Institut für Informatik I
Prof. Dr. Heiko Röglin
Tobias Brunsch

universitätbonn

Algorithmen und Berechnungskomplexität II
Sommersemester 2011

Musterlösung Probeklausur

Aufgabe 1

(a) Eine 1-Band Turingmaschine ist ein 7-Tupel $(Q, \Sigma, \Gamma, B, q_0, \bar{q}, \delta)$ mit

- einer endlichen Zustandsmenge Q,
- einem endlichen Eingabealphabet $\Sigma \neq \emptyset$,
- einem endlichen Bandalphabet $\Gamma \supseteq \Sigma$,
- einem Blank-Symbol $B \in \Gamma \setminus \Sigma$,
- einem Anfangszustand $q_0 \in Q$,
- einem Endzustand $\bar{q} \in Q$ und
- einer Zustandsüberführungsfunktion $\delta\colon (Q \setminus \{\bar{q}\}) \times \Gamma \to Q \times \Gamma \times \{L, N, R\}$.

(b) Eine Turingmaschine M entscheidet eine Sprache L, wenn gilt:

- M akzeptiert jedes Wort $w \in L$ und
- M verwirft jedes Wort $w \notin L$.

Eine Turingmaschine M erkennt eine Sprache L, wenn gilt:

- M akzeptiert jedes Wort $w \in L$ und
- M akzeptiert kein Wort $w \notin L$, d.h. M verwirft Wörter $w \notin L$ oder hält nicht.

(c) Es gibt rekursiv aufzählbare Sprachen, deren Komplement ebenfalls rekursiv aufzählbar ist. Das sind gerade die rekursiven Sprachen. Zum Beispiel sind $L = \emptyset$ und $\overline{L} = \Sigma^*$ rekursive, also insbesondere rekursiv aufzählbare Sprachen.

(d) Die Sprache $L = L_1 \cup L_2$ lässt sich auch schreiben als $L = G \setminus L_{99} = G \cap \overline{L_{99}}$ mit

$$L_{99} = \{\langle M\rangle : M \text{ hält auf der leeren Eingabe nach spätestens 99 Schritten}\} \quad \text{und}$$

$$G = \{\langle M\rangle : M \text{ ist Turingmaschine}\} .$$

Da sowohl G als auch L_{99} rekursiv sind, ist auch die Sprache L rekursiv.

(e) Es gibt keinen Grund anzunehmen, dass es einen solchen Algorithmus nicht gibt. In der Tat werden wir in Algorithmen und Berechnungskomplexität III so einen Algorithmus kennenlernen. Dessen Existenz impliziert jedoch nicht, dass $\mathcal{P} = \mathcal{NP}$ gilt: Die Eingabegröße für das Rucksack-Problem ist $O(n \cdot (\log P + \log b))$, wobei b die Gewichtsschranke ist. Sofern P nicht polynomiell durch n und $\log b$ beschränkt ist, lässt sich nicht folgern, dass $n^2 \cdot P$ polynomiell in der Eingabegröße ist.

(f) Wir überlegen uns, was die beste Laufzeitschranke für L_1 ist, die wir aus der Aufgabenstellung folgern können, falls L_2 in Zeit $O(n^2)$ entschieden werden kann. Wir bezeichnen die in Zeit $O(n \log n)$ berechenbare Reduktion von L_1 auf L_2 mit f. Um L_1 zu entscheiden, können wir wie folgt vorgehen:

(1) Transformiere die Eingabe x auf $y = f(x)$.
(2) Entscheide, ob y zur Sprache L_2 gehört.
(3) Akzeptiere x, falls $y \in L_2$. Ansonsten verwirf x.

Die Laufzeit von Schritt 1 ist nach Voraussetzung $O(n \log n)$ mit $n = |x|$. Schritt 2 kann nach unserer Annahme in Zeit $O(N^2)$ durchgeführt werden. Hier gilt jedoch $N = |y|$, da eine Laufzeit immer in Abhängigkeit von der Eingabegröße angegeben wird und y die Eingabe für die Turingmaschine ist, die L_2 entscheidet. Der dritte Schritt benötigt nur konstante Zeit.

Wir erhalten damit eine Laufzeit von $O(n \log n + N^2)$. Da y in Zeit $O(n \log n)$ berechnet werden kann, gilt $N = |y| = O(n \log n)$. Wir können nicht ausschließen, dass y tatsächlich in dieser Größenordnung liegt. Daher ist die beste Laufzeitschranke für L_1, die wir folgern können, $O(n \log n + (n \log n)^2) = O(n^2 \cdot (\log n)^2)$.

Kann L_1 nicht in Zeit $O(n)$ oder $O(n^2)$ entschieden werden, widerspricht das dieser Laufzeitschranke nicht. Prinzipiell ist es möglich, dass L_2 in Zeit $O(n^2)$ entschieden werden kann. Ist L_1 hingegen nicht in Zeit $O(n^3)$ entscheidbar, dann kann L_2 nicht in Zeit $O(n^2)$ entschieden werden, da das obiger Schranke widerspricht.

Aufgabe 2

(a) Wir können $L_1, \ldots, L_9$ so wählen, dass $L_{\geq 3}^{\leq 5}$ nicht rekursiv aufzählbar ist: Setze $L_1 = L_2 = L_3 = \Sigma^*$ und $L_4 = \ldots = L_9 = L$ für eine rekursiv aufzählbare, aber nichtrekursive Sprache L. Zum Beispiel können wir für L das spezielle Halteproblem H_ε wählen. Dann gilt: $L_{\geq 3}^{\leq 5} = \overline{L}$:

- Ein Wort $w \in L$ gehört zu jeder Sprache $L_1, \ldots, L_9$ und liegt damit nicht in der Sprache $L_{\geq 3}^{\leq 5}$.
- Ein Wort $w \notin L$ gehört den Sprachen L_1, L_2 und L_3, aber nicht zu den Sprachen $L_4, \ldots, L_9$ und liegt damit in der Sprache $L_{\geq 3}^{\leq 5}$.

Die Sprache $L_{\geq 3}^{\leq 5} = \overline{L}$ ist daher nicht rekursiv aufzählbar, da L nicht rekursiv ist.

(b) Wir betrachten die Menge S aller berechenbaren Funktionen f, die Kodierungen zusammenhängender ungerichteter Graphen auf 1 und Kodierungen nichtzusammenhängender ungerichteter Graphen auf 0 abbilden. An den Funktionswert $f(w)$ von Wörtern w, die keinen ungerichteten Graphen kodieren, stellen wir keine Forderungen. Da Graphzusammenhang entscheidbar ist, ist S nicht leer. Außerdem enthält S nicht alle berechenbaren Funktionen, da zum Beispiel die Konstant-1-Funktion nicht in S enthalten ist. Aufgrund des Satzes von Rice ist die Sprache $L_{\text{connect}} = \{\langle M \rangle : f_M \in S\}$ nicht entscheidbar.

(c) Wir reduzieren das Komplement $\overline{H}_\varepsilon$ des speziellen Halteproblems auf die Sprache $\overline{H}_\varepsilon^{\text{even}}$. Dazu betrachten wir folgende berechenbare Funktion $f\colon \Sigma^* \to \Sigma^*$: Die Funktion f bildet ein Wort, dass keine Turingmaschine kodiert (und damit in $\overline{H}_\varepsilon$ liegt), auf das Wort $\langle M^* \rangle \in \overline{H}_\varepsilon^{\text{even}}$ ab, wobei M^* eine Turingmaschine ist, die das leere Wort nach einem Schritt akzeptiert.

Weiterhin bildet f ein Wort der Form $\langle M \rangle$ auf das Wort $\langle M_{2x} \rangle$ ab, wobei sich M_{2x} genau wie M verhält, aber vor jedem Schritt von M einen Dummy-Schritt ausführt. Dadurch benötigt M_{2x} genau doppelt so viele Schritte wie M und es folgt: M hält genau dann nicht auf der leeren Eingabe, wenn M_{2x} nicht nach einer geraden Anzahl an Schritten auf der leeren Eingabe hält. Für alle Wörter $w \in \Sigma^*$ gilt also $w \in \overline{H}_\varepsilon \iff f(w) \in \overline{H}_\varepsilon^{\text{even}}$. Da $\overline{H}_\varepsilon$ nicht rekursiv aufzählbar ist, kann $\overline{H}_\varepsilon^{\text{even}}$ nicht rekursiv aufzählbar sein.

(d) Wir zeigen, dass L entscheidbar ist, indem wir eine Reduktion auf die Sprache

$$L' = \{\text{code}(p) : p \text{ ist ein univariates Polynom mit einer ganzzahligen Nullstelle}\}$$

durchführen und zeigen, dass L' entscheidbar ist. Die Reduktion ist trivial: Das Polynom p nimmt genau dann an einer ganzzahligen Stelle den Wert 3 an, wenn das Polynom $p' = p - 3$ eine ganzzahlige Nullstelle besitzt.

Es bleibt zu zeigen, dass L' entscheidbar ist. Man beachte, dass es sich bei dieser Fragestellung **nicht** um das 10. Hilbertsche Problem handelt, bei dem es um ganzzahlige Nullstellen **multivariater** Polynome geht. Für ein univariates Polynom $p(z) = \sum_{k=0}^{n} \left(a_k \cdot z^k\right)$ können wir wie folgt testen, ob es eine ganzzahlige Nullstelle besitzt oder nicht:

Teste, ob $p(z) = 0$ für mindestens eine ganze Zahl $z \in \{-|a_0|, \ldots, |a_0|\}$ gilt. Falls ja, dann besitzt p eine ganzzahlige Nullstelle, ansonsten nicht.

Zur Korrektheit: Wir zeigen, dass es genügt, nur die Zahlen z im Bereich von $-|a_0|$ bis $|a_0|$ zu überprüfen. Sei z eine ganzzahlige Nullstelle von p. Dann gilt $\sum_{k=0}^{n} \left(a_k \cdot z^k\right) = 0$ und damit $a_0 = -\sum_{k=1}^{n} \left(a_k \cdot z^k\right) = -z \cdot \sum_{k=1}^{n} \left(a_k \cdot z^{k-1}\right)$, d.h. a_0 ist ein ganzzahliges Vielfaches von z. Für $a_0 \neq 0$ folgt damit $z \in \{-|a_0|, \ldots, |a_0|\}$, für $a_0 = 0$ ist $z' = 0 \in \{-|a_0|, \ldots, |a_0|\}$ eine ganzzahlige Nullstelle von p.

Aufgabe 3

(a) Mit $t_{\max}(\phi)$ bezeichnen wir die maximale Anzahl an Klauseln einer Formel ϕ in KNF, die unter einer geeigneten Belegung der in ϕ vorkommenden Variablen erfüllt werden können. Unter der Annahme, dass die Entscheidungsvariante von `MAX SAT` in $\mathcal{P}$ liegt, können wir $t_{\max}(\phi)$ in Polynomialzeit bestimmen: Ist m die Anzahl der Klauseln, dann testen wir absteigend für $t = m, \ldots, 0$, ob es eine Belegung gibt, unter der mindestens t Klauseln wahr sind. Sobald wir „ja“ als Antwort erhalten, haben wir $t_{\max}$ bestimmt.

Folgende rekursive Funktion bestimmt eine optimale Belegung:

`MaxSat`(ϕ)
{
  Falls ϕ leer ist oder nur leere Klauseln enthält, brich ab.
  Bestimme $t_{\max}(\phi)$.
  Wähle eine in ϕ vorkommende Variable x_i.
  Konstruiere aus ϕ Formeln ϕ^+ und ϕ^- wie folgt:
  Entferne alle Klauseln, die das Literal x_i/das Literal $\bar{x}_i$ enthalten. Dies seien l^+ bzw. l^- viele.
  Entferne aus den verbleibenden Klauseln alle Vorkommen des Literals $\bar{x}_i$/des Literals x_i.
  Falls $t_{\max}(\phi^+) + l^+ = t_{\max}(\phi)$ gilt, dann setze $x_i = 1$ und rufe `MaxSat`(ϕ^+) auf.
  Ansonsten setze $x_i = 0$ und rufe `MaxSat`(ϕ^-) auf.
}

Die Laufzeit ist polynomiell: Die Anzahl der rekursiven Aufrufe ist beschränkt durch die Anzahl der in ϕ vorkommenden Variablen, und jeder einzelne Aufruf ist in Polynomialzeit durchführbar.

Die Korrektheit folgt aus folgender Überlegung: Bei einer optimalen Belegung muss entweder $x_i = 1$ oder $x_i = 0$ gelten. Wenn man sich für eine der beiden Belegungen von x_i entscheidet, dann werden l^+ bzw. l^- der Klauseln sofort wahr. Klauseln, in denen das Literal von x_i vorkommt, das durch die Wahl der Belegung von x_i gerade 0 wird, können nur durch Belegungen der restlichen Variablen wahr werden. Wir können daher das Literal $\bar{x}_i$ bzw. x_i aus den Klauseln streichen. Dadurch können auch leere Klauseln entstehen. Diese sind als Klauseln zu interpretieren, die unabhängig von der Belegung der restlichen Variablen falsch sind.

Da in jeder optimalen Belegung $x_i = 1$ oder $x_i = 0$ gilt, muss für eine der beiden übrigbleibenden Formeln ϕ^+ und ϕ^- gelten: $t_{\max}(\phi^+) + l^+ = t_{\max}(\phi)$ oder $t_{\max}(\phi^-) + l^- = t_{\max}(\phi)$. Wir wählen eine Belegung für x_i, für die diese Gleichheit gilt, und belegen rekursiv die verbleibenden Variablen so, dass von den verbleibenden Klauseln maximal viele erfüllt werden.

(b) Eine mögliche polynomielle Reduktion sieht wie folgt aus: Wir konstruieren aus einem gerichteten Graphen $G = (V \cup \{w\}, E)$ mit einem ausgezeichneten Knoten w einen neuen Graphen $G' = (V \cup \{w_{\text{in}}, w_{\text{out}}\}, E')$, wobei w_{in} die Kopie von w mit allen eingehenden Kanten und w_{out} die Kopie von w mit allen ausgehenden Kanten ist, d.h.

$$E' = \{(u, v) : (u, v) \in E \text{ und } u, v \neq w\} \cup \{(u, w_{\text{in}}) : (u, w) \in E\} \cup \{(w_{\text{out}}, v) : (w, v) \in E\} .$$

Wir zeigen, dass dann gilt: G besitzt genau dann einen Hamiltonkreis, wenn G' einen einfachen Weg der Länge mindestens n besitzt.

„$\Longrightarrow$“: Sei $(v_1, v_2, \ldots, v_n, v_1)$ ein Hamiltonkreis von G. Ohne Einschränkung sei $v_1 = w$. Dann ist $(w_{\text{out}}, v_2, \ldots, v_n, w_{\text{in}})$ ein einfacher Weg der Länge n: Die Kanten (w_{out}, v_2) und (v_n, w_{in}) existieren in G', da die Kanten (w, v_2) und (v_n, w) in G existiert haben. Die Kanten $(v_i, v_{i+1}$, $i = 2, \ldots, n-1$, haben wir von G übernommen.

„$\Longleftarrow$“: Sei $(v_1, v_2, \ldots, v_n, v_{n+1})$ ein einfacher Weg der Länge (mindestens) n in G'. Dann ist jeder Knoten von G' genau einmal in diesem Weg enthalten. Da w_{out} nur ausgehende Kanten und w_{in} nur eingehende Kanten besitzt, können nur diese Knoten Anfang und Ende des Weges sein, d.h. $v_1 = w_{\text{out}}$ und $v_{n+1} = w_{\text{in}}$. Das Tupel $(w, v_2, \ldots, v_n, w)$ ist dann ein Hamiltonkreis von G: Jeder Knoten kommt genau einmal darin vor. Die Kanten (w, v_2) und (v_n, w) existieren in G, da in G' die Kanten (w_{out}, v_2) und (v_n, w_{in}) existieren. Die Kanten (v_i, v_{i+1}) existieren in G, da sie nicht von der Form (u, w_{in}) oder (w_{out}, v) sind und daher bei der Konstruktion von G' von G übernommen wurden.

Damit haben wir gezeigt, dass die Abbildung $G \mapsto (G', n)$ eine Reduktion von `HAMILTONIAN CYCLE` auf `PATH` ist. Da diese Abbildung in Polynomialzeit berechenbar ist, handelt es sich um eine polynomielle Reduktion.

Aufgabe 4

(a) $\mathcal{NP}$ ist die Menge aller Sprachen L, für die ein Polynom p und ein Polynomialzeitverifizierer V existieren, sodass für jedes Wort $x \in L$ gilt:

- Gehört x zur Sprache L, dann existiert ein Zertifikat $y \in \{0,1\}^*$ mit $|y| \leq p(|x|)$, sodass der Verifizierer V das Wort $y\#x$ akzeptiert.
- Gehört x nicht zur Sprache L, dann wird kein Wort der Gestalt $y\#x$ mit $y \in \{0,1\}^*$ und $|y| \leq p(|x|)$ vom Verifizierer V akzeptiert.

(b) Sei L eine Sprache aus $\mathcal{NP}$. Dann existieren ein Polynom p und ein Polynomialzeitverifizierer V mit den in (a) beschriebenen Eigenschaften. Sei $\hat{q}$ ein Polynom, dass die Laufzeit von V beschränkt. Wir betrachten eine Turingmaschine, die wie folgt arbeitet:

(1) Erhalte x als Eingabe.
(2) Setze $n = |x|$.
(3) Führe Schritt (4) für alle Wörter $y \in \{0,1\}^*$ mit $|y| \leq p(n)$ aus.
(4) Falls der Verifizierer V das Wort $y\#x$ akzeptiert, dann akzeptiere x.
(5) Hat der Verifizierer V kein Wort $y\#x$ akzeptiert, dann verwirf x.

Die Korrektheit folgt unmittelbar aus der äquivalenten Charakterisierung von $\mathcal{NP}$ und der Wahl von p und V. Für die Laufzeitanalyse ignorieren wir unwesentliche Schritte wie das Berechnen von $p(n)$ und das Aufzählen aller $y \in \{0,1\}^*$ mit $|y| \leq p(n)$ und betrachten nur den mehrmaligen Aufruf des Verifizierers V: Insgesamt gibt es $\sum_{k=0}^{p(n)} 2^k \leq 2^{p(n)+1}$ Aufrufe von V. Ein einzelner Aufruf ist in Zeit $\hat{q}(|y\#x|) = O(\hat{q}(p(n)+n)) = O(q(n))$ für ein geeignet gewähltes Polynom q durchführbar. Damit ergibt sich als Gesamtlaufzeit $O\big(q(n) \cdot 2^{p(n)}\big)$.