Berechenbarkeit und Komplexität

Prof. Dr. Martin Grohe
Kamal Al-Bawani, Oliver Göbel, Sandra Kiefer, Daniel Neuen

WS 2015/16 - Klausur
04.03.2016

Aufgabe 1: Deterministische Turingmaschinen (3 + 6 + 2 + 6 = 17)

a) Geben Sie die Definition einer Turingmaschine an. Geben Sie die Bedeutung der einzelnen Komponenten an.

Lösung: Eine Turingmaschine ist ein 7-Tupel $M = (Q, \Sigma, \Gamma, B, q_0, \overline{q}, \delta)$ mit

- endlicher Zustandsmenge Q,
- endlichem Eingabealphabet Σ,
- endlichen Bandalphabet $\Gamma \supseteq \Sigma$,
- Blank-Symbol B,
- Startzustand q_0,
- Endzustand $\overline{q}$,
- Übergangsfunktion $\delta = (Q \setminus \{\overline{q}\}) \times \Gamma \to Q \times \Gamma \times \{R, N, L\}$

b) Betrachten Sie die folgende Turingmaschine M.

$$M = (\{q_0, q_1, q_2, \overline{q}\}, \{0, 1\}, \{0, 1, B\}, B, q_0, \overline{q}, \delta)$$

δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_0, 1, R)$	(q_1, B, L)
q_1	(q_2, B, L)	(q_2, B, L)	$(\overline{q}, 0, N)$
q_2	$(\overline{q}, 1, N)$	$(\overline{q}, 0, N)$	$(\overline{q}, 0, N)$

Welche Funktion wird von M berechnet? Geben Sie eine kurze Begründung an.

Lösung: Die Turingmaschine M berechnet die Funktion

$$f(w) = \begin{cases} 1 & \text{falls } w = v0a \text{ für ein } a \in \{0,1\}, v \in \{0,1\}^* \\ 0 & \text{sonst} \end{cases}$$

Im Zustand q_0 läuft die TM bis ans Ende der Eingabe ohne diese zu verändern und wechselt dann in den Zustand q_1, wobei der Kopf danach auf dem letzten Zeichen der Eingabe steht. In Zustand q_1 wird das letzte Zeichen entfernt (falls dieses nicht existiert, die Eingabe also leer war, verwirft M), ein Schritt nach links gemacht und in den Zustand q_2 gewechselt. Dieser betrachtet nun das vorletzte Zeichen der Eingabe und akzeptiert genau dann wenn dieses eine 0 ist.

c) Was besagt die Church-Turing-These?

Lösung: Die Klasse der TM-berechenbaren Funktionen stimmt mit der Klasse der "intuitiv berechenbare" Funktionen überein.

d) Wir betrachten folgende Einschränkung einer Turingmaschine: Bei jedem Berechnungsschritt muss sich der Kopf der Turingmaschine entweder nach rechts oder links bewegen, d.h. Transitionen, bei denen sich der Kopf nicht bewegt, sind nicht erlaubt.

Zeigen Sie, dass für jede Turingmaschine M' eine Turingmaschine M' existiert, wobei M' die gleiche Funktion berechnet wie M und M' keine Transitionen der Form $(q, a) \mapsto (p, b, N)$ besitzt. Gehen Sie dabei auch auf die Korrektheit und den Laufzeitverlust der Simulation ein.

Lösung: Die Idee für die Simulation ist, dass wir jeden Schritt, in dem sich der Kopf der Turingmaschine nicht bewegt, durch einen Schritt nach rechts gefolgt von einem Schritt nach links simulieren. Dazu erweitern wir die Turingmaschine M um die Zustände q' für jedes $q \in Q$. Außerdem ersetzen wir jede Transition der Form $(q, a) \mapsto (p, b, N)$ durch Transitionen $(q, a) \mapsto (p', b, R)$ und $(p', c) \mapsto (p, b, N)$. Ansonsten lassen wir M unverändert.

Die Simultion ist korrekt, da sich M' nach Ausführen der zwei Schritt in derselben Konfiguration befindet wie M nach Ausführen der Transition $(q, a) \mapsto (p, b, N)$. Somit erhalten wir für beide Turingmaschinen dieselben Konfigurationsfolgen abgesehen von den Zwischenkonfigurationen von M'. Also berechnet M' dieselbe Konfiguration wie M. Der Laufzeitverlust ist kostant, da wir jeden Schritt von M durch höchstens zwei Schritte von M' ersetzen.

Aufgabe 2: Entscheidbare und rekursiv aufzählbare Sprachen $(2 + 2 + 3 + 4 + 8 = 19)$

a) Was bedeutet es, dass eine Turingmaschine die Sprache L entscheidet?

Lösung: Eine Turingmaschine entscheidet eine Sprache L, wenn sie auf jeder Eingabe x hält und genau dann akzeptiert, wenn $x \in L$ gilt.

b) Was bedeutet es, dass eine Turingmaschine die Sprache L aufzählt?

Lösung: Eine Turingmaschine zählt eine Sprache L auf, wenn sie jedes $x \in L$ mindestens einmal auf ein spezielles Ausgabeband schreibt (Aufzähler).

c) Geben Sie eine Sprache (inklusive Definition) an, bei der Weder die Sprache selbst noch deren Komplement semi-entscheidbar ist.

Lösung: $H_{all} = \{\langle M \rangle \mid M \text{ hält auf allen Eingaben}\}$

d) Zeigen Sie, dass die Sprache

$$L = \{\langle M \rangle \mid M \text{ schreibt zu der leeren Eingabe ein nicht-leeres Zeichen irgendwo auf das Band}\}$$

rekursiv ist. Ein nicht-leeres Zeichen ist ein Zeichen aus $\Gamma \setminus \{B\}$.

(**Hinweis:** Betrachten Sie die ersten m Konfigurationen, wobei m die Anzahl der Zustände von M ist.)

Lösung: Wir konstruieren eine TM M_L, die L entscheidet. M_L simuliert M auf der leeren Eingabe. Wenn M kein nicht-leeres Zeichen innerhalb der ersten m Schritte schreibt, akzeptiert M_L; sonst verwirft M_L.

Korrektheit: Offensichtlich terminiert M_L immer. Wir müssen nur das folgende zeigen: Wenn M kein nicht-leeres Zeichen innerhalb der ersten n Schritte schreibt, schreibt sie nie ein nicht-leeres Zeichen.

Betrachte den Fall wenn M kein nicht-leeres Zeichen schreibt. Da das Band in diesem Fall leer ist, gibt es nur $m-1$ viele verschiedene Konfigurationen, die kein Stopzustand sind. Nach m Schritten hat M also entweder einen Stopzustand erreicht oder eine Konfiguration zweimal besucht. Besucht eine TM zweimal die selbe Konfiguration, so befindet sie sich in einer Endlosschleife. Deshalb schreibt M nie ein nicht-leeres Zeichen, wenn sie das innerhalb der ersten m Schritte nicht macht.

e) Zeigen Sie durch Unterprogrammtechnik, dass die Sprache

$$L' = \{\langle M\rangle \mid M \text{ schreibt bei der leeren Eingabe eine 1 irgendwo auf das Band}\}$$

nicht rekursiv ist.

Lösung: Zum Widerspruch nehmen wir an, es gibt eine TM $M_{L'}$, die die Sprache L' entscheidet.

Konstruktion: Wir konstruieren nun eine TM M_{H_ϵ}, die die Maschine $M_{L'}$ als Unterprogramm verwendet, um die Sprache H_ϵ zu entscheiden: Ist die Eingabe nicht der Form $\langle M\rangle$, so verwirft M_{H_ϵ}. Zu einer Eingabe $\langle M\rangle$ berechnet M_{H_ϵ} zunächst die Gödelnummer einer TM $M_\#$, die eine Kopie von $\langle M\rangle$ ist, mit dem einzigen Unterschied, dass jede 1 in $\langle M\rangle$ durch # ersetzt wird, wobei # nicht in dem Bandalphabet von M enthalten ist. Offensichtlich schreibt $M_\#$ nie eine 1. Danach berechnet M_{H_ϵ} die Gödelnummer einer anderen TM M^*, die $M_\#$ simuliert und, wenn $M_\#$ stoppt, eine 1 irgendwo auf das Band schreibt. Anschließend simuliert M_{H_ϵ} die Maschine $M_{L'}$ auf der Eingabe $\langle M^*\rangle$ und übernimmt deren Akzeptanzverhalten.

Korrektheit: Wir zeigen nun, dass M_{H_ϵ} die Sprache H_ϵ entscheidet. Da H_ϵ gemäß der Vorlesung unentscheidbar ist, ist dies ein Widerspruch und deshalb ist L' unentscheidbar
Gemäß der Konstruktion verwirft M_{H_ϵ}, wenn die Eingabe keine Gödelnummer ist. Wir nehmen im folgenden an, dass die Eingabe der Form $\langle M\rangle$ ist. Es gilt dann

$$\begin{aligned}
\langle M\rangle \in H_\epsilon &\Rightarrow M \text{ hält auf der leeren Eingabe}\\
&\Rightarrow M_\# \text{ hält auf der leeren Eingabe}\\
&\Rightarrow M^* \text{ schreibt bei der leeren Eingabe eine 1 irgendwo auf das Band}\\
&\Rightarrow M_{L'} \text{ akzeptiert } \langle M^*\rangle\\
&\Rightarrow M_{H_\epsilon} \text{ akzeptiert } \langle M\rangle,
\end{aligned}$$

und weiterhin

$$\begin{aligned}
\langle M\rangle \notin H_\epsilon &\Rightarrow M \text{ hält nicht auf der leeren Eingabe}\\
&\Rightarrow M_\# \text{ hält auf der leeren Eingabe}\\
&\Rightarrow M^* \text{ schreibt bei keiner Eingabe eine 1 auf das Band}\\
&\Rightarrow M_{L'} \text{ verwirft } \langle M^*\rangle\\
&\Rightarrow M_{H_\epsilon} \text{ verwirft } \langle M\rangle.
\end{aligned}$$

Aufgabe 3: Satz von Rice $(4 + 7 = 11)$

a) Was besagt der Satz von Rice? Nennen Sie insbesondere auch die Voraussetzungen, die erfüllt sein müssen, damit der Satz von Rice angewendet werden kann.

Lösung: Sei $\mathcal{R}$ die Menge der von Turingmaschinen berechenbaren partiellen Funktionen und $\mathcal{S}$ eine Teilmenge von $\mathcal{R}$ mit $\emptyset \neq \mathcal{S} \neq \mathcal{R}$. Dann ist die Sprache

$$L(\mathcal{S}) = \{\langle M \rangle \mid M \text{ berechnet eine Funktion aus } \mathcal{S}\}$$

nicht rekursiv.

b) Zeigen Sie mit Hilfe des Satzes von Rice, dass die Sprache

$$T_{-2} = \{\langle M \rangle \mid M \text{ hält auf allem außer genau zwei Eingabewörtern}\}$$

nicht entscheidbar ist. Zeigen Sie dazu, dass die Bedingungen des Satzes von Rice erfüllt sind.

Lösung: Betrachte die folgende Menge von TM-berechenbaren partiellen Funktionen

$$S = \{f_M \mid \exists\, w_1, w_2 \in \Sigma^* : f_M(w_1) = f_M(w_2) = \bot \bigwedge \forall\, w \in \Sigma^* \setminus \{w_1, w_2\}, f_M \neq \bot\}$$

S ist ungleich $\emptyset$, da wir eine TM M konstruieren können, sodass M auf jedem Eingabewort außer zwei Wörtern, z.B. 00 oder 11 hält. Die entsprechendende Funktion f_M ist offensichtlich in S.
Ebenfalls ist S ungleich $\mathcal{R}$, da die Funktion $f_M(w) = 1$ für alle $w \in \Sigma*$, d.h. die entsprechende Turingmaschine M hält auf allen Eingaben, offensichtlich nicht in S enthalten ist. Damit ist die Sprache

$$\begin{aligned} L(S) &= \{\langle M \rangle \mid M \text{ berechnet eine Funktion aus } S\} \\ &= \{\langle M \rangle \mid M \text{ hält auf jedem Eingabewort außer zwei Wörtern}\} \\ &= T_{-2} \end{aligned}$$

gemäß des Satzes von Rice nicht entscheidbar.

Aufgabe 4: Reduktion $(3 + 4 + 4 + 10 = 21)$

a) Was bedeutet es formal, dass eine Sprache $L_1 \subseteq \Sigma_1^*$ auf eine Sprache $L_2 \subseteq \Sigma_2^*$ reduzierbar ist $(L_1 \leq L_2\ )$?

Lösung: L_1 und L_2 seien zwei Sprachen über Σ_1 bzw. Σ_2. L_1 ist reduzierbar auf L_2, wenn es eine berechenbare Funktion $f : \Sigma_1^* \to \Sigma_2^*$ gibt, so dass für alle $x \in \Sigma_1^*$ gilt:

$$x \in L_1 \Leftrightarrow f(x) \in L_2$$

b) Definieren Sie das Halteproblem H und sein Komplement $\overline{H}$.

$H = \{$ $\}$

$\overline{H} = \{$ $\}$

Lösung: $H = \{\langle M \rangle w \mid M \text{ hält auf } w\}$
$\overline{H} = \{w \mid w \neq \langle M \rangle v \text{ oder } (w = \langle M \rangle v \text{ und } M \text{ hält nicht auf } v)\}$.

c) Zeigen Sie, dass $\overline{H}$ nicht rekursiv aufzählbar ist.

Lösung: H ist semi-entscheidbar, indem wir bei Eingabe $\langle M \rangle w$ die Turingmaschine M auf Eingabe w simulieren und akzeptieren, falls die Simulation hält. Somit ist H rekursiv aufzählbar. Angenommen $\overline{H}$ ist rekursiv aufzählbar, dann ist sowohl H als auch $\overline{H}$ rekursiv aufzählbar, also ist H entscheidbar. Das ist ein Widerspruch zur Vorlesung. Also ist $\overline{H}$ nicht rekursiv aufzählbar.

d) Zeigen Sie durch Reduktion, dass die Sprache

$$EQ_{TM} = \{(\langle M_1 \rangle, \langle M_2 \rangle) \mid L(M_1) = L(M_2)\}$$

nicht rekursiv aufzählbar ist. Beweisen Sie insbesondere die Korrektheit Ihrer Reduktion, d.h. die verwendete Äquivalenzaussage.

Lösung: Wir zeigen eine Reduktion $H \leq \overline{EQ_{TM}}$. Dann gilt gemäß der Vorlesung, dass $\overline{H} \leq \overline{EQ_{TM}}$. Da $\overline{H}$ nach Teilaufgabe c) nicht rekursiv aufzählbar ist, ist $\overline{EQ_{TM}}$ ebenfalls nicht rekursiv aufzählbar.

Konstruktion:
Für die Reduktion konstruieren wir die Funktion $f : \Sigma^* \to \Sigma^*$, so sodass

$$x \in H \Leftrightarrow f(x) \in \overline{EQ_{TM}}$$

gilt. Zunächst sei x der Form $\langle M \rangle w$. Wir konstruieren zwei Turingmaschinen M_1 und M_2 wie folgt, und setzen $f(x) = (\langle M_1 \rangle, \langle M_2 \rangle)$. Unabhängig von der Eingabe verwirft M_1 immer. Offensichtlich gilt $L(M_1) = \emptyset$. M_2 arbeitet auf folgende Weise: Zunächst löscht M_2 die Eingabe, dann simuliert sie die Maschine M auf der Eingabe w. Wenn M hält, dann akzeptiert M_2. Falls x nicht der Form $\langle M \rangle w$ ist, dann setze $f(x) = (\langle M_1 \rangle, \langle M_1 \rangle)$.

Berechenbarkeit:
Offensichtlich ist die Funktion f berechenbar.

Korrektheit:
Falls x nicht der Form $\langle M \rangle w$ ist, so ist $x \notin H$ und gemäß der Konstruktion $f(x) \notin \overline{EQ_{TM}}$. Nun sei $x = \langle M \rangle x$. Es gilt

$$\begin{aligned}
x \in H &\Rightarrow M \text{ hält auf } w \\
&\Rightarrow M_2 \text{ akzeptiert jede Eingabe} \\
&\Rightarrow L(M_1) \neq L(M_2) \\
&\Rightarrow (\langle M_1 \rangle, \langle M_2 \rangle) \notin EQ_{TM} \\
&\Rightarrow (\langle M_1 \rangle, \langle M_2 \rangle) \in \overline{EQ_{TM}}
\end{aligned}$$

$$\begin{aligned}
x \notin H &\Rightarrow M \text{ hält nicht auf } w \\
&\Rightarrow M_2 \text{ hält auf keiner Eingabe} \\
&\Rightarrow M_2 \text{ akzeptiert keine Eingabe} \\
&\Rightarrow L(M_1) = L(M_2) \\
&\Rightarrow (\langle M_1 \rangle, \langle M_2 \rangle) \in EQ_{TM} \\
&\Rightarrow (\langle M_1 \rangle, \langle M_2 \rangle) \notin \overline{EQ_{TM}}
\end{aligned}$$

Aufgabe 5: Komplexitätstheorie $(3+3+3+3+2+5=19)$

a) Wie kann man die Zugehörigkeit eines Problems zur Klasse P zeigen?

Lösung: Ein Problem ist in P wenn es einen Polynomialzeitalgorithmus gibt, der das Problem auf einer DTM löst.

b) Geben Sie eine Möglichkeit an, die Zugehörigkeit eines Problems zur Klasse NP zu zeigen.

Lösung: Ein Problem ist in NP wenn es einen Polynomialzeitalgorithmus gibt, der das Problem auf einer NTM löst **oder**
Zertifikat poly. in der Eingabelänge beschränkt und Verifizierer mit poly. Laufzeit.

c) Wie ist das Akzeptanzverhalten einer nichtdeterministischen Turingmaschine definiert?

Lösung: Eine nichtdeterministische Turingmaschine M akzeptiert die Eingabe $x \in \Sigma^*$, falls es mindestens einen Rechenweg von M gibt, der in eine akzeptierende Endkonfiguration führt.

d) Definieren Sie, wann ein Entscheidungsproblem A NP-vollständig ist.

Lösung: A ist NP-vollständig, falls gilt

1) $A \in \mathsf{NP}$ und
2) A ist NP-schwer.

e) Was ist die Aussage des Satzes von Cook und Levin?

Lösung: SAT ist NP-vollständig.

f) Für ein Entscheidungsproblem A gebe es einen Algorithmus mit Laufzeitschranke $O(n^2 \cdot \log(n))$. Ferne gebe es eine polynomielle Reduktion $B \leq_p A$, die Laufzeit $O(m^3)$ benötigt. Dabei bezeichnen n und m jeweils die Eingabelängen der Probleme A bzw. B. Welche Laufzeitschranke kann man daraus für einen Algorithmus für B folgern? Geben Sie eine möglichst genaue Abschätzung an.

Lösung: $O(m^6 \cdot \log(m))$. Algorithmus B erhält eine Eingabe der Länge m. B führt nun zuerst eine Reduktion aus. Dabei wird die Eingabeinstanz von B auf eine Eingabeinstanz von A der Länge höchstens $O(m^3)$ abgebildet. Nun wird der Algorithmus für das Problem A auf der Eingabeinstanz aufgerufen. Seine Laufzeit beträgt bei Eingabelänge $O(m^3)$ höchstens $O((m^3)^2 \cdot \log(m^3)) = O(m^6 \cdot \log(m))$.

Aufgabe 6: NP-Vollständigkeit $(15+8=23)$

a) HittingSet ist das folgende Entscheidungsproblem:

Eingabe: Eine Grundmenge S an Elementen, eine Familie $\mathcal{C} = \{C \mid C \subseteq S\}$ an Teilmengen von Elementen, eine natürliche Zahl $k \leq |S|$.

Ausgabe: Ja, gdw. es eine Teilmenge $S' \subseteq S$ mit $|S'| \leq k$ gibt, sodass S' aus jeder der Teilmengen $C \in \mathcal{C}$ mindestens ein Element enthält.

Zeigen Sie, dass das Problem HittingSet NP-vollständig ist.

Beachten Sie dabei beide Aspekte der NP-Vollständigkeit, beweisen Sie Korrektheit und polynomielle Beschränktheit, sofern diese in Ihrem Beweis wichtig sind.

Lösung: Damit NP-Vollständigkeit gegeben ist, muss das vorliegende Problem in NP enthalten sein und es muss NP-schwer sein.

Wir zeigen, dass HittingSet in NP enthalten ist, indem wir ein Zertifikat mit polynomiell beschränkter Länge sowie einen Verifizierer angeben, der in polynomiell beschränkter Laufzeit die Zugehörigkeit zu NP anhand des Zertifikats entscheidet.

Das Zertifikat besteht aus einer Teilmenge von S, welche die Mengen aus $\mathcal{C}$ abdecken soll. Die Länge eines gültigen Zertifikats ist damit durch die Größe von S und damit polynomiell in der Eingabe beschränkt.

Der Verifizierer iteriert nun über die Elemente des Zertifikats und markiert für jedes Element, welche Menge aus $\mathcal{C}$ es abdeckt. Am Ende wird über alle Mengen aus C iteriert und geprüft, ob jede Menge markiert (also abgedeckt) ist. Die Laufzeit des Verifizierers ist polynomiell in der Eingabelänge beschränkt.

Wir zeigen nun, dass HittingSet NP-schwer ist. Dazu geben wir eine Reduktion von VertexCover an, d.h. wir zeigen wie wir aus einer Instanz für VertexCover eine Instanz für HittingSet erhalten. Bei VertexCover besteht die Instanz aus einem Graphen $G = (V, E)$ mit Knotenmenge V und Kantenmenge $E = \{\{u, v\} \mid u, v \in V\}$. Wir definieren $S = V$, d.h. die Menge an wählbaren Elementen besteht aus genau den Knoten des Graphens. Weiterhin definieren wir $\mathcal{C} = E$, d.h. die zu überdeckenden Teilmengen sind genau die Kanten des Graphens. Die Konstruktion ist polynomiell beschränkt.
Korrektheit: Existiert ein gültiges VertexCover so bedeutet dies, dass es eine Teilmenge an Knoten gibt, so dass jede Kante abgedeckt ist. Aufgrund der Konstruktion entspricht dieses VertexCover dann einer Teilmenge S' fdie alle Mengen in $\mathcal{C}$ abdeckt. Somit hat die konstruierte Instanz ein gültiges HittingSet. Die Rückrichtung gilt analog.

b) Bei der Optimierungsvariante von HittingSet ist nach der Menge S' gefragt, die die minimale Größe hat und sonst die gleiche Anforderungen erfüllt wie in der Defition von HittingSet beschrieben.

Beweisen Sie die folgende Aussage: Wenn die in Teil a) definierte Einscheidungsvariante von HittingSet in Pliegt, so ist auch die Optimierungsvariante von HittingSet in P.

Lösung: Wir nehmen an, es existiere ein in der Laufzeit polynomiell beschränkter Algorithmus, der die Entscheidungsvariante von HittingSet löst.

Es ist bekannt, dass für die Größe k des optimalen HittingSet gilt, dass $1 \leq k^* \leq n$ ist, wobei $|S| = n$ ist. Gemäß der Annahme existiert ein Algorithmus, der in polynomiell beschränkter Laufzeit entscheidet, ob eine Instanz ein HittingSet einer bestimmten Größe k hat.

Wir bestimmen nun zunächst die Größe k^* des optimalen HittingSet. Dazu führen wir eine Binärsuche auf dem Intervall $[1; n]$ aus und starten für jeden betracheten Wert k' den effizienten Entscheidungsalgorithmus der ausgibt, ob es ein HittingSet mit Größe k' gibt. Für den kleinsten, auf diese Weise bestimmten, Wert k' gilt dann $k^* = k'$.

Nun berechnen wir mit Hilfe von k^* die optimale Lösung. Dazu iterieren wir über alle Elemente $s \in S$. Wird ein Element s betrachtet, so starten wir einen effizienten Entscheidungsalgorithmus auf $(S \setminus \{s\}, k^*)$und erfahren damit, ob es eine Lösung der (optimalen) Größe k^* gibt, auch wenn das Element s nicht zum Abdecken zur Verfügung steht. Ist die Antwort "JA", so gibt es eine Lösung die k^* erreicht, ohne s zu verwenden und wir wissen, dass s nicht Teil des optimalen HittingSet ist. Wir setzen $S \leftarrow S \setminus \{s\}$ und entfernen s somit. Ist die Antwort des Algorithmus "NEIN" so wissen wir, dass s Teil des optimalen HittingSet ist. Wir belassen s in S.

Nachdem über alle Elemente in S iteriert wurde sind nur noch die Elemente enthalten, die auch Teil des optimalen HittingSet sind.

Der hier beschriebene Algorithmus führt eine Binärsuche durch und iteriert danach über alle Elemente in S, beide ist in polynomiell beschränkter Laufzeit möglich. Andernfalls wir nur der, gemäß der Annahme ebenfalls effizient arbeitende, Entscheidungsalgorithmus aufgerufen. Die Laufzeit des gesamten Verfahrens ist somit in polynomiell beschränkter Zeit möglich, womit die Optimierungsvariante von HittingSet effizient lösbar ist.

Aufgabe 7: Polynomielle Reduktion (10)

a) Wir betrachten das folgende Problem Half-Clique:

Eingabe: Ein Graph $G = (V, E)$ mit gerader Anzahl an Knoten $n = |V|$.

Ausgabe: Ja, gdw. es eine Knotenmenge $C \subseteq V$ mit $|C| = \frac{n}{2}$ gibt, so dass der von C induzierte Subgraph vollständig ist, d.h. die Menge C bildet eine $\frac{n}{2}$-Clique.

Zeigen Sie, dass Clique $\leq_p$ Half-Clique.

Hierbei ist Clique das aus der Vorlesung bekannte Problem:

Eingabe: Ein Graph $G = (V, E)$ und eine Zahl $b \in \mathbb{N}$.

Ausgabe: Ja, gdw. es eine Knotenmenge $K \subseteq V$ mit $|K| \geq b$ gibt, so dass der von K induzierte Subgraph vollständig ist, d.h. die Menge K bildet eine b-Clique.

Lösung: Sei (G, b), wobei $G = (V, E)$ ein Graph ist und $b \in \mathbb{N}$, die Eingabe für das Clique-Problem. Sei $n = |V|$. Für die Reduktion unterscheiden wir zwei Fälle. Falls $b \geq \frac{n}{2}$, dann konstruieren wir den Graphen G' aus dem Graphen G durch Hinzufügen von $2b - n \geq 0$ isolierten Knoten (Knoten ohne inzidente Kanten). Andernfalls ist $b < \frac{n}{2}$ und wir konstruieren den Graphen G' aus dem Graphen G durch Hinzufügen von $n - 2b > 0$ universellen Knoten (Knoten, die eine Kante zu allen anderen Knoten haben).

Die beschriebene Reduktion ist in polynomieller Zeit berechenbar, da in jedem Fall höchstens n Knoten und $2n^2$ Kanten hinzugefügt werden.

Wir müssen also die Korrektheit zeigen. Für die Hinrichtung nehmen wir an, dass $K \subseteq V$ eine Clique der Größe mindestens b in G bildet. Durch Entfernen von Knoten aus K können wir ohne Einschränkung annehmen, dass K genau b Elemente hat. Falls $b \geq \frac{n}{2}$ dann bildet K in G' eine Clique der Größe b, wobei G' genau $n + 2b - n = 2b$ viele Knoten hat. Also ist K' eine Half-Clique in G'. Im zweiten Fall ist $b < \frac{n}{2}$. Sei C die Vereinigung der Menge K mit den hinzugefügten Knoten. Da die hinzugefügten Knoten mit allen anderen Knoten verbunden sind bildet die Menge C eine Clique mit $b + n - 2b = n - b$ Knoten. Da der Graph G' genau $n + n - 2b = 2(n - b)$ viele Knoten hat, bildet C eine Half-Clique.
Für die Rückrichtung sei C eine Half-Clique in G'. Falls $b \geq \frac{n}{2}$, dann enthält C keine der hinzugefügten Knoten, da diese alle isoliert sind. Also bildet C eine Clique in G der Größe b, da G' genau $2b$ viele Knoten hat. Andernfalls ist $b < \frac{n}{2}$ und C hat die Größe $n - b$, da G' genau $2(n - b)$ viele Knoten besitzt. Da wir nun $n - 2b$ Knoten hinzugefägt haben, enthält C mindestens b Knoten aus dem Graphen G. Also ist $K = C \cap V$ eine Clique der Größe mindestens b in G.