Prof. Martin Hofmann
Steffen Jost

Ludwig-Maximilians-Universität München
Institut für Informatik

Wintersemester 2002/03
7. November 2002

Lösungsvorschläge zur 3. Übung

Komplexitätstheorie

Lösung zu G11:

a) $\mathcal{P} = co\mathcal{NP} \Leftrightarrow \mathcal{P} = \mathcal{NP}$ da $\mathcal{P}$ unter Komplementen abgeschlossen ist und $\mathcal{NP}$ das Komplement von $co\mathcal{NP}$ist und umgekehrt.

b) $\mathcal{P} = \mathcal{NP} \implies \mathcal{P} = \mathcal{NP} \cap co\mathcal{NP}$ wiederum wegen der Abgeschlossenheit von $\mathcal{P}$ unter Komplementen.

c) $\mathcal{NP} \neq co\mathcal{NP}$ folgt $\mathcal{P} \neq \mathcal{NP}$, denn für $L \in \mathcal{NP} \wedge L \notin co\mathcal{NP}$ folgt $L \notin \mathcal{P}$ – erneut aus Eigenschaft von $\mathcal{P}$ Komplente zu enthalten. Die Gegenrichtung scheint nicht zu gelten.

d) Wenn es eine $co\mathcal{NP}$-vollständige Sprache L gibt mit $L \in \mathcal{NP}$, so gilt $\mathcal{NP} = co\mathcal{NP}$, denn dann kann man jede Sprache aus $co\mathcal{NP}$ in polynomieller Zeit auf L und somit auf ein Problem in $\mathcal{NP}$ zurückführen. Somit $\mathcal{NP} \supseteq co\mathcal{NP}$. $\mathcal{NP} \subseteq co\mathcal{NP}$ folgt dann unmittelbar, da das Komplement von $co\mathcal{NP}$ ja wiederum $\mathcal{NP}$ ist.

Lösung zu G12:

a) Wir nehmen an, das die Formel in konjunktiver Normalform gegeben ist (in der Literatur werden manchmal auch beliebige Formeln zugelassen). Als Beweis geben wir die Konstruktion von T an. Wir verzweigen an der ersten Variable der Eingabeformel und benutzen das Orakel um die Erfüllbarkeit der beide kleineren Formeln zu bestimmen, d.h.: Zuerst kopieren wir die Eingabe einmal und lassen dabei jede Klausel weg, in der die gewählte Variable positiv vorkommt und entfernen zusätzlich alle negativen Vorkommen der Variable. Dann kopieren wir die Eingabe ein zweites Mal und lassen nun alle Klauseln weg, in denen diese Variable negativ vorkommt und entfernen analog alle positiven Vorkommen. Auf die Kopien wenden wir das Orakel an, den schließlich sind die so enstandenen Formeln kleiner als die Eingabe (eine Variable wurde ja eliminiert) und akzeptieren die Eingabe, sofern einer der Orakelaufrufe positiv antwortet. Allerdings sind dabei noch zwei Sonderfälle zu beachten: Sollte bei der Elimination der Variablen die leere Formel entstehen, so können wir sofort mit Akzeptanz terminieren; sollte hingegen eine leere Klausel entstehen, so sparen wir uns den Orakelaufruf und werten diesen gleich als negativ.

Das Kopieren der Eingabe auf das Orakelband benötigt lineare Zeit. Um dabei eine Variable herauszulöschen muß man jede Klausel im voraus anschauen, entscheiden wie die Klausel zu verändern ist und zum Kopieren an den Klauselanfang zurückspulen, was jeweils wieder linear ist. Nach dem Orakelaufruf müssen wir erneut das Eingabeband zurückspulen und den Kopiervorgang noch einmal wiederholen und erhalten damit insgesamt eine lineare, also polynomielle Laufzeitbeschränkung.

b) *Diskussion – Kein Beweis:* Nein, denn wir können eine Sprache $L \in \mathcal{NP}$ zwar in polynomieller Zeit auf ein SAT-Problem reduzieren (da SAT $\mathcal{NP}$-vollständig ist), jedoch gemäß der Definition einer selbst-reduzierenden Sprache benötigen wir die Angabe einer deterministischen, polynomial zeitbeschränkten Turing Maschine, die nur ein Orakel für $L_{<n}$ benutzen darf, nicht jedoch für $\text{SAT}_{<n}$! Also müßten wir, gemäß der oben angegebenen Konstruktion, die beiden Teilformeln wieder in Instanzen der Sprache L zurückübersetzen, was im allgemeinen nicht möglich ist. Selbst wenn L eine $\mathcal{NP}$-vollständige Sprache wäre und somit eine polynomial zeitbeschränkte Rückübersetzung möglich wäre, ist es nicht garantiert das diese nach der Rückübersetzung wirklich kleiner wären als die Instanz, welche wir zu Beginn betrachtet hatten (z.B. so könnte die Rückübersetzung immer deutlich größer ausfallen als die Übersetzung nach SAT, obwohl beide polynomial beschränkt sind).

Lösung zu G13:

a) Jede deterministische Turing Maschine besitzt eine endliche Beschreibung, die man sich wiederum als ein String über einem endlichen Alphabet codiert vorstellen kann. Alle Wörter ber einem endlichen Alphabet kann man konstruktiv aufzählen. Somit haben wir bereits eine konstruktive Aufzählung aller deterministischen Turing Maschinen; zwar enthält diese Aufzählung auch viele unsinnige Maschinen, jedoch wird keine "vernünftige" dabei vergessen. Um nun die Klasse $\mathcal{P}$ konstruktiv aufzuzählen, benötigen wir noch eine Aufzählung aller Polynome und ein Paarkodierung. Eine Zahl z codiert dann ein Zahlenpaar (i, j), was wiederum der i-ten Turing Maschine und dem j-ten Polynom entspricht.

   Nun können wir einen Interpreter dieser Kodierung (manchmal auch *universelle* Turing Maschine genannt) konstruieren, der bei Eingabe (z, x) die Zahl z entsprechend dekodiert, und dann die i-te Turing Maschine auf Eingabe x simuliert und mit Ablehnung abbricht, sobald die Simulation soviel Schritte getan hat, wie dem Wert des j-ten Polynoms an der Stelle $|x|$ entspricht. Somit ist sichergestellt, das unsere Aufzählung nur polynomiell zeitbeschränkte Turing Maschinen enthält.

   Andererseits gilt: Zu jeder Sprache $\hat{L} \in \mathcal{P}$ gibt es nach Definition eine Turing Maschine T mit Laufzeitbeschränkung p. Die Turing Maschine T kommt somit in der Aufzählung aller Turing Maschinen vor, ebenso das Polynom p in der Aufzählung der Polynome. Damit gibt es aber auch eine Zahl z, die dieses Zahlenpaar kodiert.

b) Analog zur vorhergehenden Teilaufgabe, jedoch starten wir zuerst mit einer konstruktiven Aufzählung aller nicht-deterministischen Turing Maschinen. Da die Aufzählung nach Definition jedoch nur deterministische Maschinen aufzählen darf, müssen wir noch den Interpreter abändern: Der Interpreter simuliert die i-te nicht deterministische Turing Maschine auf der Eingabe in deterministischer Weise, d.h. er berechnet alle möglichen Pfade der nicht-deterministischen Maschine. Die polynomielle Laufzeitbeschränkung der nicht-deterministischen erhalten wir, in dem wir nun jeden einzelnen Berechnungspfad nach $p_j(|x|)$ vielen Schritten abschneiden.

c) Sei L eine solche Sprache. Wir müssen nun zeigen, dass wir jede Sprache $L' \in \mathcal{NP}$ in polynomieller Zeit auf L reduzieren können. Da SAT $\mathcal{NP}$-vollständig ist, existiert eine polynomielle Reduktion von L' auf SAT. Eine Reduktion von L' auf L erhalten wir nun, indem wir die Reduktion auf SAT auf den Wörtern abändern, auf denen sich L von SAT unterscheidet, und eine entsprechende Instanz von L zurückliefern. Da wir die Reduktion nur auf endlich vielen Wörtern abändern müssen, ist sie weiterhin in polynomieller Zeit durchführbar.

d) Wiederum konstruieren wir eine universelle deterministische Maschine, die auf Eingabe (z, x) das folgende macht: Die Zahl z interpretieren wir wieder als Paar (i, k). Nach unserer Lösung zu Teil b) haben wir eine konstruktive Aufzählung aller Sprachen aus $\mathcal{NP}$, sei also L_i die i-te Sprache aus $\mathcal{NP}$.

   Zuerst prüfen wir ob für alle x' mit $|x'| < |x|$ gilt: $x' \in \text{SAT} \iff f_k(x') \in L_i$ wobei f_k die k-te deterministisch polynomiell berechenbare Funktion bezeichne. Falls dies so ist, so akzeptieren wir falls $x \in L_i$ und verneinen sonst, ansonsten prüfen wir ob $x \in \text{SAT}$.

   Es bleibt zu zeigen, dass dies tatsächlich die gewünschte Aufzählung ist. Angenommen L ist eine $\mathcal{NP}$-vollständige Sprache. Dann gibt es aber auch ein i' mit $L = L_{i'}$ und eine Reduktion von SAT nach L durch eine deterministich polynomiell berechenbare Funktion f_k. Somit fällt der Test immer positiv aus und wir antworten jenachdem ob $x \in L$ ist oder nicht. Andererseits ist jede erkannte Sprache entweder $\mathcal{NP}$-vollständig oder weicht nur in endlich vielen Stellen von SAT ab, was nach c) aber ebenso $\mathcal{NP}$-vollständigkeit bedeutet. Den wenn die Sprache nicht $\mathcal{NP}$-vollständig war, so gibt es auch keine Reduktion auf SAT und wir erkennen wir dies ab einer Eingabe die groß genug ist und berechnen ja dann ob $x \in \text{SAT}$.