Lehrstuhls i1 oder von Prof. W. Thomas. Die Aufgaben haben in keiner Weise etwas mit dem Übungssystem oder der Prüfungsklausur zu tun. Es ist nicht gewährleistet, dass die genutzten Begriffe und Definitionen mit den Vorlesungsinhalten übereinstimmen. Ebenso besteht kein Anspruch auf Korrektheit oder Vollständigkeit der Lösungen für die Aufgaben. Insbesondere ist sämtlicher Inhalt dieses Dokuments nicht repräsentativ für die Inhalte der Klausuraufgaben oder der Bewertung dieser.

Die Aufgaben sind gedacht, um das eigene Wissen zu überprüfen und das Verständnis der Vorlesungsinhalte zu festigen. Im Vergleich zu typischen Aufgaben, die in einer Klausur vorkommen, sind die meisten Aufgaben hier zeitaufwändiger.
Besonders schwere Aufgaben, die über das Übungsniveau hinausgehen, sind mit einem Stern (*) markiert.

Verbesserungen, Fehlermeldungen und insbesondere eine Lösung zu Aufgabe 7.a.viii sind erwünscht.

Update 1 (25.02.2015)

- Aufgabe 2.a.i: Fehlerhaften Eintrag in der Tabelle behoben.
- Aufgabe 2.a.ii: Einen Fehler im Lauf behoben.
- Aufgabe 2.c: Fallunterscheidung erweitert.
- Aufgabe 3.a.ii: Beispiel-Exekution eingefügt.
- Aufgabe 6.a: Lösungen für L_4 erweitert.
- Aufgabe 7.a: Fehler behoben, bei dem H_ϵ als nicht aufzählbar bezeichnet wurde.

Update 2 (20.03.2015)

- Aufgabe 1.h: Neu hinzugefügt.
- Aufgabe 7.b.iii: Copy-Paste-Fehler behoben.
- Aufgabe 8.b: Klauseln sind jetzt korrekt als Konjunktionen, nicht mehr als Disjunktionen, beschrieben.

Fehlermeldungen von: Besa Demiri, Stefanie Koß, Hong An Nguyen, Philipp Nolte, Sascha Welten, Lennart Bader

Algorithmen und Turing-Maschinen

Aufgabe 1: Wissensfragen

a) M berechnet f, wenn für jedes Wort $w \in \Sigma^*$ M auf Eingabe w terminiert und dabei die Ausgabe $f(w)$ hat.

b) Eine Turing-Maschine befindet sich zu jedem "Zeitpunkt" in einer Konfiguration. Eine Konfiguration K ist ein Tupel $K \in Q \times \mathbb{Z} \times \{f \mid f : \mathbb{Z} \to \Gamma\}$, das den Zustand der TM, die Position des Lesekopfes und den Inhalt des Bands darstellt.

Die Anfangskonfiguration ist $K_0 = (q_0, 0, f_x)$, wobei für die Eingabe $x \in \Sigma^*$: $f_x(n) = \begin{cases} x_n & \text{falls } 0 \leq n < |x| \\ B & \text{sonst} \end{cases}$.

Eine Endkonfiguration ist $K_F = (\overline{q}, n, f)$, wobei dann die *Ausgabe* der Turing-Maschine das Wort $f(n)f(n+1)\ldots f(n+k)$ ist, sodass $f(n+k+1) \in \Gamma$ und $f(n+i) \in \Sigma$ für alle $0 \leq i \leq k$.

Ein Schritt einer TM M auf einer Konfiguration $K = (q, n, f)$ ist ein Übergang zu einer anderen Konfiguration $K' = (q', n', f')$. Diese wird definiert durch $\delta(q, f(n)) = (q', S, D)$, wobei $f'(x) = \begin{cases} f(x) & \text{falls } x \neq n \\ S & \text{sonst} \end{cases}$ und $n' = \begin{cases} n & \text{falls } D = N \\ n+1 & \text{falls } D = R \\ n-1 & \text{falls } D = L \end{cases}$.

c) Die Bänder einer TM können in Spuren unterteilt sein, da mehrere Spuren nur eine Bezeichnung für den Fall sind, dass Γ Tupel enthält.
Eine Spur kann nicht in Bänder unterteilt werden, aus demselben Grund.

d) Da Σ endlich ist, ist Σ^* abzählbar. Insbesondere existiert eine berechenbare Bijektion $f : \Sigma^* \to \mathbb{N}$.
Man kann also jede Eingabe x durch eine Zahl $f(x)$ eindeutig identifizieren. Insofern ist es möglich, Σ auf ein einziges Element 0 zu beschränken, indem man die Eingabe zu $0^{f(x)}$ transformiert.

e) Die Kodierung einer Turing-Maschine, notiert durch $<M>$, ist eine eindeutige Darstellung der TM als endliche Zeichenkette.
Also folgt, dass die Menge aller Turing-Maschinen $\mathcal{M}$ höchstens so groß ist wie die Menge aller Wörter: $|\mathcal{M}| \leq |\Sigma^*|$. Da Σ^* abzählbar ist, existieren auch nur abzählbar viele TMs.

f)
1. unendlich abzählbar
2. überabzählbar
3. unendlich abzählbar
4. unendlich abzählbar
5. überabzählbar
6. unendlich abzählbar
7. unendlich abzählbar

g) Aus der Vorlesung ist bekannt, dass TMs, RAM- und WHILE-Programme gleich mächtig sind. TMs, die nur gerade Bandpositionen verwenden sind, sind ebenfalls gleich mächtig zu diesen Modellen, da eine Konstruktion einer solchen Einschränkung aus einer normalen TM möglich ist. Zum Beispiel kann die Zustandsmenge verdreifacht werden, wobei die neuen Zustände dafür genutzt werden, dass die Kopfposition sich immer um zwei Schritte bewegt.

LOOP-Programme sind bekanntermaßen schwächer als die anderen Modelle, z.B. aufgrund der Ackermann-Funktion. DEAs/DFAs sind das schwächste Modell, da sie durch LOOP-Programme und damit auch durch die anderen simuliert werden können. DFAs können auf das Alphabet $\{0, 1\}$ eingeschränkt werden. Betrachtet man dann das Eingabewort w als Binärkodierung einer Zahl n, kann diese als Eingabe an das LOOP-Programm gegeben werden. Der DFA terminiert in $\log_2(n)$ Schritten, was durch eine LOOP-Schleife realisierbar ist.

h) Ein DFA lässt sich fast direkt in eine TM übertragen, indem der Kopf in jedem Schritt nach rechts bewegt wird. Der einzige Unterschied ist das Akzeptanzverhalten, das bei der TM nicht über Endzustände geregelt ist.
Sei $A = (Q, \Sigma, q_0, \delta, F)$ ein DFA. Wir definieren die TM $M = (Q', \Sigma, \Gamma, B, q_0, \overline{q}, \delta')$ mit $Q' = Q \cup \{\overline{q}\}$, $\Gamma = \Sigma \cup \{B\}$ und δ' wie folgt:
Sei $q \in Q$. Für alle $a \in \Sigma$, setze $\delta'(q, a) = (\delta(q, a), a, R)$. Setze dann noch $\delta'(q, B) = \begin{cases} (\overline{q}, 0, N) & \text{falls } q \notin F \\ (\overline{q}, 1, N) & \text{sonst} \end{cases}$.

Aufgabe 2: Turing-Maschinen

a)

i)

δ	0	1	#	a	b	B
q_0	q_1^0, B, R	q_1^1, B, R	$q_4, \#, N$	q_0, a, L	q_0, b, L	q_0, B, R
q_1^0	$q_1^0, 0, R$	$q_1^0, 1, R$	$q_2^0, \#, R$	q_2^0, a, R	q_2^0, b, R	q_3, B, L
q_1^1	$q_1^1, 0, R$	$q_1^1, 1, R$	$q_2^1, \#, R$	q_2^1, a, R	q_2^1, b, R	q_3, B, L
q_2^0	q_3, a, N	q_3, b, N	$q_2^0, \#, R$	q_2^0, a, R	q_2^0, b, R	$\overline{q}, 0, N$
q_2^1	q_3, b, N	q_3, a, N	$q_2^1, \#, R$	q_2^1, a, R	q_2^1, b, R	$\overline{q}, 0, N$
q_3	$q_3, 0, L$	$q_3, 1, L$	$q_3, \#, L$	q_3, a, L	q_3, b, L	q_0, B, R
q_4	$q_4, 0, R$	$q_4, 1, R$	$q_4, \#, R$	$q_4, 0, R$	$q_4, 1, R$	q_5, B, L
q_5	$q_5, 0, L$	$q_5, 1, L$	$\overline{q}, \#, R$	$q_5, 0, L$	$q_5, 1, L$	q_5, B, R

Der Fehler liegt in Eintrag $\delta(q_4, B)$. Die Bewegungsrichtung R würde verhindern, dass die TM terminiert.
Die leeren Einträge lauten $\delta(q_2^0, 1) = (q_3, b, N), \delta(q_2^1, 1) = (q_3, a, N), \delta(q_3, \#) = (q_3, \#, L)$.

ii)

$q_0010\#110 \vdash Bq_1^010\#110 \vdash B1q_1^00\#110 \vdash B10q_1^0\#110 \vdash B10\#q_2^0110 \vdash B10\#q_3b10 \vdash B10q_3\#b10 \vdash B1q_30\#b10 \vdash Bq_310\#b10 \vdash q_3B10\#b10 \vdash Bq_010\#b10$

$q_0\#baa \vdash q_4\#baa \vdash \#q_4baa \vdash \#1q_4aa \vdash \#10q_4a \vdash \#100q_4B \vdash \#10q_50 \vdash \#1q_500 \vdash \#q_5100 \vdash q_5\#100 \vdash \#\overline{q}100$

iii)

δ	0	1	#	a	b	B
q_0	q_1^0, B, R	q_1^1, B, R	$q_4, \#, N$	-	-	-
q_1^0	$q_1^0, 0, R$	$q_1^0, 1, R$	$q_2^0, \#, R$	-	-	-
q_1^1	$q_1^1, 0, R$	$q_1^1, 1, R$	$q_2^1, \#, R$	-	-	-
q_2^0	q_3, a, N	q_3, b, N	-	q_2^0, a, R	q_2^0, b, R	-
q_2^1	q_3, b, N	q_3, a, N	-	q_2^1, a, R	q_2^1, b, R	-
q_3	$q_3, 0, L$	$q_3, 1, L$	$q_3, \#, L$	q_3, a, L	q_3, b, L	q_0, B, R
q_4	-	-	$q_4, \#, R$	$q_4, 0, R$	$q_4, 1, R$	q_5, B, L
q_5	$q_5, 0, L$	$q_5, 1, L$	$\overline{q}, \#, R$	-	-	-

b)

i) $M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, \overline{q}\}, \{0, 1, \#, \$\}, \{0, 1, B, \#, \$\}, q_0, B, \overline{q}, \delta)$

δ	0	1	B	#	\$
q_0	$q_1, 0, L$	$q_1, 1, L$	$\overline{q}, B, N$	-	-
q_1	-	-	$q_2, \$, R$	-	-
q_2	$q_2, 0, R$	$q_2, 1, R$	$q_3, \$, L$	-	-
q_3	$q_3, 0, L$	$q_3, 1, L$	-	-	$q_4, \$, R$
q_4	q_5, B, R	q_5, B, R	-	-	$q_6, \$, L$
q_5	$q_4, \#, R$	$q_4, \#, R$	-	-	$q_6, \$, L$
q_6	-	-	q_6, B, L	$q_6, \#, L$	$\overline{q}, \$, N$

ii) $L = 0^*1^+ = \{w \in \Sigma^* \mid w \text{ enthält mindestens eine 1 und nicht das Infix } 10\}$

iii) $q_0 \triangleq 1, \overline{q} \triangleq 2, q_1 \triangleq 3$

$a_1 = 0, a_2 = 1, a_3 = B$

$\text{code}(q_0, 0, q_0, 0, R) = 0101010100$
$\text{code}(q_0, 1, q_1, 1, R) = 01001000100100$
$\text{code}(q_0, B, \overline{q}, 0, N) = 01000100101000$
$\text{code}(q_1, 0, \overline{q}, 0, N) = 00010100101000$
$\text{code}(q_1, 1, q_1, 1, R) = 0001001000100100$
$\text{code}(q_1, B, \overline{q}, 1, N) = 00010001001001000$

$<M>= 111\,0101010100\,11\,01001000100100\,11\,01000100101000$
$11\,00010100101000\,11\,0001001000100100\,11\,00010001001001000\,111$

c)

Turing-Maschinen, die auf mehrdimensionalen Bändern arbeiten, sind gleichmächtig zu den üblichen TMs. Eine mögliche Konstruktion: Die TM benutzt zwei Bänder, von deinen eins das zweidimensionale Band enthält, während das andere eine die Bandposition des ersten Kopfes zählt und eine Injektion von $\mathbb{Z} \times \mathbb{Z}$ und $\mathbb{Z}$ berechnet, um das zweidimensionale Band abbilden zu können. Eine Möglichkeit für eine solche Injektion ist

$$\pi : \mathbb{Z} \times \mathbb{Z} \xrightarrow{\sim} \mathbb{Z}, (x, y) \mapsto \begin{cases} 0 & \text{falls } x = y = 0 \\ g(x, y) + 4 \cdot f(|x|, |y|) & \text{sonst} \end{cases}, \text{ wobei } f(x, y) = x + \sum_{i=2}^{x+y} i \text{ und}$$

$$g(x, y) = \begin{cases} 1 & \text{falls } x \geq 0, y \geq 0 \\ 2 & \text{falls } x \leq 0, y \geq 0 \\ 3 & \text{falls } x \geq 0, y \leq 0 \\ 4 & \text{falls } x \leq 0, y \leq 0 \end{cases}.$$

Da π nur aus simplen Fallunterscheidungen und arithmetischen Operationen besteht, ist es natürlich berechenbar.

Aufgabe 3: Kodierungen

a)

i)

Eine mögliche Kodierung ist Präfix-Notation:

- Falls $\varphi = X_i \in \text{VAR}$, $\text{code}(\varphi) = \#\#\text{bin}(i)$
- Falls $\varphi = \neg\psi$, $\text{code}(\varphi) = \#0\text{code}(\psi)$
- Falls $\varphi = \psi \wedge \theta$, $\text{code}(\varphi) = \#1\text{code}(\psi)\text{code}(\theta)$

Eine Raute # notiert den Anfang eines neuen Zeichens, gefolgt von einem Zeichen, das definiert, welche Art von Formel folgt (Variable, Negation, Konjunktion).

Beispiel: $\varphi = X_0 \wedge \neg(X_1 \wedge X_2)$

$$
\begin{aligned}
&\text{code}(\varphi)\\
=&\#1\text{code}(X_0)\text{code}(\neg(X_1 \wedge X_2))\\
=&\#1\#\#\text{bin}(0)\text{code}(\neg(X_1) \wedge X_2))\\
=&\#1\#\#0\text{code}(\neg(X_1 \wedge X_2))\\
=&\#1\#\#0\#0\text{code}(X_1 \wedge X_2)\\
=&\#1\#\#0\#0\#1\text{code}(X_1)\text{code}(X_2)\\
=&\#1\#\#0\#0\#1\#\#\text{bin}(1)\#\#\text{bin}(2)\\
=&\#1\#\#0\#0\#1\#\#1\#\#10
\end{aligned}
$$

Mit jedem Induktionsschritt kommt eine konstante Anzahl von Symbolen (2) hinzu. Die Formeln, die nur Variablen enthalten, werden als die Indizes dieser Variablen kodiert, welche durch n beschränkt sind. Also gilt $|\text{code}(\varphi)| = \mathcal{O}(|\varphi| \cdot (n+2))$. Unter der Annahme, dass n konstant ist: $|\text{code}(\varphi)| = \mathcal{O}(|\varphi|)$.

ii)

1. Füge eine zweite und dritte Spur mit B gefüllt zum Band hinzu.
2. Kopiere die Kodierung der Interpretation auf die zweite Spur an den Anfang der Eingabe.
3. Durchlaufe die kodierte Formel von links nach rechts. Nutze dabei die dritte Spur als "Stack", um die Induktionstiefe der Formel zu merken. Wird eine Variable erreicht, überprüfe den Wert der Interpretation und schreibe diesen auf den Stack. Werte dann nach Möglichkeit aus.

Ein Beispiel dieses Algorithmus für die Formel $\varphi = (\neg X) \wedge (\neg(Y \wedge Z))$ und die Interpretation $\mathcal{I} : X \mapsto 0, Y \mapsto 0, Z \mapsto 1$, könnte wie folgt aussehen. Dabei ist die linke Spalte das "Band", was noch gelesen werden muss, und die rechte Spalte der "Stack".

$\wedge\neg X\neg \wedge YZ$		(φ in Präfix-Notation)
$\neg X\neg \wedge YZ$	$\wedge$	
$X\neg \wedge YZ$	$\wedge, \neg$	
$\neg \wedge YZ$	$\wedge, \neg, X$	(An dieser Stelle kann ausgewertet werden)
$\neg \wedge YZ$	$\wedge, \neg, 0$	(Einsetzen von $\mathcal{I}(X)$ für X)
$\neg \wedge YZ$	$\wedge, 1$	(Anwenden der Negation)
$\neg \wedge YZ$		($1 \wedge x \equiv x$, also "löschen" sich $\wedge$ und 1 gegenseitig)
$\wedge YZ$	$\neg$	
YZ	$\neg, \wedge$	
Z	$\neg, \wedge, Y$	
Z	$\neg, \wedge, 0$	(Einsetzen für Y)
	$\neg, \wedge, 0, Z$	
	$\neg, \wedge, 0, 1$	
	$\neg, 0$	
	1	

Die Formel wurde komplett gelesen und das letzte Symbol auf dem Stack ist eine 1. Somit erfüllt die Interpretation die Formel.

Unter der Annahme, dass n konstant ist, benötigt Schritt 1 eine Zeit von $\mathcal{O}(|\text{code}(\varphi)|)$ und Schritt 2 $\mathcal{O}(1)$.
Schritt 3 benötigt $|\text{code}(\varphi)|$ Schritte zum Durchlaufen der Formel, wobei nach jedem Zeichen der Stack bearbeitet werden muss, was im Worst Case ebenfalls $|\text{code}(\varphi)|$ Schritte benötigt.
Insgesamt ist die Laufzeit damit $\mathcal{O}(|\text{code}(\varphi)|^2)$, was nach den Überlegungen in i) $\mathcal{O}(|\varphi|^2)$ entspricht.

b)

i)

code = 1110000101000000
$|\text{code}(G)| \in \mathcal{O}(|V|^2)$
$|\text{code}(G)| \in \Omega(|V|^2)$
Es werden immer genau $|V|^2$ viele Bits angegeben, unabhängig von den Kanten.

ii)

code = 001#010#011##100##010##
$|\text{code}(G)| \in \mathcal{O}(|V|^2 \cdot \log(|V|))$
$|\text{code}(G)| \in \Omega(|V|)$
Bei einem vollständigen Graph müssen $|V|^2$ viele Kanten angegeben werden, wobei jede als Binärkodierung etwa $\log_2(|V|)$ viele Symbole benötigt. Bei einem leeren Graphen besteht die Kodierung nur aus Rauten.

iii)

code = 100#001#001#001#010#001#011#010#100#011#010
$|\text{code}(G)| \in \mathcal{O}(|V|^2 \cdot \log(|V|))$
$|\text{code}(G)| \in \Omega(\log(|V|))$
Bei einem vollständigen Graph müssen $|V|^2$ viele Kanten angegeben werden, wobei jede als Binärkodierung etwa $2\log_2(|V|)$ viele Symbole benötigt. Bei einem leeren Graphen besteht die Kodierung nur aus der Binärkodierung der Anzahl der Knoten.

Aufgabe 4: Alternative Rechenmodelle

a)

RAM

1. CLOAD 0
2. STORE 2
3. MULT 2
4. STORE 3
5. LOAD 1
6. SUB 3
7. IF $c(0) = 0$ GOTO 12
8. LOAD 2
9. CADD 1
10. STORE 2
11. GOTO 3
12. LOAD 2
13. STORE 1
14. END

WHILE-Programm

$x_2 := 0;$
$x_4 := x_1;$
WHILE $x_4 \neq 0$ DO
    $x_2 := x_2 + 1;$
    $x_4 := x_2;$
    $x_3 := 0;$
    WHILE $x_4 \neq 0$ DO
        $x_5 := 0;$
        WHILE $x_2 \neq 0$ DO
            $x_3 := x_3 + 1;$
            $x_5 := x_5 + 1;$
            $x_2 := x_2 - 1$
        ENDWHILE;
        $x_2 := x_5;$
        $x_4 := x_4 - 1$
    ENDWHILE;
    $x_4 := x_1;$
    WHILE $x_3 \neq 0$ DO
        $x_4 := x_4 - 1;$
        $x_3 := x_3 - 1$
    ENDWHILE
ENDWHILE;
$x_1 := x_2$

LOOP-Programm

$x_2 := 0;$
$x_4 := x_1;$
LOOP x_1 DO
    IF $x_4 \neq 0$ THEN
        $x_2 := x_2 + 1;$
        $x_3 := 0;$
        LOOP x_2 DO
            LOOP x_2 DO
                $x_3 := x_3 + 1$
            ENDLOOP
        ENDLOOP;
        $x_4 := x_1;$
        LOOP x_3 DO
            $x_4 := x_4 - 1$
        ENDLOOP
    ELSE $x_1 := x_1$ ENDIF
ENDLOOP;
$x_1 := x_2$

Berechenbarkeit

Aufgabe 5: Wissensfragen und Allgemeines

a)

Eine Sprache L ist

- entscheidbar, wenn eine Turing-Maschine M existiert, sodass M auf jeder Eingabe w hält und genau dann akzeptiert, d.h. ein Wort ausgibt, das mit 1 beginnt, wenn $w \in L$.
- semi-entscheidbar, wenn eine Turing-Maschine M existiert, sodass M jede Eingabe $w \in L$ akzeptiert und auf allen anderen Eingaben nicht terminiert.
- aufzählbar, wenn eine Turing-Maschine M existiert, sodass M L aufzählt. Das heißt, dass M nicht terminiert, wenn der Endzustand erreicht wird, sondern weiterläuft. Wenn M das i-te Mal den Endzustand erreicht, ist die Ausgabe der Konfiguration w_i. Die Aufzählung von M ist dann die Folge $w_1, w_2, w_3, \ldots$.
- co-aufzählbar, wenn $\overline{L} = \Sigma^* \setminus L$ aufzählbar ist. Alternativ ist L "co-semi-entscheidbar", was äquivalent zu co-aufzählbar ist, wenn eine TM M existiert, sodass M alle Wörter $w \notin L$ verwirft und auf allen anderen Eingaben nicht hält.

b)

Für ein Alphabet Σ ist die Menge der "Domino-Steine" $D = \Sigma^+ \times \Sigma^+$.
PKP $= \{K \subseteq D \mid$
$K = \{(x_1, y_1), \ldots, (x_n, y_n)\},$ es existiert eine Folge von Indizes $i_1, \ldots, i_k$, sodass $x_{i_1} \ldots x_{i_k} = y_{i_1} \ldots y_{i_k}\}$
MPKP $= \{K \in D^+ \mid$
$K = ((x_1, y_1), \ldots, (x_n, y_n)),$ es existiert eine Folge von Indizes $i_2, \ldots, i_k$, sodass $x_1 x_{i_2} \ldots x_{i_k} = y_1 y_{i_2} \ldots y_{i_k}\}$

c)

Entscheidbar: SAT, $\emptyset$
Aufzählbar: H, H_ϵ, SAT, PKP, $\emptyset$
Co-Aufzählbar: SAT, $\emptyset$

d)

Beispiele:
$L = \{<M> \mid M$ terminiert auf keiner Eingabe$\}$
$\overline{H_\epsilon}$

e)

$\{w \in 0^* \mid w = 0^n, n \in \mathbb{N},$ es existiert eine TM M, sodass $<M> = \text{bin}(n)$ und $<M> \in H_\epsilon\}$

f)

Entscheidbar: LOOP-Programme, Kellerautomaten (Stack automata), Endliche Automaten (DFA)
Unentscheidbar: WHILE-Programme, RAM-Programme, Turing-Maschinen, alle Turing-vollständigen Systeme

g)

Sei A ein Aufzähler für L. Definiere die TM M, die für eine Eingabe x so lange A simuliert, bis x aufgezählt wird. Dann semi-entscheidet M L.

Sei M eine TM, die L semi-entscheidet. Definiere einen Aufzähler A für L wie folgt:

1. $i := 1$
2. Berechne die Wörter $w_1, \ldots, w_i$, wobei w_j das j-te Wort kanonischer Reihenfolge ist.
3. Simuliere M für genau i Schritte auf allen $w_1, \ldots, w_i$ und berechne so die Menge $L_i = \{w_j \in \{w_1, \ldots, w_i\} \mid M \text{ hält auf } w_j \text{ in höchstens } i \text{ Schritten}\}$.
4. Gib alle Wörter $w \in L_i$ aus.
5. Erhöhe i um 1.
6. Wiederholung ab Schritt 2.

h)

i) Das präsentierte Modell der NTMs ist *nicht* mächtiger als normale DTMs, denn jede NTM kann durch eine DTM simuliert werden.

ii) Das präsentierte Modell der NTMs ist *nicht* mächtiger als normale DTMs. Betrachtet man den Lauf einer DTM auf einer Eingabe als Folge von Konfigurationen, so kann man alle Läufe einer NTM auf einer Eingabe als Baum interpretieren. Dabei ist die Wurzel die Ausgangskonfiguration und die Nachfolger eines Knotens sind alle möglichen Nachfolge-Konfigurationen. Dann kann eine DTM die NTM simulieren, indem sie nach dem Muster einer Breitensuche die Knoten abläuft. Das heißt, die DTM nutzt einen zusätzlichen Speicher, in dem sie sich den Pfad in dem Baum der NTM von der Wurzel zum aktuellen Knoten merkt. Nach jedem Schritt nutzt sie Backtracking, um den nächsten Knoten zu simulieren. Die DTM akzeptiert dann entsprechend, wenn sie ein akzeptierendes Blatt findet, und verwirft, wenn sie alle Blätter abgearbeitet hat, aber kein akzeptierendes dabei war.

Wir zeigen nun die Korrektheit dieser Konstruktion. Sei $w \in \Sigma^*$ ein Wort, das von der NTM akzeptiert wird. Die Anzahl der Optionen, zwischen denen in jedem Schritt gewählt werden können, ist endlich, also hat jeder Knoten in dem Baum nur endlich viele Nachfolger und auf jeder Tiefe des Baumes liegen nur endlich viele Knoten. Das heißt, die DTM wird mit ihrer Simulation jede Tiefe irgendwann erreichen. Da die NTM w akzeptiert, muss sie dies auf einem Lauf nach n Schritten tun. Das heißt, die DTM findet bei Baumtiefe n ein akzeptierendes Blatt und akzeptiert damit w.

Sei $w \in \Sigma^*$ ein Wort, das von der NTM verworfen wird. Das bedeutet, das alle Läufe von der NTM auf w verwerfend sind. Insbesondere terminieren also alle Läufe auf w und somit können in dem Baum keine unendlichen Pfade existieren. Nach dem Lemma von König (siehe Hinweis) folgt, dass es ein $n \in \mathbb{N}$ gibt, sodass alle Pfade in dem Baum höchstens die Länge n haben. Das heißt, dass alle Läufe der NTM auf w nach höchstens n Schritten terminieren. Damit wird die DTM auch irgendwann alle Läufe abgearbeitet und somit alle Blätter des Baumes gefunden haben. Da alle Blätter verwerfende Konfigurationen repräsentieren, verwirft die DTM damit w.

i)

	entscheidbar	semi-entscheidbar
$L \subseteq L_1$	Nein	Nein
$L \supseteq L_1$	Nein	Nein
$L = L_1 \cap L_2$	Ja	Ja
$L = L_1 \cup L_2$	Ja	Ja
$L = \Sigma^* \setminus L_1$	Ja	Nein
$L_1 \leq L$	Nein	Nein
$L \leq L_1$	Ja	Ja
$L = L_1 \cdot L_2$	Ja	Ja
$L = L_1^*$	Ja	Ja
$L = L_1 \Delta L_2$	Ja	Nein

- $\boldsymbol{L \subseteq L_1, L \supseteq L_1}$
  Σ^* ist entscheidbar und aufzählbar, aber $H_{\text{total}} \subseteq \Sigma^*$ ist weder das eine, noch das andere. Das gleiche gilt für $\emptyset$.

- $\boldsymbol{L = L_1 \cap L_2, L = L_1 \cup L_2}$
  Die Turing-Maschinen für L_1 und L_2 können "parallel" ausgeführt werden, d.h. z.B. auf zwei unterschiedlichen Bändern. Dann ist eine Eingabe im Schnitt / in der Vereinigung genau dann, wenn beide / mindestens eine der beiden Maschinen das Wort akzeptiert.

- $\boldsymbol{L = \Sigma^* \setminus L_1}$
  Entscheidet eine TM das Komplement von L, so kann daraus eine TM für L konstruiert werden, indem das Ergebnis (accept/reject) "umgedreht" wird.
  Für aufzählbare Sprachen gilt dies nicht. Wenn L_1 aufzählbar ist, dann ist L co-aufzählbar. Wäre L dann auch aufzählbar, so wäre L und damit L_1 entscheidbar.

- $\boldsymbol{L_1 \leq L}$
  H_ϵ ist entscheidbar und damit aufzählbar, aber $H_\epsilon \leq H_{\text{total}}$ ist weder das eine, noch das andere.

- $\boldsymbol{L \leq L_1}$
  Aus einer Reduktion kann man einen Widerspruchsbeweis für die Entscheidbarkeit / Aufzählbarkeit einer Sprache folgern, siehe z.B. Aufgabe 7.b.i

- $\boldsymbol{L = L_1 \cdot L_2}$
  Wenn L_1 und L_2 von M_1 und M_2 entschieden werden, kann L durch eine Turing-Maschine entschieden werden, die "rät", wie die Aufteilung einer Eingabe ist. Das heißt, für eine Eingabe $w \in \Sigma^*$ verhält sich die TM wie folgt:

  1. $i := 1$
  2. Setze $V := \{(u, v) \in \Sigma^* \times \Sigma^* \mid uv = w\}$.
  3. Für alle $(u, v) \in V$, simuliere M_1 auf u und M_2 auf v für i Schritte.
  4. Wenn ein Paar von Worten von beiden TMs akzeptiert wird, akzeptiere w. Wenn alle Paare verworfen werden, verwirf w.
  5. Erhöhe i um 1.

  Die Menge V in Schritt 2 ist endlich (genauer: $|V| = |w| + 1$). Daraus und aus der Tatsache, dass M_i L_i entscheidet, folgt, dass die beschriebene TM immer terminiert.

  Analog ist die Begründung für semi-entscheidbare Sprachen.

- $\boldsymbol{L = L_1^*}$
  Eine Konstruktion einer TM, die L (semi-)entscheidet ist ähnlich zu der vorherigen Teilaufgabe. Es muss eine andere Menge V betrachtet werden, die aber trotzdem endlich ist.
  $V = \{v_1 \dots v_n \in (\Sigma^+)^* \mid v_1 \dots v_n = w\}$
  $|V| = \begin{cases} 1 & \text{falls } w = \epsilon \\ 2^{|w|-1} & \text{sonst} \end{cases}$

- $\boldsymbol{L = L_1 \Delta L_2}$
  Da entscheidbare Sprachen unter Schnitt, Vereinigung und Komplementbildung abgeschlossen sind, und da $L_1 \Delta L_2 = (L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2)$, ist L entscheidbar.
  Für aufzählbare Sprachen gilt dies nicht, z.B. für $L_1 = H_\epsilon$, $L_2 = H_{\text{total}}$. Dann ist
  $L = \{<M> \mid M \text{ hält auf Eingabe } \epsilon \text{ aber nicht auf allen Eingaben}\}$ nicht aufzählbar, was z.B. durch die Reduktion $H_{\text{total}} \leq L$ gezeigt werden kann.

j)

Angenommen, L wird durch M entschieden. Man kann einen einfachen Widerspruch folgern:
$(<M>)^{-1} \in L \quad \Leftrightarrow \quad M$ verwirft $(<M>)^{-1} \quad \Leftrightarrow \quad (<M>)^{-1} \notin L(M) = L$
Somit kann L also nicht entscheidbar sein.

Aufgabe 6: Satz von Rice

Sprache	L_1	$L_{2,n}$	L_3	L_4	L_5	L_6	L_7	L_8
Entscheidbarkeit	n	RE	RE	n	R	R	unbekannt	RE
Satz von Rice	Ja	Ja	Nein	Ja	Nein	Nein	Nein	Ja

(R = entscheidbar, RE = unentscheidbar, aber aufzählbar, n = nicht entscheidbar oder aufzählbar)

Für alle Sprachen wird die Unentscheidbarkeit, wenn möglich, mit dem Satz von Rice in zwei Varianten gezeigt. Sowohl die Formulierung von Prof. Thomas im WS 14/15 als auch die von Prof. Vöcking in früheren Semestern. Sei dafür im folgenden immer $\mathcal{R} = \{f : \Sigma^* \to \Sigma^* \cup \{\perp\} \mid f \text{ ist berechenbar}\}$.

- $\boldsymbol{L_1}$
  Satz von Rice, Version 1
  Sei $S = \{f \in \mathcal{R} \mid \forall w \in \Sigma^* : \exists v \in \Sigma^* : f(w) = 1v\}$. Für $f_0 : x \mapsto 0$ und $f_1 : x \mapsto 1$ gilt offensichtlich $f_0 \notin S, f_1 \in S$, also $\emptyset \subset S \subset \mathcal{R}$.
  Außerdem ist nach Definition von L_1 für alle TMs M: $<M> \in L_1$ gdw. $f_M \in S$. Also ist L_1 nach dem Satz von Rice unentscheidbar.

  Satz von Rice, Version 2
  $L_1 = \{<M> \mid M \text{ erfüllt die Eigenschaft } E\}$ für $E(M) = M$ akzeptiert alle Eingaben. E ist eine nicht-triviale Eigenschaft, denn es existieren die TMs M_0, die alle Eingaben verwirft und damit E nicht erfüllt, und M_1, die alle Eingaben akzeptiert und damit E erfüllt.
  Seien M_1, M_2 Turing-Maschinen, sodass M_1 E erfüllt und $f_{M_1} = f_{M_2}$. Angenommen, M_2 erfüllt E nicht. Dann existiert ein $w \in \Sigma^*$, sodass M_2 w nicht akzeptiert, also $f_{M_2}(w) = f_{M_1}(w) \neq 1$. Damit akzeptiert M_1 w nicht und erfüllt somit nicht E. Aus diesem Widerspruch folgt, dass E eine semantische Eigenschaft ist.
  Es folgt also nach dem Satz von Rice, dass L_1 unentscheidbar ist.

- $\boldsymbol{L_{2,n}}$
  Satz von Rice, Version 1
  Sei $S = \{f \in \mathcal{R} \mid \forall w \in \Sigma^n : \exists v \in \Sigma^* : f(w) = 1v\}$. Für $f_0 : x \mapsto 0$ und $f_1 : x \mapsto 1$ gilt offensichtlich $f_0 \notin S, f_1 \in S$, also $\emptyset \subset S \subset \mathcal{R}$.
  Außerdem ist nach Definition von $L_{2,n}$ für alle TMs M: $<M> \in L_{2,n}$ gdw. $f_M \in S$. Also ist $L_{2,n}$ nach dem Satz von Rice unentscheidbar.

Satz von Rice, Version 2
$L_{2,n} = \{<M>|\ M$ erfüllt die Eigenschaft $E_n\}$ für $E_n(M) = M$ akzeptiert alle Eingaben der Länge n. E_n ist eine nicht-triviale Eigenschaft, denn es existieren die TMs M_0, die alle Eingaben verwirft und damit E nicht erfüllt, und M_1, die alle Eingaben akzeptiert und damit E erfüllt.
Seien M_1, M_2 Turing-Maschinen, sodass M_1 E_n erfüllt und $f_{M_1} = f_{M_2}$. Angenommen, M_2 erfüllt E_n nicht. Dann existiert ein $w \in \Sigma^n$, sodass M_2 w nicht akzeptiert, also $f_{M_2}(w) = f_{M_1}(w) \neq 1$. Damit akzeptiert M_1 w nicht und erfüllt somit nicht E_n. Aus diesem Widerspruch folgt, dass E_n eine semantische Eigenschaft ist.
Es folgt also nach dem Satz von Rice, dass $L_{2,n}$ unentscheidbar ist.

Algorithmus
$L_{2,n}$ ist semi-entscheidbar durch einen Algorithmus, der alle $|\Sigma|^n$ vielen Wörter der Länge n durchgeht und M auf ihnen simuliert. Wenn $<M> \in L_{2,n}$, dann terminiert dieses Vorgehen in jedem Fall. Wenn dann alle Wörter von M akzeptiert wurden, kann $<M>$ akzeptiert werden.

- ***L_3***
  Satz von Rice
  Der Satz von Rice ist für diese Sprache nicht anwendbar, da ihre Elemente nicht die Form $<M>$ haben.

  Algorithmus L_3 kann semi-entschieden werden durch den (fast) gleichen Algorithmus wie L_2, mit dem Unterschied, dass n keine Konstante ist, sondern aus der Eingabe gelesen werden muss.

- ***L_4***
  Das Besuchen des Endzustandes ist äquivalent zur Terminierung der TM, also kann die Sprache auch anders formuliert werden, als $L_4 = \{<M>|\ \forall w \in \Sigma^* : M$ hält auf w nicht $\rightarrow M$ akzeptiert $w\}$.
  Damit die Implikation wahr wird, muss die Antezendens (linke Seite) falsch werden, da eine TM, die nicht hält, auch nicht akzeptieren kann. Somit gilt $L_4 = \{<M>|\ M$ hält auf allen Eingaben$\} = H_{\text{total}}$.

  Satz von Rice, Version 1
  Sei $S = \{f \in \mathcal{R} \mid \forall w \in \Sigma^* : f(w) \neq \perp\}$. Für $f_\perp : x \mapsto \perp$ und $f_1 : x \mapsto 1$ gilt offensichtlich $f_0 \notin S, f_1 \in S$, also $\emptyset \subset S \subset \mathcal{R}$.
  Außerdem ist nach Definition von L_4 für alle TMs M: $<M> \in L_4$ gdw. $f_M \in S$. Also ist L_4 nach dem Satz von Rice unentscheidbar.

  Satz von Rice, Version 2
  $L_4 = \{<M>|\ M$ erfüllt die Eigenschaft $E\}$ für $E(M) = M$ hält auf allen Eingaben. E ist eine nicht-triviale Eigenschaft, denn es existieren die TMs M_0, die auf allen Eingaben nie terminiert und damit E nicht erfüllt, und M_1, die alle Eingaben akzeptiert und damit E erfüllt.
  Seien M_1, M_2 Turing-Maschinen, sodass M_1 E erfüllt und $f_{M_1} = f_{M_2}$. Angenommen, M_2 erfüllt E nicht. Dann existiert ein $w \in \Sigma^*$, sodass M_2 auf w nicht hält, also $f_{M_2}(w) = f_{M_1}(w) = \perp$. Damit hält M_1 auf w nicht und erfüllt somit nicht E. Aus diesem Widerspruch folgt, dass E eine semantische Eigenschaft ist.
  Es folgt also nach dem Satz von Rice, dass L_4 unentscheidbar ist.

- ***L_5***
  Da jede TM in Zustand q_0 beginnt, ist $L_5 = \mathcal{M} = \{<M>|\ M$ ist eine Turing-Maschine$\}$.
  Satz von Rice, Version 1
  Der Satz von Rice kann für L_5 nicht angewandt werden, da die zu L_5 "äquivalente" Menge $S = \{f \in \mathcal{R} \mid f$ ist eine berechenbare Funktion$\}$ gleich zur Menge aller berechenbare Funktionen $\mathcal{R}$ ist.

  Satz von Rice, Version 2
  Der Satz von Rice kann für L_5 nicht angewandt werden, da für die Eigenschaft E mit $L_5 = \{<M>|$

M erfüllt Eigenschaft $E\}$ E trivial ist, denn es existiert keine TM, die E nicht erfüllt. (*Hinweis*: Die Eigenschaft ist semantisch!)

Algorithmus
Ein Algorithmus der L_5 entscheidet, kann jede Eingabe akzeptieren, bzw. ggf. noch einen Syntax-Check durchführen, ob die Eingabe die Kodierung einer Turing-Maschine ist.

- $\boldsymbol{L_6}$
  Satz von Rice, Version 1
  Der Satz von Rice kann für L_5 nicht angewandt werden, da es keine Menge $S \subseteq \mathcal{R}$ gibt, sodass für alle Turing-Maschinen M gilt, dass $f_M \in S$ gdw. $<M> \in L_5$.

  Satz von Rice, Version 2
  Der Satz von Rice kann für L_5 nicht angewandt werden, da für die Eigenschaft E mit $L_5 = \{<M>|$ M erfüllt Eigenschaft $E\}$ E nicht semantisch ist. Es existieren TMs, die die gleiche Funktion berechnen, von denen eine E erfüllt, die andere aber nicht.

  Algorithmus
  Ein Algorithmus, der L_5 entscheidet, kann die Eingabe-TM M auf ϵ simulieren, bis jeder Zustand einmal erreicht wurde oder bis ein Zustand mehr als einmal erreicht wurde, und dann das entsprechende Resultat ausgeben.

- $\boldsymbol{L_7}$
  Der Satz von Rice gilt für diese Sprache nicht, aus demselben Grund wie bei L_6. Der Beweis für die Unentscheidbarkeit wird in Aufgabe 7.a.viii genauer ausgeführt.

- $\boldsymbol{L_8}$
  Die umständliche Formulierung von L_8 lässt sich umformen zu
  $L_8 = \{<M>|$ M gibt für die Eingabe ϵ das Wort 1 aus$\}$.

  Satz von Rice, Version 1
  Sei $S = \{f \in \mathcal{R} \mid f(\epsilon) = 1\}$. Für $f_0 : x \mapsto 0$ und $f_1 : x \mapsto 1$ gilt offensichtlich $f_0 \notin S, f_1 \in S$, also $\emptyset \subset S \subset \mathcal{R}$.
  Außerdem ist nach Definition von L_8 für alle TMs M: $<M> \in L_8$ gdw. $f_M \in S$. Also ist $L_8 1$ nach dem Satz von Rice unentscheidbar.

  Satz von Rice, Version 2
  $L_8 = \{<M>|$ M erfüllt die Eigenschaft $E\}$ für $E(M) = M$ gibt für die Eingabe ϵ das Wort 1 aus. E ist eine nicht-triviale Eigenschaft, denn es existieren die TMs M_0, die alle Eingaben verwirft und damit E nicht erfüllt, und M_1, die alle Eingaben akzeptiert und damit E erfüllt.
  Seien M_1, M_2 Turing-Maschinen, sodass M_1 E erfüllt und $f_{M_1} = f_{M_2}$. Dann gilt $f_{M_1}(\epsilon) = f_{M_2}(\epsilon) = 1$, also erfüllt M_2 E. Daraus folgt, dass E eine semantische Eigenschaft ist.
  Es folgt also nach dem Satz von Rice, dass L_8 unentscheidbar ist.

  Algorithmus
  Ein Algorithmus, der L_8 semi-entscheidet, kann einfach die Eingabe-TM M auf ϵ simulieren und die Eingabe akzeptieren, wenn M ϵ akzeptiert.

b)

Seien S, w und L wie beschrieben. Wir zeigen die Reduktion $\overline{H_\epsilon} \leq L$. Daraus folgt dann, dass L nicht aufzählbar sein kann.

Definiere $f(<M>) = <M^*(M)>$, wobei $M^*(M)$ eine TM ist, die sich auf der Eingabe x wie folgt verhält: Falls $x \neq w$, simuliere M auf x. Falls $x = w$, simuliere M auf ϵ. Offensichtlich ist f berechenbar, es bleibt also die Korrektheit der Reduktion zu zeigen.

$<M> \in \overline{H_\epsilon}$
$\Rightarrow M$ hält nicht auf ϵ
$\Rightarrow M^*(M)$ hält nicht auf w
$\Rightarrow f_{M^*(M)}(w) = \perp$
$\Rightarrow <M^*(M)> \in L$

$<M> \notin \overline{H_\epsilon}$
$\Rightarrow <M> \in H_\epsilon$
$\Rightarrow M$ hält auf ϵ
$\Rightarrow M^*(M)$ hält auf w
$\Rightarrow f_{M^*(M)}(w) \neq \perp$
$\Rightarrow <M^*(M)> \notin L$

□

Aufgabe 7: Unterprogramme und Reduktion

a)

Die folgenden Aufgaben werden alle durch Reduktion gelöst. Mit Hilfe von Teilaufgabe b) können daraus ggf. Beweise über Unterprogrammtechnik konstruiert werden.

i) Beh.: $H_{\text{total}} \leq L_1$. Da H_{total} nicht aufzählbar ist, folgt daraus, dass L_1 nicht aufzählbar ist.

Definiere dafür $f(<M>) = <M^*>$, wobei M^* sich auf Eingabe x wie folgt verhält: Falls x nicht mit einer 1 endet, simuliere M auf x. Sonst $x = v1$ für ein $v \in \Sigma^*$. Dann simuliere M auf v und akzeptiere, falls M hält. f ist offenbar berechenbar, es bleibt also die Korrektheit zu beweisen.

Korrektheit

$<M> \in H_{\text{total}}$
$\Leftrightarrow M$ hält auf allen Eingaben
$\Leftrightarrow \forall w \in \Sigma^* : M$ hält auf w
$\Leftrightarrow \forall w \in \Sigma^* : M^*$ hält auf $w1$
$\Leftrightarrow \forall w \in \Sigma^* : M^*$ akzeptiert $w1$
$\Leftrightarrow M^*$ akzeptiert alle Eingaben, die mit 1 enden
$\Leftrightarrow <M^*> \in L_1$

ii) Fall 1: $\perp \in$ Bild(g)

Sei für diesen Beweis $w \in \Sigma^*$ ein Wort, sodass $g(w) = \perp$. Beh.: $\overline{H_\epsilon} \leq L_{2,g}$. Da $\overline{H_\epsilon}$ nicht aufzählbar ist, folgt daraus, dass $L_{2,g}$ nicht aufzählbar ist.

Definiere dafür $f(<M>) = <M^*>$, wobei M^* sich auf Eingabe x wie folgt verhält: Falls $x = w$, simuliere M auf ϵ. Sonst berechne $g(x)$. f ist offenbar berechenbar, es bleibt also die Korrektheit zu beweisen.

Korrektheit

$<M> \in \overline{H_\epsilon}$
$\Leftrightarrow M$ hält nicht auf ϵ
$\Leftrightarrow \forall v \in \Sigma^* \setminus \{w\} : M^*$ gibt auf v $g(v)$ aus und hält auf w nicht
$\Leftrightarrow \forall v \in \Sigma^* : M^*$ gibt auf v $g(v)$ aus
$\Leftrightarrow M^*$ berechnet g
$\Leftrightarrow <M^*> \in L_{2,g}$

Fall 2: $\perp \notin$ Bild(g)
Beh.: $H_{\text{total}} \leq L_{2,g}$. Da H_{total} nicht aufzählbar ist, folgt daraus, dass $L_{2,g}$ nicht aufzählbar ist.

Definiere dafür $f(<M>) = <M^*>$, wobei M^* sich auf Eingabe x wie folgt verhält: Simuliere M auf x und gib danach $f(x)$ aus. f ist offenbar berechenbar, es bleibt also die Korrektheit zu beweisen.

Korrektheit

$<M> \in H_{\text{total}}$
$\Leftrightarrow M$ hält auf allen Eingaben
$\Leftrightarrow \forall v \in \Sigma^* : M$ hält auf v
$\Leftrightarrow \forall v \in \Sigma^* : M^*$ gibt auf v $g(v)$ aus
$\Leftrightarrow M^*$ berechnet g
$\Leftrightarrow <M^*> \in L_{2,g}$

iii) Beh.: $L_{2,g} \leq L_3$ für $g : <M> \mapsto \begin{cases} 1 & \text{falls } <M> \in H_\epsilon \\ \perp & \text{sonst} \end{cases}$. Da $L_{2,g}$ nicht aufzählbar ist, folgt daraus, dass L_3 nicht aufzählbar ist.

Definiere dafür $f(<M>) = <M^*>$, wobei M^* sich auf Eingabe x wie folgt verhält: Simuliere M auf x. Falls M dann terminiert, die Ausgabe aber nicht 1 ist, gehe in eine Endlosschleife über. (*Nach der Definition dieser Vorlesung steht jede Ausgabe, die mit einer 1 beginnt für Akzeptanz. Definiert man stattdessen nur die 1 als akzeptierende Ausgabe, ist die Reduktion einfacher.*)
f ist offenbar berechenbar, es bleibt also die Korrektheit zu beweisen.

Korrektheit

$<M> \in L_{2,g}$
$\Rightarrow M$ berechnet g
$\Rightarrow \forall w \in \Sigma^* : M$ gibt auf w 1 aus, falls $w \in H_\epsilon$ und hält sonst nicht
$\Rightarrow \forall w \in \Sigma^* : M^*$ gibt auf w 1 aus, falls $w \in H_\epsilon$ und hält sonst nicht
$\Rightarrow M^*$ semi-entscheidet H_ϵ
$\Rightarrow <M^*> \in L_3$

$<M^*> \in L_3$

$\Rightarrow M^*$ semi-entscheidet H_ϵ

$\Rightarrow \forall w \in \Sigma^* : M^*$ gibt auf w ein Wort $v = 1u$ aus, falls $w \in H_\epsilon$ und hält sonst nicht

$\Rightarrow \forall w \in \Sigma^* : M^*$ gibt auf w 1 aus, falls $w \in H_\epsilon$ und hält sonst nicht

$\Rightarrow \forall w \in \Sigma^* : M$ gibt auf w 1 aus, falls $w \in H_\epsilon$ und hält sonst nicht

$\Rightarrow M$ berechnet g

$\Rightarrow <M> \in L_{2,g}$

Anmerkung: Die Folgerung von der dritten zur vierten Zeile gilt, da nach Konstruktion von M^* Bild$(f_{M^*}) = \{1, \perp\}$.

iv) Beh.: $H \leq L_4$. Da H nicht entscheidbar ist, folgt daraus, dass L_4 nicht entscheidbar ist.

Definiere dafür $f(<M>\#x) = <M^*>\#x\#1$, wobei M^* sich auf Eingabe w wie folgt verhält: Simuliere M auf w und gib danach 1 aus. f ist offenbar berechenbar, es bleibt also die Korrektheit zu beweisen.

Korrektheit

$<M>\#x \in H$

$\Leftrightarrow M$ hält auf Eingabe x

$\Leftrightarrow M^*$ hält auf Eingabe x

$\Leftrightarrow M^*$ gibt auf Eingabe x 1 aus

$\Leftrightarrow <M^*>\#x\#1 \in L_4$

v) Beh.: $\overline{H_\epsilon} \leq L_5$. Da $\overline{H_\epsilon}$ nicht aufzählbar ist, folgt daraus, dass L_5 nicht aufzählbar ist.

Definiere dafür $f(<M>) = <M^*>$, wobei M^* sich auf Eingabe x wie folgt verhält: Simuliere M auf ϵ und akzeptiere danach. f ist offenbar berechenbar, es bleibt also die Korrektheit zu beweisen.

Korrektheit

$<M> \in \overline{H_\epsilon}$

$\Rightarrow M$ hält nicht auf ϵ

$\Rightarrow M^*$ hält auf keiner Eingabe

$\Rightarrow L(M^*) = \emptyset$

$\Rightarrow L(M^*)$ ist endlich

$\Rightarrow <M^*> \in L_5$

$<M^*> \in L_5$

$\Rightarrow L(M^*)$ ist endlich

$\Rightarrow$ es existieren unendlich viele $w \in \Sigma^*$ sodass $w \notin L(M^*)$

$\Rightarrow$ es existiert ein $w \in \Sigma^*$ sodass $w \notin L(M^*)$

$\Rightarrow$ es existiert ein $w \in \Sigma^*$ sodass M^* auf w nicht hält

$\Rightarrow M$ hält nicht auf ϵ

$\Rightarrow <M> \in \overline{H_\epsilon}$

vi) Beh.: $H_\epsilon \leq L_6$. Da H_ϵ nicht entscheidbar ist, folgt daraus, dass L_6 nicht entscheidbar ist.

Definiere dafür $f(<M>) =<M^*>$, wobei M^* sich auf Eingabe x wie folgt verhält: Simuliere M auf ϵ und verwirf dann. f ist offenbar berechenbar, es bleibt also die Korrektheit zu beweisen.

Korrektheit

$<M>\in H_\epsilon$
$\Leftrightarrow M$ hält auf ϵ
$\Leftrightarrow M^*$ hält auf allen Eingaben
$\Leftrightarrow M^*$ verwirft alle Eingaben
$\Leftrightarrow M^*$ verwirft $<M^*>$
$\Leftrightarrow <M^*>\in L_6$

vii) Beh.: $H_{\text{total}} \leq L_7$. Da H_{total} nicht aufzählbar ist, folgt daraus, dass L_7 nicht aufzählbar ist.

Definiere dafür $f(<M>) = \text{code}(Q', \Sigma, \Gamma', B, \delta')$. Q', Γ' und δ' sind nach einer TM $M' = (Q', \Sigma, \Gamma', q_0, B, , \overline{q}, \delta')$ mit folgendem Verhalten definiert:
$Q' = \{q_0, \overline{q}\}$ und $\Gamma' = \Gamma \cup (\Gamma \times Q)$. M' nutzt eine neue Spur, in der Zustände eingetragen werden, um so M zu simulieren, aber nur einen Zustand (q_0) zu nutzen. Das heißt, M' berechnet die gleiche Funktion wie M.
Weiterhin wird definiert $\delta'(\overline{q}, a) = (\overline{q}, a, N)$, d.h. von $\overline{q}$ aus terminiert eine TM nie.

f ist offenbar berechenbar, es bleibt also die Korrektheit zu beweisen.

Korrektheit

$<M>\in H_{\text{total}}$
$\Leftrightarrow M$ hält auf allen Eingaben
$\Leftrightarrow M'$ hält auf allen Eingaben
$\Leftrightarrow$ es existieren $q_1, q_2 \in Q'$ sodass $(Q', \Sigma, \Gamma', q_1, B, q_2, \delta') \in H_{\text{total}}$
$\Leftrightarrow f(<M>) \in L_7$

Anmerkung: Die $\Leftarrow$-Folgerung von der vierten zur dritten Zeile gilt, da die einzigen Optionen $(q_1, q_2) \in \{(q_0, \overline{q}), (\overline{q}, q_0)$ sind und eine TM mit $\overline{q}$ als Startzustand und δ' als Transitionsfunktion nie terminiert.

viii) Beh.: $\overline{H_\epsilon} \leq L_5$. Da $\overline{H_\epsilon}$ nicht entscheidbar ist, folgt daraus, dass L_5 nicht entscheidbar ist.

Definiere dafür $f(< M >) =< M^* >$, wobei M^* eine Erweiterung von M ist, indem drei Zustände p_0, p_1, p_2 und eine neue Spur, die Elemente aus Q enthält, zum Bandalphabet hinzugefügt werden. Der neue Startzustand ist p_0, der in der Anfangsposition die neue Spur mit dem Symbol q_0 füllt und danach zu p_2 wechselt. Danach verhält sich M^* wie M, wobei aber jeder Schritt, den M ausgeführt hätte, durch mehrere Schritte in M^* simuliert wird.
Für alle $q \in Q$: $\delta^*(q, (a, r)) = (p_1, (a, q), N)$, das heißt, M^* wechselt zu Zustand p_1 und "merkt" sich den Zustand q in der neuen Spur.
$\delta^*(p_1, (a, r)) = (p_2, (a, r), N)$, das heißt, von Zustand p_1 aus wechselt M^* nur zu p_2 ohne die Konfiguration sonst zu ändern.
$\delta^*(p_2, (a, r)) = (q, (b, r), d)$ für $(q, b, d) = \delta(r, a)$, das heißt von p_2 aus wird der ursprüngliche Schritt von dem "gemerkten" Zustand ausgeführt.

Man kann sehen, dass, wenn M auf der Eingabe x $n \in \mathbb{N} \cup \{\infty\}$ Schritte läuft, so werden p_1, p_2 von M^* auf Eingabe x $n, n+1$ Mal besucht. f ist offenbar berechenbar, es bleibt also die Korrektheit zu beweisen.

Korrektheit

$<M> \in H_\epsilon$
$\Rightarrow$ M hält auf ϵ
$\Rightarrow$ M hält auf ϵ nach $n \in \mathbb{N}$ Schritten
$\Rightarrow$ M^* besucht auf ϵ p_1 n Mal und p_2 $n+1$ Mal
$\Rightarrow$ M^* besucht auf ϵ p_2 öfter als alle anderen Zustände
$\Rightarrow$ $<M^*> \in L_8$

$<M> \notin H_\epsilon$
$\Rightarrow$ M hält nicht auf ϵ
$\Rightarrow$ M^* besucht auf ϵ p_1 und p_2 unendlich oft
$\Rightarrow$ M^* besucht p_1 und p_2 mindestens so oft, wie alle anderen Zustände
$\Rightarrow$ Es existiert kein Zustand q den M^* auf ϵ öfter besucht, als alle anderen
$\Rightarrow$ $<M^*> \notin L_8$

Ob L_8 aufzählbar ist oder nicht, konnte ich bisher nicht lösen :(

b)

i) Angenommen L_2 ist entscheidbar durch eine TM M_2. Daraus können wir eine TM M_1 konstruieren, die L_1 entscheidet. Dafür berechnet M_1 für eine Eingabe x zunächst $y = f(x)$ und simuliert dann M_2 auf y. Nach der Voraussetzung folgt sofort:
$w \in L_1 \quad \Leftrightarrow \quad f(w) \in L_2 \quad \Leftrightarrow \quad M_2$ akzeptiert $f(w) \quad \Leftrightarrow \quad M_1$ akzeptiert w
Also entscheidet M_1 die Sprache L_1. Aus diesem Widerspruch folgt, dass M_2 nicht existieren kann.

ii) Angenommen, die TM M_L entscheidet L. Daraus konstruieren wir eine TM M_ϵ, die H_ϵ entscheidet. Dafür verhält sich M_ϵ auf Eingabe $<M>$ wie folgt:

1. Konstruiere eine TM M_1, die sofort hält.
2. Konstruiere eine TM M_2, die nach einem Schritt hält.
3. Simuliere M_L auf $<M>\#<M_1>$ und $<M>\#<M_2>$.
4. Wenn M_L beide Wörter akzeptiert, verwerfe $<M>$. Ansonsten akzeptiere.

$<M> \in H_\epsilon$
$\Rightarrow$ M hält auf ϵ nach n Schritten
$\Rightarrow$ $n \neq 1$ oder $n \neq 2$
$\Rightarrow$ $<M>\#<M_1> \notin L$ oder $<M>\#<M_2> \notin L$
$\Rightarrow$ M_ϵ akzeptiert $<M>$

$<M> \notin H_\epsilon$

$\Rightarrow M$ hält auf ϵ nicht

$\Rightarrow$ $<M>\#<M_1> \in L$ und $<M>\#<M_2> \in L$

$\Rightarrow M_\epsilon$ verwirft $<M>$

Also entscheidet M_ϵ die Sprache H_ϵ. Da H_ϵ aber bekanntermaßen unentscheidbar ist, kann M_L nicht existieren.

iii) Aus der Übung ist bekannt, dass $H_\epsilon \not\leq \overline{H_\epsilon}$, und, dass die Reduktions-Relation $\leq$ transitiv ist. Wir zeigen, dass $L \leq \overline{H_\epsilon}$. Dann kann nicht $H_\epsilon \leq L$ sein.

Definiere dafür $f(<M_1>\#<M_2>) = <M^*>$, wobei M^* eine TM ist, die sich auf Eingabe x wie folgt verhält:

1. Simuliere M_1 auf x und zähle dabei n_1, die Anzahl der Schritte, die M_1 benötigt.
2. Simuliere M_2 auf x und zähle dabei n_2, die Anzahl der Schritte, die M_2 benötigt.
3. Falls $n_1 = n_2$, gehe über in eine Endlosschleife. Ansonsten akzeptiere die Eingabe.

Offenbar ist f berechenbar, es bleibt also noch die Korrektheit der Reduktion zu beweisen.

$<M_1>\#<M_2> \notin L$

$\Rightarrow M_1, M_2$ halten auf ϵ nach n_1, n_2 Schritten und $n_1 \neq n_2$

$\Rightarrow M^*$ akzeptiert ϵ

$\Rightarrow M^*$ hält auf ϵ

$\Rightarrow$ $<M^*> \notin \overline{H_\epsilon}$

Für die andere Beweisrichtung wird eine Fallunterscheidung benötigt.
Sei $<M_1>\#<M_2> \in L$.

Fall 1:

M_1 hält nicht auf ϵ

$\Rightarrow M^*$ hält nicht auf ϵ

$\Rightarrow$ $<M^*> \in \overline{H_\epsilon}$

Fall 2:

M_2 hält nicht auf ϵ

$\Rightarrow M^*$ hält nicht auf ϵ

$\Rightarrow$ $<M^*> \in \overline{H_\epsilon}$

Fall 3:

M_1 und M_2 halten beide auf ϵ nach genau $n \in \mathbb{N}$ Schritten

$\Rightarrow M^*$ geht in Schritt 3 der Definition in eine Endlosschleife

$\Rightarrow M^*$ hält nicht auf ϵ

$\Rightarrow$ $<M^*> \in \overline{H_\epsilon}$

Komplexität

Aufgabe 8: Wissensfragen und Allgemeines

a) Die Laufzeit einer DTM ist die maximale Laufzeit, die auf einer Eingabe der Länge n benötigt wird, d.h. $t(n) = \max_{w \in \Sigma^n} t(w)$.

Die Laufzeit einer NTM ist analog definiert, wobei $t(w)$ die Länge des *kürzesten* Laufs ist, um w zu akzeptieren.

b)

> 3-SAT
> **Eingabe:** Eine aussagenlogische Formel der Form $\varphi = \bigvee_{i=1}^{n} C_i$, wobei $C_i = x_i \wedge y_i \wedge z_i$ für Literale x_i, y_i, z_i.
> **Ausgabe:** Existiert eine Belegung der Variablen, sodass φ erfüllt wird?

c) `P` ist die Klasse aller Sprachen, die durch eine Turing-Maschine entscheidbar sind, die die Laufzeit $\mathcal{O}(n^k)$ für ein $k \in \mathbb{N}$ hat.
`NP` ist die Klasse aller Sprachen, die durch eine nicht-det. Turing-Maschine entscheidbar sind, die die Laufzeit $\mathcal{O}(n^k)$ für ein $k \in \mathbb{N}$ hat.
Ein Problem ist NP-schwer, wenn für alle Probleme $L \in$ `NP` gilt, dass L auf dieses Problem in Polynomialzeit reduzierbar ist.
Ein Problem ist NP-vollständig, wenn es NP-schwer ist und in `NP` liegt.

d) L_1 kann nicht NP-schwer und damit nicht NP-vollständig sein, da sonst `P` = `NP`.
L_2 kann beides sein, da `NP` $\subseteq$ `EXPTIME`. Allerdings ist dies nicht notwendig, da auch `P` $\subseteq$ `EXPTIME`.

e) Da über Graphisomorhpism nicht bekannt ist, ob das Problem NP-vollständig ist, würde ein solcher Algorithmus keine direkten Folgen haben. (außer solche, die über das Wissen der Vorlesung hinausgehen)
Da 2-SAT bekanntermaßen in `P` liegt, hätte ein solcher Algorithmus keinen Einfluss auf die Komplexitätshierarchie.
3-SAT ist NP-vollständig, insofern würde die Existent eines solchen Algorithmus die Aussage `P` = `NP` beweisen.

f) Ein Polynomialzeit-Verifizierer für eine Sprache L ist eine DTM, die als Eingabe eine Instanz x von L und ein "Zertifikat" y erhält, wobei y polynomial in der Länge von x beschränkt sein muss. Der Verifizierer akzeptiert eine Eingabe $x\#y$ in Polynomialzeit, wenn $x \in L$.

g) Üblicherweise werden die Klassen bzgl. Turing-Maschinen definiert. Allerdings lassen sich diverse andere Modelle, wie z.B. RAM, in Polynomialzeit mit Turing-Maschinen simulieren, weshalb das genaue Modell nicht von Relevanz ist.

h) * Wir zeigen zunächst die Aussage `NP` $\subseteq$ `PSPACE`.
Dafür reicht es, sich die Simulation einer NTM durch eine DTM zu betrachten. Wir nehmen der Einfachheit halber an, dass in jedem Schritt der NTM eine Wahl zwischen genau zwei Alternativen besteht. Eine solche NTM lässt sich aus jeder anderen mit konstantem Zeitverlust konstruieren. Wenn die NTM dann in $p(n)$ Zeit läuft, so werden auch $p(n)$ Entscheidungen "getroffen". Betrachtet man die Läufe als Entscheidungsbaum heißt das, dass alle Blätter höchstens einen Pfad der Länge $p(n)$ von der Wurzel entfernt sind. Das heißt, dass eine Zeichenkette der Länge $p(n)$ über $\{0, 1\}$ ausreicht, um

jedes Blatt eindeutig zu identifizieren. Insofern reicht auch zusätzlicher Speicher von $\mathcal{O}(p(n))$ aus, um die NTM durch eine DTM simulieren zu lassen.

Für die Aussage $\texttt{PSPACE} \subseteq \texttt{EXPTIME}$ nutzen wir eine Aussage aus Übungsblatt 2: Wenn eine TM nur $s(n)$ Speicher verwendet, so hält sie nach maximal $(|Q| - 1) \cdot |\Gamma|^{s(n)} \cdot s(n) + 1$ Schritten oder nie.
Wir betrachten nur TMs, die die Probleme entscheiden und polynomiellen Speicher verwenden, also $s(n) = p(n)$ für ein Polynom p. Damit hält jede TM, die ein Problem aus $\texttt{PSPACE}$ entscheidet, nach $\mathcal{O}(|\Gamma|^{p(n)} \cdot p(n)) = \mathcal{O}(2^{\log(|\Gamma|) \cdot 2p(n)})$ Schritten. Somit entscheidet diese TM das Problem auch in exponentieller Arbeitszeit, also $L(M) \in \texttt{EXPTIME}$. □

Aufgabe 9: P und NP

a)

Die deterministische TM für die beiden Sprachen ist fast gleich:

1. Lies das aktuelle Zeichen, überschreibe es mit _ und merke es im Zustand.
2. Gehe nach rechts bis zur ersten #. Gehe weiter nach rechts bis zum ersten nicht-_-Symbol.
3. Lies das aktuelle Zeichen, überschreibe es mit _ und merke es im Zustand.
4. Gehe nach rechts bis zur ersten #. Gehe weiter nach rechts bis zum ersten nicht-_-Symbol.
5. Falls ein B gelesen wird, akzeptiere die Eingabe.
6. Lies das aktuelle Zeichen und vergleiche es mit dem XOR-Ergebnis der beiden gemerkten Zeichen. Falls es nicht übereinstimmt, verwirf die Eingabe. Ansonsten überschreibe es mit _.
7. Gehe nach links bis zur ersten #. Gehe weiter nach links bis zur nächsten #. Gehe weiter nach links bis zum ersten _. Gehe dann einen Schritt nach rechts.
8. Wiederholung ab Schritt 1.

L_1 kann nicht durch eine NTM in Linearzeit entschieden werden. Dies ist ersichtlich, da die Eigenschaft der Wörter durch eine universelle ("für alle...") Eigenschaft formuliert ist. Die NTM kann also nicht "raten", um Rechenzeit zu sparen.

L_2 hingegen ist in Linearzeit durch eine NTM entscheidbar. Die negierte Eigenschaft von L_1 ist existentiell ("es existiert ein..."). Die NTM kann also die fehlerhafte Position i im Wort "raten" und muss dann nur noch verifizieren, dass $x_i \oplus y_i \neq z_i$.

b)

i)

- Eingabe: $x = (n, v, V, w, W)$
- Eingabekodierung: $\text{code}(x) = \text{bin}(n)\#\text{code}(v)\#\text{code}(w)\#\text{bin}(V)\#\text{bin}(W)$, wobei $\text{code}(f) = \text{bin}(f(1))\# \ldots \#\text{bin}(f(n))$
- Eingabelänge: $|\text{code}(x)| = \mathcal{O}(\log(n) + n \cdot \log(\max \text{Bild}(v)) + n \cdot \log(\max \text{Bild}(w)) + \log(V) + \log(W)) = \mathcal{O}(\log(n) + n \cdot \log(V) + n \cdot \log(W) + \log(V) + \log(W)) = \mathcal{O}(n \cdot (\log(V) + \log(W)))$
- Zertifikat: $y = (m)$, wobei m die Anzahl-Funktion ist.
- Zertifikat-Kodierung: $\text{code}(y) = \text{code}(m)$
- Zertifikat-Länge: $|\text{code}(y)| = |\text{code}(m)| = \mathcal{O}(n \cdot \log(n)) = \mathcal{O}(n \cdot \log(n)) = \mathcal{O}(|\text{code}(x)|^2)$
- Verifizierer:

1. Prüfe, ob $\sum_{i=1}^{n} w(i)m(i) \leq W$ und verwirf sonst.
2. Prüfe, ob $\sum_{i=1}^{n} v(i)m(i) \geq V$ und verwirft sonst ($\mathcal{O}(n \cdot \log(n) \cdot \log(\max \text{Bild}(v)))$).
3. Akzeptiere.

- Verifizierer Laufzeit:
  1. $\mathcal{O}(n \cdot \log(n) \cdot \log(\max \text{Bild}(w))) = \mathcal{O}(|\text{code}(x)|^2)$
  2. $\mathcal{O}(n \cdot \log(n) \cdot \log(\max \text{Bild}(v))) = \mathcal{O}(|\text{code}(x)|^2)$
  3. $\mathcal{O}(1)$

ii)

- Eingabe: $x = (s, M)$, wobei $M = \{m_1, \ldots, m_n\}$
- Eingabekodierung: $\text{code}(x) = \text{bin}(s)\#\text{bin}(m_1)\# \ldots \#\text{bin}(m_n)$
- Eingabelänge: $|\text{code}(x)| = \mathcal{O}(\log(|s|) + n \cdot \log(\max_{m_i \in M} |m_i|))$
- Zertifikat: $y = (M')$, wobei $M' = \{m'_1, \ldots, m'_k\}$. M' ist die beschriebene Menge, deren Summe s ergibt.
- Zertifikat-Kodierung: $\text{code}(y) = \text{bin}(m'_1)\# \ldots \#\text{bin}(m'_k)$
- Zertifikat-Länge: $|\text{code}(y)| = \mathcal{O}(k \cdot \log(\max_{m'_i \in M'} |m_i|)) = \mathcal{O}(n \cdot \log(\max_{m_i \in M} |m_i|)) = \mathcal{O}(|\text{code}(x)|)$
- Verifizierer:
  1. Berechne $\sum_{i=1}^{k} m'_i = s'$.
  2. Prüfe, ob $s = s'$. Wenn nicht, verwirf. Sonst akzeptiere.
- Verifizierer Laufzeit:
  1. $\mathcal{O}(k \cdot \log(\max_{m'_i \in M'} |m_i|)) = \mathcal{O}(n \cdot \log(\max_{m_i \in M} |m_i|)) = \mathcal{O}(|\text{code}(x)|)$
  2. $\mathcal{O}(\log(s)) = \mathcal{O}(|\text{code}(x)|)$

iii)

- Eingabe: $x = (n, R)$, wobei $R = \{(a_1, b_1, c_1), \ldots, (a_m, b_m, c_m)\}$
- Eingabekodierung: $\text{code}(x) = \text{bin}(n)\#\text{bin}(a_1)\#\text{bin}(b_1)\#\text{bin}(c_1)\# \ldots \#\text{bin}(a_m)\#\text{bin}(b_m)\#\text{bin}(c_m)$
- Eingabelänge: $|\text{code}(x)| = \mathcal{O}(\log(n) + m \cdot \log(n)) = \mathcal{O}(m \cdot \log(n))$
- Zertifikat: $y = (c_1, \ldots, c_k)$ eine zyklische Ordnung, die die Voraussetzungen von R erfüllt, wobei aber $k < n$ sein kann. Es müssen alle Elemente, die in R in einem Tripel vorkommen, auch in y enthalten sein.
- Zertifikat-Kodierung: $\text{code}(y) = \text{bin}(c_1)\# \ldots \#\text{bin}(c_k)$
- Zertifikat-Länge: $|\text{code}(y)| = \mathcal{O}(k \cdot \log(n)) = \mathcal{O}(m \cdot \log(n)) = \mathcal{O}(|\text{code}(x)|)$
- Verifizierer:

1. Für alle $(c_i, c_j, c_l) \in R$, prüfe: $i = l$ oder ($i < l$ und $j < i$) oder ($i < l$ und $k < j$) oder ($l < i$ und $i < j < l$). Wenn nicht, dann verwirf. Ansonsten akzeptiere.

- Verifizierer Laufzeit:

1. $\mathcal{O}(m \cdot k \cdot \log(n)) = \mathcal{O}(m^2 \cdot \log(n)) = \mathcal{O}(|\text{code}(x)|^2)$

Aufgabe 10: Polynomielle Reduktion

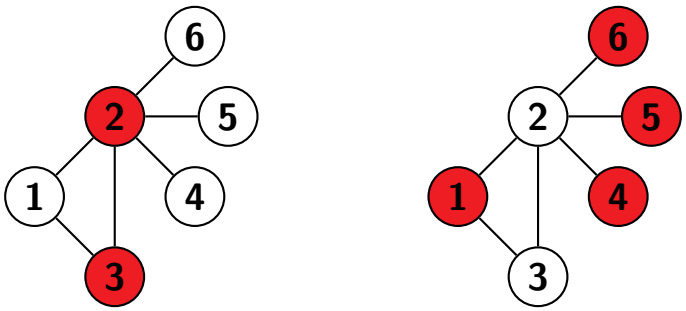
a)

Reduktion
Wir zeigen, dass IndependentSet $\leq_P$ VertexCover. Sei dafür $(G, k) = ((V, E), k)$ eine Instanz von IndependentSet. Wir definieren die Transformation als $(G, k) \mapsto (G, |V| - k)$.

Sei $(G, k) \in$ VertexCover. Dann existiert also ein Vertex Cover $C \subseteq V$ der Größe $|C| = k$. Behauptung: $V \setminus C$ ist ein Independent Set auf G (und hat offensichtlich die richtige Größe).
Angenommen es existieren $u, v \in V \setminus C$ mit $\{u, v\} \in E$. Da $u, v \notin C$, würde eine Kante existieren, die kein Element des Vertex Cover enthält. Dieser Widerspruch zeigt, dass solche u, v nicht existieren können, also ist $V \setminus C$ ein Independent Set.

Sei $(G, |V| - k) \in$ IndependentSet. Dann existiert also ein Independent Set $I \subseteq V$ der Größe $|I| = |V| - k$. Behaptung: $V \setminus I$ ist ein Vertex Cover auf G.
Angenommen, es existiert eine Kante $\{u, v\} \in E$, sodass beide Knoten u, v nicht in $V \setminus I$ enthalten sind. Dann müssen $u, v \in I$ sein, also existiert eine Kante, die zwei Knoten aus I miteinander verbindet, was ein Widerspruch ist. Also kann eine solche Kante nicht existieren und $V \setminus I$ ist ein Vertex Cover. □

Beispiel
Der Beispiel-Graph hat ein Vertex Cover der Größe 2 (links) und ebenso ein Independent Set der Größe 4 (rechts). Allerdings existiert kein Cover der Größe 1 und auch kein Independent Set der Größe 5.



b)

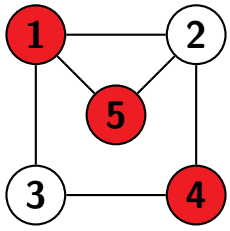
Wir zeigen, dass VertexCover $\leq_P$ SetCover. Sei dafür $(G, k) = ((V, E), k)$ eine Instanz von VertexCover. Wir definieren die Transformation als $(G, k) \mapsto (E, \mathcal{S}, k)$, wobei $\mathcal{S} = \{S_v \mid v \in V\}$ und $S_v = \{e \in E | v \in e\}$.

Sei $(G, k) \in$ VertexCover. Dann existiert also ein Vertex Cover $C \subseteq V$ der Größe $|C| = k$. Behauptung: $\{S_v \mid v \in C\}$ ist ein Set Cover von $\mathcal{S}$ (und hat offensichtlich die richtige Größe).
Angenommen, es existiert ein $e \in U = E$, sodass $e \notin \bigcup\{S_v \mid v \in C\}$. Das heißt, dass für alle Kanten $v \in C$,

e nicht in S_v enthalten ist, was nach Definition bedeutet, dass e nicht zu v inzident ist. Das widerspricht aber der Aussage, dass C ein Vertex Cover von G ist. Also kann so eine Kante e nicht existieren.

Sei $(E, \mathcal{S}, k) \in$ SetCover. Dann existiert ein Set Cover $C \subseteq \mathcal{S}$ der Größe $|S| = k$. Behauptung: $\{v \in V \mid S_v \in C\}$ ist ein Vertex Cover von G.
Angenommen, es existiert ein $e \in E$, sodass für alle Knoten $v \in \{v \in V \mid S_v \in C\}$ gilt, dass e nicht zu v inzident ist. Dann ist $e \notin S_v$ für alle solche v, also $e \notin \bigcup C$. Damit ist C kein Set Cover. Aus diesem Widerspruch folgt, dass eine solche Kante e nicht existieren kann. □

Beispiel
Der Graph enthält ein Vertex Cover der Größe 3, aber keins der Größe 2.



Die Transformation ergibt $(E, \mathcal{S}, k)$ mit $\mathcal{S} = \{S_1, S_2, S_3, S_4, S_5\} =$
$\{\{e_{1,2}, e_{1,3}, e_{1,5}\}, \{e_{1,2}, e_{2,4}, e_{2,5}\}, \{e_{1,3}, e_{3,4}\}, \{e_{2,4}, e_{3,4}\}, \{e_{1,5}, e_{2,5}\}\}$, dabei steht $e_{i,j}$ für die Kante $\{i, j\}$.

Es existiert ein Set Cover der Größe 3 ($C = \{S_1, S_4, S_5\} = \{\{e_{1,2}, e_{1,3}, e_{1,5}\}, \{e_{2,4}, e_{3,4}\}, \{e_{1,5}, e_{2,5}\}$), aber keins der Größe 2.

c)

Wir zeigen, dass Knapsack $\leq_P$ GeneralizedAssignment. Sei dafür (n, v, w, V, W) eine Instanz von Knapsack. Wir definieren die Transformation als $(n, v, w, V, W) \mapsto (X, B, s, v', w', V)$, wobei $X = \{1, \ldots, n\}$, $B = \{1\}$, $s : 1 \mapsto W$, $v' : (x, 1) \mapsto v(x)$, $w' : (x, 1) \mapsto w(x)$.

Sei $(n, v, w, V, W) \in$ Knapsack. Also existiert eine Anzahlfunktion $m : \{1, \ldots, n\} \to \mathbb{N}$, die die Bedingungen erfüllt. Nach Konstruktion ist dann $c : (x, 1) \mapsto m(x)$ eine Anzahlfunktion für die GeneralizedAssignment-Instanz, die die Bedingungen erfüllt.
Analog folgt, dass, wenn keine Anzahlfunktion für (n, v, w, V, W) existiert, auch keine für (X, B, s, v', w', V) existieren kann. □

Beispiel:
$X = \{1, 2, 3\}, B = \{1\}$
$v' : (1, 1) \mapsto 1, (2, 1) \mapsto 3, (3, 1) \mapsto 6$
$w' : (1, 1) \mapsto 2, (2, 1) \mapsto 4, (3, 1) \mapsto 5$
$s_1 : 1 \mapsto 10, s_2 : 1 \mapsto 9$
$V_1 = 10, V_2 = 11$
Für den ersten Fall hat die KP-Instanz die Lösung $m : 1 \mapsto 0, 2 \mapsto 0, 3 \mapsto 2$ und ebenso die GA-Instanz die Lösung $c : (1, 1) \mapsto 0, (2, 1) \mapsto 0, (3, 1) \mapsto 2$.

Für den zweiten Fall haben weder die KP- noch die GA-Instanz eine Lösung.

d)[2]

Wir zeigen, dass 3-SAT $\leq_P$ MAX2-SAT. Sei dafür $\varphi = \bigwedge_{i=1}^{n} C_i$ mit $C_i = x_i \vee y_i \vee z_i$ eine Instanz von 3-SAT. Wir definieren die Transformation als $\varphi \mapsto (\psi = \bigwedge_{i=1}^{n} D_i, 7n)$. Dafür definieren wir $D_i = x_i \wedge y_i \wedge z_i \wedge w_i \wedge (\overline{x_i} \vee \overline{y_i}) \wedge (\overline{x_i} \vee \overline{z_i}) \wedge (\overline{y_i} \vee \overline{z_i}) \wedge (x_i \vee \overline{w_i}) \wedge (y_i \vee \overline{w_i}) \wedge (z_i \vee \overline{w_i})$, wobei w_i eine neu eingeführte Variable für jede Klausel ist.

Bevor wir die Korrektheit der Konstruktion beweisen, zeigen wir eine Aussage über die konstruierten D_i. Dafür sei $\mathcal{I}$ eine Belegung der Variablen. Wir definieren $|C_i(\mathcal{I})|$ als die Anzahl der Literale, die in C_i vorkommen (x_i, y_i, z_i), die von $\mathcal{I}$ erfüllt werden.

Behauptung: Sei $1 \leq i \leq n$ beliebig und sei $\mathcal{I}$ eine Belegung der Variablen von φ. Erweitert man nun $\mathcal{I}$ um eine Definition für w_i, so gilt:

- Unabhängig von der Wahl von $\mathcal{I}(w_i)$ werden in D_i maximal 7 Klauseln durch $\mathcal{I}$ erfüllt.
- Es existiert eine Wahl von $\mathcal{I}(w_i)$, sodass in D_i genau 7 Klauseln erfüllt werden, genau dann, wenn $\mathcal{I}$ C_i erfüllt.

Für den Beweis dieser Behauptung werden wir alle Möglichkeiten für $|C_i(\mathcal{I})|$ und $\mathcal{I}(w_i)$ durchgehen und die erfüllten Klauseln in D_i zählen.

- $|C_i(\mathcal{I})| = 0$ und $\mathcal{I}(w_i) = 0$
  Erfüllte Klauseln in D_i: $\overline{x_i} \vee \overline{y_i}, \overline{x_i} \vee \overline{z_i}, \overline{y_i} \vee \overline{z_i}, x_i \vee \overline{w_i}, y_i \vee \overline{w_i}, z_i \vee \overline{w_i}$
  6 Klauseln in D_i sind erfüllt.
- $|C_i(\mathcal{I})| = 0$ und $\mathcal{I}(w_i) = 1$
  Erfüllte Klauseln in D_i: $w_i, \overline{x_i} \vee \overline{y_i}, \overline{x_i} \vee \overline{z_i}, \overline{y_i} \vee \overline{z_i}$
  4 Klauseln in D_i sind erfüllt.
- $|C_i(\mathcal{I})| = 1$ und $\mathcal{I}(w_i) = 0$
  OBdA nehmen wir an, dass x_i von $\mathcal{I}$ erfüllt wird und y_i und z_i nicht.
  Erfüllte Klauseln in D_i: $x_i, \overline{x_i} \vee \overline{y_i}, \overline{x_i} \vee \overline{z_i}, \overline{y_i} \vee \overline{z_i}, x_i \vee \overline{w_i}, y_i \vee \overline{w_i}, z_i \vee \overline{w_i}$
  7 Klauseln in D_i sind erfüllt.
- $|C_i(\mathcal{I})| = 1$ und $\mathcal{I}(w_i) = 1$
  OBdA nehmen wir an, dass x_i von $\mathcal{I}$ erfüllt wird und y_i und z_i nicht.
  Erfüllte Klauseln in D_i: $x_i, w_i, \overline{x_i} \vee \overline{y_i}, \overline{x_i} \vee \overline{z_i}, \overline{y_i} \vee \overline{z_i}, y_i \vee \overline{w_i}, z_i \vee \overline{w_i}$
  7 Klauseln in D_i sind erfüllt.
- $|C_i(\mathcal{I})| = 2$ und $\mathcal{I}(w_i) = 0$
  OBdA nehmen wir an, dass x_i und y_i von $\mathcal{I}$ erfüllt werden und z_i nicht.
  Erfüllte Klauseln in D_i: $x_i, y_i, \overline{x_i} \vee \overline{z_i}, \overline{y_i} \vee \overline{z_i}, x_i \vee \overline{w_i}, y_i \vee \overline{w_i}, z_i \vee \overline{w_i}$
  7 Klauseln in D_i sind erfüllt.
- $|C_i(\mathcal{I})| = 2$ und $\mathcal{I}(w_i) = 1$
  OBdA nehmen wir an, dass x_i und y_i von $\mathcal{I}$ erfüllt werden und z_i nicht.
  Erfüllte Klauseln in D_i: $x_i, y_i, w_i, \overline{x_i} \vee \overline{z_i}, \overline{y_i} \vee \overline{z_i}, x_i \vee \overline{w_i}, y_i \vee \overline{w_i}$
  7 Klauseln in D_i sind erfüllt.

- $|C_i(\mathcal{I})| = 3$ und $\mathcal{I}(w_i) = 0$
  Erfüllte Klauseln in D_i: $x_i, y_i, z_i, x_i \vee \overline{w_i}, y_i \vee \overline{w_i}, z_i \vee \overline{w_i}$
  6 Klauseln in D_i sind erfüllt.

- $|C_i(\mathcal{I})| = 3$ und $\mathcal{I}(w_i) = 1$
  Erfüllte Klauseln in D_i: $x_i, y_i, z_i, w_i, x_i \vee \overline{w_i}, y_i \vee \overline{w_i}, z_i \vee \overline{w_i}$
  7 Klauseln in D_i sind erfüllt.

Damit ist die Behauptung gezeigt und wir können nun die Korrektheit der Reduktion beweisen.

Sei $\varphi \in$ 3-SAT, also existiert eine Belegung $\mathcal{I}$ der Variablen, sodass jede Klausel in φ erfüllt wird. Dann existiert nach der Behauptung eine Erweiterung von $\mathcal{I}$ bzgl. $w_1, \ldots, w_n$, sodass in jedem D_i 7 Klauseln erfüllt werden. Damit sind in ψ $7n$ Klauseln erfüllt, also $\psi \in$ Max2-SAT.

Sei $\psi \in$ Max2-SAT, also existiert eine Belegung $\mathcal{I}$ der Variablen, sodass $7n$ Klauseln in ψ erfüllt sind. Nach der Behauptung muss damit $\mathcal{I}$ jede Klausel C_i erfüllen, also auch φ. □

Beispiel:
$\varphi_1 = (x \vee x \vee \overline{x})$
$\Rightarrow \psi_1 = x \wedge x \wedge \overline{x} \wedge w \wedge (\overline{x} \vee \overline{x}) \wedge (\overline{x} \vee x) \wedge (\overline{x} \vee x) \wedge (x \vee \overline{w}) \wedge (x \vee \overline{w}) \wedge (\overline{x} \vee \overline{w})$
φ_1 wird erfüllt z.B. durch die Belegung $x \mapsto 1$. Die Belegung $x \mapsto 1, w \mapsto 1$ erfüllt in ψ die benötigten 7 Klauseln.

$\varphi_2 = (x \vee x \vee x) \wedge (\overline{x} \vee \overline{x} \vee \overline{x})$
$\Rightarrow \psi_2 = D_1 \wedge D_2$
$D_1 = x \wedge x \wedge x \wedge w_1 \wedge (\overline{x} \vee \overline{x}) \wedge (\overline{x} \vee \overline{x}) \wedge (\overline{x} \vee \overline{x}) \wedge (x \vee \overline{w_1}) \wedge (x \vee \overline{w_1}) \wedge (x \vee \overline{w_1})$
$D_2 = \overline{x} \wedge \overline{x} \wedge \overline{x} \wedge w_2 \wedge (x \vee x) \wedge (x \vee x) \wedge (x \vee x) \wedge (\overline{x} \vee \overline{w_2}) \wedge (\overline{x} \vee \overline{w_2}) \wedge (\overline{x} \vee \overline{w_2})$
φ_2 hat keine erfüllende Belegung und in ψ_2 können maximal 13 Klauseln gleichzeitig wahr werden.

e) **[3]**

Bevor wir zu der Reduktion kommen, zeigen wir eine allgemeine Aussage über Zeichenketten. Wir beschränken uns dabei auf das Alphabet $\Sigma = \{0, 1\}$ und bezeichnen die Hamming-Distanz mit $d(w_1, w_2)$. Für eine Menge von Wörtern $S \subseteq \Sigma^n$ schreiben wir auch $d(S, w) = \max_{v \in S} d(s, v)$ als die maximale Distanz von w zu einem Wort aus S.

Sei $n \in \mathbb{N}$. Dann definieren wir die Zeichenketten $\gamma_n = (10)^n$ und $\overline{\gamma}_n = (01)^n$, sowie $\beta_n(i) = (10)^i 01 (10)^{n-i-1}$ und $\overline{\beta}_n(i) = (01)^i 10 (01)^{n-i-1}$ als die beiden Wörter, die man erhält, wenn in γ_n bzw. $\overline{\gamma}_n$ das $2i$-te und das $(2i+1)$-te Zeichen vertauscht werden. Weiterhin definieren wir noch die Menge $\mathcal{S}_n = \{\gamma_n, \overline{\gamma}_n, \beta_n(0), \overline{\beta}_n(0), \ldots, \beta_n(n-1), \overline{\beta}_n(n-1)\}$. Zum Beispiel: $\mathcal{S}_3 = \{101010, 010101, 011010, 100101, 100110, 011001, 101001, 010110\}$.

Behauptung: Für alle $n \geq 3$ gilt, dass für ein Wort $w \in \{0,1\}^{2n}$ genau dann $d(\mathcal{S}_n, w) \leq n$ gilt, wenn w die Form $w \in \{00, 11\}^n$ annimmt, d.h. 0 und 1 kommen immer nur in gerade Länge hintereinander vor.

Die Rück-Richtung der Aussage ist trivial, also beschränken wir uns auf die Hin-Richtung. Angenommen, die Aussage ist falsch und es existiert ein w mit $d(\mathcal{S}_n, w) \leq n$, aber es existiert ein k, sodass $w_{2k} \neq w_{2k+1}$. OBdA sei $w_{2k} = 1$, $w_{2k+1} = 0$. Diese beiden Positionen stimmen mit γ_n überein, also müssen sie von $\beta_n(k)$ verschieden sein, da das gerade das Wort ist, was durch vertauschen diese beiden Symbole erzeugt wird. Es gilt also $d(\gamma_n, w) + 2 = d(\beta_n(k), w)$. Da $\beta_n(k) \in \mathcal{S}_n$ und $d(\mathcal{S}, w) \leq n$, muss $d(\beta_n(k), w) \leq n$ sein und damit $d(\gamma_n, w) \leq n - 2$.

Da $\overline{\gamma}_n$ das "Komplement" von γ_n ist, d.h. alle Zeichen wurden vertauscht, ist $d(\overline{\gamma}_n, w) = 2n - d(\gamma_n, w) > n$. Das ist ein Widerspruch, da $\overline{\gamma}_n \in \mathcal{S}_n$, also kann w nicht existieren. Somit ist die Behauptung bewiesen.

Wir zeigen nun, dass 3-SAT $\leq_P$ ClosestString. Sei dafür $\varphi = \bigwedge_{i=1}^{n} C_i$ eine Instanz von 3-SAT, wobei in φ die Variablen $x_1, \ldots, x_m$ vorkommen. Wir nehmen oBdA an, dass in keinem C_i eine Variable sowohl positiv als auch negiert auftaucht, da solche Klauseln immer erfüllt sind.
Wir definieren die Transformation als $\varphi \mapsto (\mathcal{W}, m+1)$ mit $\mathcal{W} = \{u_1, \ldots, u_n\} \cup \{v_1, \ldots, v_r\} \cup \mathcal{S}_{m+1}$. $\mathcal{S}_{m+1}$ ist definiert wie oben und $u_i \in \{0,1\}^{2m+2}$ ist das Wort, das an $2j$- und $(2j+1)$-ter Stelle die Zeichen stehen hat: $\begin{cases} 00 & \text{falls } x_j \text{ in } C_i \text{ vorkommt oder falls } j = m \\ 11 & \text{falls } \overline{x_j} \text{ in } C_i \text{ vorkommt} \\ 01 & \text{sonst} \end{cases}$.
Zum Beispiel ist für $C_i = x_1 \vee \overline{x_2} \vee x_3$ mit $m = 4$ das entsprechende Wort $u_i = 1100110100$.
Weiterhin fügen wir für jede Klausel der Form $C_i = x_j \vee x_j \vee x_j$ das Wort v_i zu $\mathcal{W}$ hinzu, mit $v_i = (01)^{j-1}11(01)^{m-j}01$. Analog für $C_i = \overline{x_j} \vee \overline{x_j} \vee \overline{x_j}$ mit $v_i = (01)^{j-1}00(01)^{m-j}01$.
Zum Beispiel für $C_i = x_1 \vee x_1 \vee x_1$ und $m = 3$ ist $v_i = 11010101$.

Sei $\varphi \in$ 3-SAT, also existiert eine erfüllende Belegung $\mathcal{I}$ der Variablen. Definiere das Wort $w = w_0 \ldots w_{2m+1}$ mit $w_{2i}w_{2i+1} = \begin{cases} 00 & \text{falls } \mathcal{I}(x_i) = 0 \text{ oder i = m} \\ 11 & \text{sonst} \end{cases}$. Dann gilt $w \in \{00, 11\}^{m+1}$, also $d(\mathcal{S}_{m+1}, w) \leq m+1$ nach der Behauptung.
Nun zeigen wir, dass jedes u_i höchstens $m+1$ von w entfernt ist: $d(u_i, w) \leq m+1$. Die letzten beiden Symbole beider Wörter sind 00 und stimmen somit überein. Für jede Variable x_j, die nicht in C_i vorkommt, steht in u_i an der entsprechenden Position 01, was sich in jedem Fall an genauer einer Position von w unterscheidet. Da C_i drei Literale enthält, gibt es höchstens drei Positionen, an denen dies nicht der Fall ist. Da $\mathcal{I}$ φ erfüllt, stimmen sich w und u_i an mindestens einer dieser Positionen überein. Also: 2 Zeichen stimmen überein für das erfüllte Literal + 2 Zeichen am Ende der Wörter stimmen überein + $m-3$ Zeichen stimmen überein für nicht enthaltene Variablen. Somit gilt $d(w, u_i) \leq 2 + 2 + m - 3 = m + 1$.

Es bleiben die Wörter v_i zu betrachten. Die Argumentation für diese Wörter verläuft analog zu den u_i und hat dasselbe Ergebnis: $d(w, v_i) \leq m + 1$.

Sei $(\mathcal{W}, m+1) \in$ ClosestString, also existiert ein Wort $w \in \{0,1\}^{2m+2}$ mit $d(\mathcal{W}, w) \leq m+1$. Da $\mathcal{S}_{m+1} \subseteq \mathcal{W}$ folgt nach der Behauptung, dass w die Form $w \in \{00, 11\}^{m+1}$ hat. Da kein Wort $v \in \mathcal{W}$ mit 11 endet, nehmen wir oBdA an, dass w in 00 endet, da die Hamming-Distanz dadurch höchstens so groß ist, als wenn w in 11 enden würde.
Definiere $\mathcal{I} : \{x_1, \ldots, x_m\} \to \{0,1\}$ die Belegung der Variablen mit $\mathcal{I}(x_j) = w_{2j} = w_{2j+1}$. Angenommen, es existiert eine Klausel C_i, die durch $\mathcal{I}$ nicht erfüllt wird.
Falls $v_i \notin \mathcal{W}$, d.h. C_i enthält mehr als ein einziges Literal, so enthält u_i mindestens zwei Positionen j, j', sodass sich w und u_i an j, $j+1$, j' und $j'+1$ unterscheiden. Außerdem unterscheiden sich die beiden Wörter auch an jedem anderen j oder $j+1$, somit ergibt sich $d(u_i, w) \geq m+2$, was ein Widerspruch wäre.
Falls $v_i \in \mathcal{W}$, so enthält C_i genau ein Literal. Sei oBdA $C_i = x_j \vee x_j \vee x_j$. Dann unterscheiden sich v_i und w an jedem $2j'$ oder $2j'+1$ für $j' \neq j$ und an $2j$ und $2j+1$, also $d(v_i, w) \geq m+2$.

In jedem Fall tritt also ein Widerspruch auf. Somit kann die Klausel C_i nicht existieren und $\mathcal{I}$ erfüllt φ. □

Beispiel:
$\varphi_1 = x \wedge (x \vee y \vee \overline{z})$, $m = 3$
$\mathcal{S}_4 = \{10101010, 01010101, 01101010, 10010101, 10011010, 01100101, 10100110, 01011001, 10101001, 01010110\}$
$u_1 = 11010100, u_2 = 11110000, v_1 = 11010101$
$\mathcal{W} = \mathcal{S}_4 \cup \{u_1, u_2, v_1\}$
φ_1 wird erfüllt durch die Belegung $\mathcal{I} : x \mapsto 1, y \mapsto 1, z \mapsto 0$. $\mathcal{W}$ hat einen String der Hamming-Distanz 4: $w = 11110000$.

$\varphi_2 = (x \vee y) \wedge (\overline{x} \vee y) \wedge (x \vee \overline{y}) \wedge (\overline{x} \wedge \overline{y})$
$\mathcal{S}_3 = \{101010, 010101, 011010, 100101, 100110, 011001, 101001, 010110\}$
$u_1 = 111100, u_2 = 001100, u_3 = 110000, u_4 = 000000$
$\mathcal{W} = \mathcal{S}_3 \cup \{u_1, u_2, u_3, u_4\}$
φ_2 hat keine erfüllende Belegung. Ebenso hat $\mathcal{W}$ keinen String der Hamming-Distanz 3 oder weniger.

f)[1]

Wir zeigen, dass VertexCover $\leq_P$ FeedbackArcSet. Sei dafür $(G, k) = ((V, E), k)$ eine Instanz von VertexCover. Wir definieren die Transformation als $(G, k) \mapsto (G', k) = ((V', E'), k)$.
G' enthält für jeden Knoten $v \in V$ zwei Knoten v_{in} und v_{out}, also $V' = \bigcup_{v \in V} \{v_{\text{in}}, v_{\text{out}}\}$.
Die ausgehenden Kanten eines Knotens in G werden in G' durch gerichtete Kanten von des "out"-Knotens zum "in"-Knoten des Nachbars ersetzt. Da G ungerichtet ist, werden diese Kanten auch vom out-Knoten des Nachbars zum in-Knoten kopiert. Weiterhin werden Kanten von jedem in-Knoten zum out-Knoten des gleichen $v \in V$ eingefügt.
$E' = \{(v_{\text{in}}, v_{\text{out}}) \mid v \in V\} \cup \{(u_{\text{out}}, v_{\text{in}}) \mid \{u, v\} \in E\}$

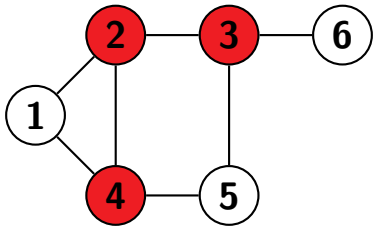
Sei $(G, k) \in$ VertexCover, also existiert ein Vertex Cover C von G mit der Größe $|C| = k$. Behaptung: $A = \{(v_{\text{in}}, v_{\text{out}} \mid v \in C\}$ ist ein FAS.
Sei $G'_A = (V', E' \setminus A)$ der Graph, der durch das Entfernen der Kanten in A entsteht. Angenommen, es existiert ein Zykel in G'_A, also eine Folge von Kanten $\rho = (v_1, \ldots, v_n) \in (V')^n$, sodass $(v_i, v_{i+1}) \in E'$ und $(v_n, v_1) \in E'$. Wir nehmen oBdA an, dass dieser Kreis minimal ist, also jeder Knoten höchstens einmal vorkommt.
Nach der Konstruktion von E' folgt, dass ρ abwechselnd Knoten der Form v_{in} und v_{out} enthalten muss. Die einzigen Kanten, die von einem Knoten v_{in} ausgehen, sind die der Form $(v_{\text{in}}, v_{\text{out}})$. Das heißt, dass in G'_A die Knoten $\{v_{\text{in}} | v \in C\}$ keine Nachfolger haben und somit nicht in ρ vorkommen. Weiterhin sind die einzigen eingehenden Kanten in v_{out} die der Form $(v_{\text{in}}, v_{\text{out}})$. Mit dem gleichen Argument wie vorhin folgt, dass auch $\{v_{\text{out}} | v \in C\}$ nicht in ρ vorkommen können.
Sei also $v \in V$ ein Knoten, sodass v_{out} in ρ vorkommt und sei $u \in V$ der Knoten, sodass u_{in} der Knoten in ρ nach v_{out} ist. Nach der Begründung von eben muss gelten, dass $u, v \notin C$, da ρ sonst kein Kreis sein kann. Da u_{in} der Nachfolger in ρ von v_{out} ist, muss nach Konstruktion von E' die Kante $\{u, v\} \in E$ existieren. Das führt aber zu einem Widerspruch: $u, v \notin C$, also ist $\{u, v\}$ zu keinem Knoten des Vertex Cover inzident. Somit kann ρ nicht existieren und G'_A ist kreisfrei.

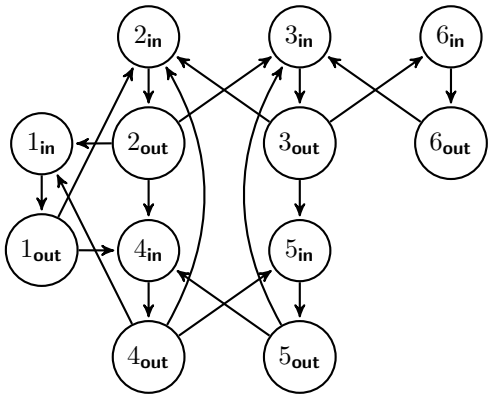
Sei $(G', k) \in$ FeedbackArcSet, also existiert ein FAS A von G' mit der Größe $|A| = k$.
Wir zeigen zunächst, dass auch ein FAS A' existiert, das nur Kanten der Form $(v_{\text{in}}, v_{\text{out}})$ enthält. Dazu benutzen wir ein iteratives Verfahren: Sei $A_0 = A$. Für ein A_i sei dann $e \in A_i$ eine Kante der Form $e = (u_{\text{out}}, v_{\text{in}})$. Dann konstruiere A_{i+1} aus A_i, indem e entfernt und die Kante $(v_{\text{in}}, v_{\text{out}})$ eingefügt wird: $A_{i+1} := (A_i \setminus e) \cup \{(v_{\text{in}}, v_{\text{out}})\}$. Da A endlich ist, muss irgendwann ein i erreicht werden, sodass A_i nur Kanten der gewünschten Form enthält. Setze dann $A' := A_i$.
Es bleibt zu zeigen, dass alle A_i FAS sind. Für A_0 ist das offensichtlich erfüllt. Sei also A_i ein FAS. Angenommen, A_{i+1} ist kein FAS, also existiert ein Kreis $\rho = (v_1, \ldots, v_n) \in (V')^n$ in $(V', E' \setminus A_{i+1})$. Da A_i ein FAS ist, muss in ρ die Kante $(u_{\text{out}}, v_{\text{in}})$ vorkommen, die in A_{i+1} nicht mehr enthalten ist. Da aber (wie schon im ersten Teil des Beweises beschrieben) die einzige ausgehende Kante von v_{in} die Kante $(v_{\text{in}}, v_{\text{out}})$ ist, muss auch diese in ρ enthalten sein. Aber $(v_{\text{in}}, v_{\text{out}}) \in A_{i+1}$, also existiert diese Kante nicht im Graphen. Somit

tritt ein Widerspruch auf, welcher zeigt, dass ρ nicht existieren kann. Also muss A_{i+1} ein FAS sein. Damit ist die Existenz des FAS A' gezeigt.

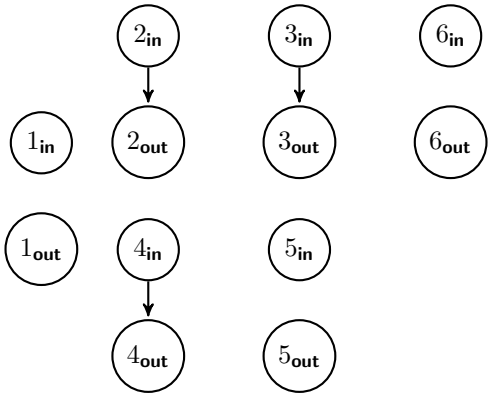
Wir nehmen also nun an, dass oBdA A nur Kanten der Form $(v_{\text{in}}, v_{\text{out}})$ enthält. Behauptung: $C = \{v \in V \mid (v_{\text{in}}, v_{\text{out}}) \in A\}$ ist ein Vertex Cover von G.
Angenommen es existiert eine Kante $e = \{u, v\} \in E$, die zu keinem Knoten aus C inzident ist. Nach Konstruktion wäre dann $\{(u_{\text{out}}, v_{\text{in}}), (v_{\text{in}}, v_{\text{out}}), (v_{\text{out}}, u_{\text{in}}), (u_{\text{in}}, u_{\text{out}})\} \subseteq E'$. Außerdem ist nach Definition von C keine dieser vier Kanten in A enthalten. Aber dann lässt sich daraus ein Kreis (der Länge 4) im Graphen $(V', E' \setminus A)$ bilden. Das widerspricht der Annahme, das A ein FAS ist. Also kann e nicht existieren und C muss ein Vertex Cover von G sein. □

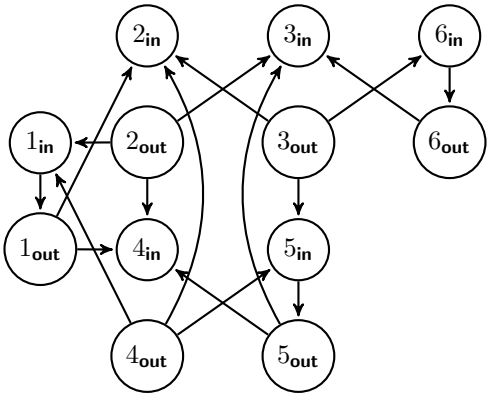
Beispiel:
Der Graph enthält ein Vertex Cover der Größe 3 (links), aber keins der Größe 2. Der transformierte Graph G' ist rechts abgebildet.





G' enthält kein FAS der Größe 2, aber eins der Größe 3, welches links unten abgebildet ist. Rechts ist G' dargestellt, nachdem die Kanten des FAS entfernt wurden. Man kann sehen, dass dieser Graph kreisfrei ist.





References

[**1**] ftp://ftp.cis.upenn.edu/pub/htdocs/.to_remove_Dec_2005/cisgrad_public_html/502-03-wpe.pdf
[**2**] http://www2.informatik.uni-freiburg.de/~accorsi/papers/MAX2SAT.pdf
[**3**] http://www.its.caltech.edu/~momar/boucher_omar.pdf