Universität Siegen
Lehrstuhl Theoretische Informatik
Markus Lohrey

Übungsblatt 8

Aufgabe 1
Zeigen Sie, dass $\leq_p$ eine reflexive, transitive, aber nicht antisymmetrische Relation ist.

Lösung
Die Reflexivität ist klar (wähle $f = \mathrm{id}$).

Transitivität: Es gelte $A \leq_p B \leq_p C$ mit Reduktionsabbildungen f (für $A \leq_p B$) und g (für $B \leq_p C$). Zu zeigen: $A \leq_p C$. Wir haben bereits gesehen, dass $\leq$ transitiv ist: Wir haben auf Blatt 5, Aufgabe 1c, gesehen, dass $h = g \circ f$ wieder berechenbar ist. Da f und g total sind, ist h auch total. Außerdem gilt $w \in A \Leftrightarrow h(w) \in C$ (Transitivität von $\Leftrightarrow$).
Es bleibt zu zeigen, dass h in polynomieller Zeit berechenbar ist. Sei $|w| = n$. Da f in polynomieller Zeit berechenbar ist, gibt es Konstanten c und k, sodass sich $f(w)$ in worst-case Laufzeit $c \cdot n^k$ berechnen lässt. Da eine Turingmaschine in dieser Zeit höchstens $c \cdot n^k$ Zeichen geschrieben haben kann, gilt auch $|f(w)| \leq c \cdot n^k$. Analog gilt für g, dass es bei Input y mit $|y| = n$ Konstanten d und ℓ gibt, so dass $g(y)$ höchstens $d \cdot n^\ell$ Rechenschritte benötigt. Wir können also annehmen, dass $|y| = |f(w)| \leq c \cdot n^k$ gilt und eine TM für g maximal $d \cdot (c \cdot n^k)^\ell = dc^\ell \cdot n^{k\ell}$ Operationen ausführt. Jetzt müssen wir nur noch die Operationen beachten, die zur Berechnung von $f(w)$ benötigt wurden. Dies liefert uns im Worst-case eine Gesamtlaufzeit von $dc^\ell \cdot n^{k\ell} + c \cdot n^k$. Dies ist jedoch ein Polynom in n.

Antisymmetrie: Wir können leicht zeigen, dass $\leq_p$ nicht antisymmetrisch ist. Für zwei **NP**-vollständige Probleme A und B gilt per Definition (siehe Folie 188) $A \leq_p B$ und $B \leq_p A$. Aber es gibt **NP**-vollständige Probleme, die verschieden sind: Zum Beispiel für $A = \mathrm{CLIQUE}$ und $B = \mathrm{VERTEX\text{-}COVER}$ (Folie 191) gilt $A \neq B$.

Aufgabe 2
Zeigen Sie, dass die Klassen **P** und **NP** unter den Operationen $\cup$, $\cap$ und $\cdot$ abgeschlossen sind. Zeigen Sie ferner, dass **NP** unter dem Kleene-Stern abgeschlossen ist.

Lösung
Seien L und K in **P**. Es gibt also zwei DTMs M_L und M_K, die L und K in polynomieller Zeit entscheiden. Für Abgeschlossenheit unter $\cup$ und $\cap$ müssen wir nur beide DTMs nacheinander ausführen (das geht also wieder in deterministisch polynomieller Zeit) und überprüfen, ob beide DTMs akzeptieren (im Fall $\cap$) oder mindestens eine von beiden DTMs akzeptiert (im Fall $\cup$).
Für die Konkatenation müssen wir etwas mehr Aufwand betreiben: Ein gegebenes Wort w ist in $L \cdot K$ genau dann, wenn es eine Aufteilung $w = uv$ mit $u \in L$ und $v \in K$ gibt. Die Stelle wo u endet und v aufhört, kann aber beliebig innerhalb des Wortes w sein, also gibt es $n+1$ Möglichkeiten, wobei $n = |w|$ ist. Sei $p(x)$ ein Polynom, so dass M_L höchstens $p(n)$ Rechenschritte benötigt und analog sei $q(x)$ ein Polynom, so dass M_K höchstens $q(n)$

Rechenschritte benötigt (Inputgröße n). Dann können wir wie folgt eine Turingmaschine $M_{L \cdot K}$ konstruieren: Für jede der $n+1$ Möglichkeiten für $w = uv$ überprüfen wir, ob u von M_L und v von M_K akzeptiert wird. Dies benötigt höchstens Zeit $(n+1)(p(n)+q(n))$. Addition und Multiplikation von Polynomen liefert aber wieder ein Polynom.
Auch **NP** ist unter diesen drei Operationen abgeschlossen. Dank der Vorarbeit können wir das leicht begründen: Das Argument für Vereinigung und Schnitt ist exakt gleich (nur für NTMs statt DTMs) und bei der Konkatenation genügt es sogar die Zerlegung $w = uv$ nichtdeterministisch zu raten.
Es bleibt also noch zu beweisen, dass **NP** unter der Sternhülle abgeschlossen ist. Sei M_L eine NTM für L, die in polynomieller Zeit arbeitet. Eine NTM M_{L^*} für L^* kann man wie folgt konstruieren: Falls $w = \varepsilon$ ist, wird w akzeptiert. Andernfalls raten wir nichtdeterministisch eine Partition $w = w_1 \cdots w_k$ mit $1 \leq k \leq |w| = n$. Jetzt muss M_L nacheinander alle k Teilwörter durchtesten. Das Wort w wird genau dann akzeptiert, wenn es eine Partition gibt, wo jedes Teilwort akzeptiert wird. Falls die Anzahl der Schritte von M_L durch das Polynom $p(x)$ beschränkt ist, so ist die Laufzeit von M_{L^*} beschränkt durch $n \cdot p(n)$.
Bemerkung: Auch **P** ist unter der Kleene-Hülle abgeschlossen, was man mit dynamischer Programmierung (wie bei CYK-Algorithmus) zeigen kann (Übung).

Aufgabe 3
Zeigen Sie die Bemerkung von Folie 183: SUBSETSUM ist in **P**, falls man den Input $t, w_1, \ldots, w_n$ unär kodiert.

Lösung
Intuition: Da die Laufzeit / Komplexität von der Inputgröße abhängt, liegen unärkodierte Probleme oft in einer kleineren Komplexitätsklasse als die entsprechenden binärkodierten Varianten. Denn jedes binärkodierte Wort ist unärkodiert exponentiell lang (man benötigt nur n Bits für die Zahl 2^n).
Jetzt zu unserem Problem: Sei $N = |t| + |w_1| + \cdots + |w_n|$. Dies ist die Inputgröße (man kann ferner $n \leq N$ fordern, also den Fall $w_i = \varepsilon$ weglassen), also ist zu zeigen, dass es in Zeit $p(N)$ möglich ist herauszufinden, ob $t = \sum_{w \in U} w$ ist für eine Teilmenge $U \subseteq \{w_1, \ldots, w_n\}$ und ein Polynom p. Dazu nutzen wir einen Ansatz der sogenannten dynamischen Programmierung (ähnlich wie beim CYK-Algorithmus). Zur Vereinfachung nehmen wir an, dass alle Inlputzahlen positiv sind.
Sei W_k die Menge $\{w_1, w_2, \ldots, w_k\}$, also die Teilmenge der $\{w_1, \ldots, w_n\}$ mit den ersten k-vielen w_i. Hierbei ist $1 \leq k \leq n$. Wir starten mit $k = 1$, also $\{w_1\}$, und berechnen jetzt alle Werte, die $\sum_{w \in U} w$, $U \subseteq W_k$ annehmen kann. Für $k = 1$ gibt es offenbar nur eine nicht-leere Teilmenge. Aber wenn wir jetzt k inkrementieren, gibt es offenbar exponentiell viele solcher Teilmengen von W_k (nämlich $2^k - 1$ Stück). Jedoch kann die geforderte Summe nur $|t|$ viele Werte $\leq |t|$ annehmen. Wir speichern uns also in einer Tabelle alle Werte $\leq |t|$, die die Summe über W_k annehmen kann.
Dies wäre immer noch sehr aufwendig, wenn wir immer wieder neu beginnen müssten. Aber wir können für W_k auf Werte der Tabelle zurückgreifen, nämlich W_{k-1}. In W_{k-1} stehen jetzt also höchstens $|t|$-viele unärkodierte Werte. Ist a^d so ein Wert und ist $w_k = a^e$, so steht in W_k neben den Werten aus W_{k-1} und a^e auch der Wert a^{d+e}, wenn $d + e \leq |t|$ ist. Für jedes

k gibt es dann also maximal $|t|$ Berechnungsschritte (genauer $c \cdot |t|$ für eine Konstante c, da Zuweisungen usw. formal auch Operationen sind, die Zeit kosten).
Der Algorithmus terminiert, wenn W_n erreicht wurde oder $|t|$ irgendwann in der Tabelle steht. Unser Vorgehen liefert für den Input genau dann 1, wenn irgendwo $|t|$ vorkommt. Schlimmstenfalls durchlaufen wir also alle k und haben somit eine Schleife (Laufzeit n). Die Gesamtlaufzeit ist also höchstens $n \cdot c \cdot |t|$. Da $n \leq N$ und $|t| \leq N$ gilt, ist die Laufzeit maximal $c \cdot N^2$.

Aufgabe 4
Welche der folgenden Probleme liegen in **P**? Erklären Sie kurz warum (nicht) oder ob man keine Aussage treffen kann.

(a) Wortproblem für reguläre Sprachen

(b) Wortproblem für kontextfreie Sprachen

(c) Wortproblem für kontextsensitive Sprachen

(d) Schnittproblem für reguläre Sprachen (gegeben reguläre Ausdrücke)

(e) Schnittproblem für kontextfreie Sprachen

(f) Leerheitsproblem für kontextfreie Sprachen (gegeben kontextfreie Grammatik)

(g) 2-Färbbarkeit von Graphen

(h) 3-Färbbarkeit von Graphen

(i) Sortierungsproblem (Sortierung von beliebigen Arrays)

(j) Verallgemeinertes Schach-Problem: Kann ein Spieler auf einem $n \times n$ Schachbrett in einer gegebenen Stellung einen Sieg erzwingen?

Lösung
Wir wissen, dass (a) und (b) in **P** liegen (aus FSA). Für (a) haben wir DFAs, die das Wortproblem in Linearzeit entscheiden können (bzw. die entsprechende Simulation auf Turingmaschinen), für (b) kennen wir den CYK-Algorithmus, der in kubischer Zeit läuft (nach einem Preprocessing, der die kontextfreie Grammatik in Chomsky-Normalform umwandelt).

Wir wissen, dass (c) durch LBAs entschieden wird (FSA). Das Tape der TM ist zwar linear beschränkt, aber wir haben bereits auf Blatt 6 (BUL) gesehen, dass dies im schlimmsten Fall exponentiell viele Konfigurationen liefert. Damit ist das Wortproblem für kontextsensitive Sprachen nicht in **P**. Genauer: Es ist in **EXPTIME** (kein Klausurwissen).

Zu (d): Wir wissen, dass man reguläre Ausdrücke in Polynomialzeit in NFAs umwandeln kann. Jedoch müssen wir diese in DFAs umwandeln, was sehr ineffizient sein kann (exponentiell viele Zustände) und anschließend den Produktautomaten bilden, was ebenfalls sehr platzaufwendig ist. Ein Blick in die Literatur sagt uns, dass das Problem **PSPACE**-vollständig ist (auch nicht relevant für die Klausur). Es gilt $\mathbf{NP} \subseteq \mathbf{PSPACE}$ und wir wissen nicht einmal, ob $\mathbf{P} = \mathbf{NP}$ ist. Man kann hier also keine Aussage machen.

Folie 169: Da (e) nicht entscheidbar ist, ist das Problem nicht in **P**.

Wir haben in FSA gesehen, dass Problem (f) effizient lösbar ist. Man braucht nur für jedes Nichtterminal (in Linearzeit) zu bestimmen, ob es produktiv ist oder nicht. Ist das Startsymbol produktiv, so ist die von der Grammatik erzeugte Sprache nicht leer. Das Problem ist also ganz klar in **P**. Hätten wir die gleiche Frage für kontextsensitive Sprachen gestellt, so wäre die Antwort nein, denn auch das ist unentscheidbar (Folie 176).

Wir können für (g) einen Polynomialzeitalgorithmus angeben: Sei $G = (V, E)$ ein ungerichteter Graph. Weise einem beliebigen Knoten $v \in V$ eine Farbe zu (Farbe 0). Weise als nächstes jedem Knoten $u \in V$ eine andere Farbe zu (Farbe 1), wenn $\{u, v\} \in E$ ist. Merke mir die Menge all dieser neuen Knoten als Menge U. Als nächstes wiederhole diesen Schritt nacheinander für alle Knoten aus U, aber versuche diesmal wieder die erste Farbe 0 zuzuweisen. Hier gibt es 3 Fälle: Ist der Nachbarknoten von $u \in U$ ungefärbt, ist das kein Problem. Ist er mit Farbe 0 versehen, ist das korrekt. Sollte er bereits mit 1 eingefärbt worden sein, so gibt es keine 2-Färbung. In den ersten beiden Fällen (wo eine 2-Färbung noch möglich ist), ersetze alle Knoten aus U durch die neuen Knoten, denen im Schritt davor noch keine Farbe zugewiesen worden war und wiederhole den letzten Schritt. Wir alternieren hier also die Farben 0 und 1 solange, bis es entweder zum Abbruch kommt oder bis wir eine Zweifärbung gefunden haben. Wichtig ist, wir müssen nicht ständig für alle Knoten alle ausgehenden Kanten testen, sondern nur die noch nicht getesteten Kanten. Also ist die Laufzeit höchstens $|E| \leq |V|^2$. Graphen mit einer 2-Färbung nennt man auch *bipartit*.

Hingegen wissen wir (Folie 192), dass (h) **NP**-vollständig ist. Hier lässt sich also keine Aussage machen.

Sortieralgorithmen sind sehr effizient. Beispielsweise erhält man mit Bubblesort einen naiven Algorithmus, der in quadratischer Zeit läuft. Also ist (i) in **P**.

Zu (j): Auf einem Schachfeld polynomieller Größe gibt es zwar nur polynomiell viele Möglichkeiten für den nächsten Zug, aber um zu siegen muss man viele Züge im Voraus berechnen. Es gibt vor allem exponentiell viele Stellungen, in denen man sich zu einem Zeitpunkt befinden kann. Dies liefert exponentiell viele Möglichkeiten, die eine TM berechnen muss. Exponentielle Laufzeit bedeutet, das Problem liegt nicht in **P** (vgl. (c)).

Aufgabe 5

Wiederholen Sie die bisherigen Themen aus diesem Semester (Berechenbarkeit, (Un-)Entscheidbarkeit, Komplexitätstheorie). Notieren Sie sich ggf. Fragen!