7. Aufgabenblatt zur Vorlesung

Grundlagen der Theoretischen Informatik, SoSe 2008

(Dr. Frank Hoffmann)

Musterlösung von Manuel Jain, Benjamin Bortfeldt

Aufgabe 1 Satz von Rice (4 Punkte)

Schauen Sie sich nochmal den Satz von Rice an und begründen Sie, in welchen der folgenden Beispiele er sich anwenden lässt. Argumentieren Sie außerdem, dass die anderen aufgeführten Sprachen entscheidbar sind.

(a) $L_1 = \{\langle M \rangle |$ Startend auf dem leeren Band erreicht M eine Konfiguration mit einem gegebenen Zustand q in höchstens 5 Schritten$\}$

(b) $L_2 = \{\langle M \rangle |$ M hält nur bei endlich vielen Eingaben$\}$

(c) $L_3 = \{\langle M \rangle |$ M berechnet 0 bei Eingabe 1$\}$

(d) $L_4 = \{\langle M \rangle |$ M hat weniger als 100 Zustände und hält bei Eingabe 0$\}$

Hinweise: L_1 lässt sich mittels universeller Turing-Maschine entscheiden. Auf L_2 und L_3 lässt sich der Satz von Rice anwenden. Man mache sich insbesondere klar, dass L_4 schon regulär ist. Es gibt nur endlich viele normierte Turingmaschinen mit weniger als 100 Zuständen. Damit ist L_4 auf jeden Fall endlich und damit regulär. Welche einzelne Maschine tatsächlich dazugehört ist nicht von Belang.

Aufgabe 2 Abschlusseigenschaften von $\mathbb{P}$ und $\mathbb{NP}$ (6 Punkte)

Zeigen Sie, dass die Klasse $\mathbb{P}$ abgeschlossen ist gegen Vereinigung, Durchschnitt und Konkatenation.
Zeigen Sie, dass die Klasse $\mathbb{NP}$ abgeschlossen ist gegen Vereinigung, Durchschnitt und Kleene–Stern.

Lösung:
Für $\mathbb{P}$:

Seien $L_1, L_2 \in \mathbb{P}$, dann kann die Zeit, die zum Entscheiden von L_1 auf einer DTM benötigt wird, durch ein Polynom $p_1(n)$ und die Zeit für L_2 durch ein Polynom $p_2(n)$ mit $x \in \Sigma^*$ und $n = |x|$ abgeschätzt werden.

Vereinigung: Zu zeigen: $L_1 \cup L_2 \in \mathbb{P}$.

Für ein gegebenes Wort $w \in \Sigma^*$ muss also geprüft werden, ob es in L_1 oder L_2 liegt, bzw. in keiner der beiden Sprachen.
Man kann folgendermaßen vorgehen:

1: Prüfe, ob $w \in L_1$, falls nein gehe zu Schritt 2. Falls ja brich ab und gebe ja zurück.

2: Prüfe, ob $w \in L_2$, falls ja, brich ab und gebe ja zurück. Falls nein brich ab und gebe nein zurück.

Falls $w \in L_1$ gilt, wird abgebrochen und man benötigt $p_1(|w|)$ Zeit, also polynomielle Zeit. Falls aber $w \in L_2$ gilt, braucht man $p_1(|w|)+p_2(|w|)$ Zeit, also muss man noch zeigen, dass die Summe zweier Polynome wieder ein Polynom ist, was man aber aus MafI II weiß.

Konkatenation: Zu zeigen: $L_\circ = L_1 \circ L_2 \in \mathbb{P}$

Für ein gegebenes Wort $w \in \Sigma^*$ muss man nun alle Zerlegungen prüfen $w = uv$, da man ja nicht weiß, wo der erste Teil des Wortes, welcher zu L_1 gehören soll aufhört:
Sei $w = w_1 w_2 \dots w_n \in \Sigma^*$ gegeben, dann gehe wie folgt vor:
Für i von 0 bis n :

1: Teile w in Teilwörter $u = w_1 w_2 \dots w_i$ und $v = w_{i+1} w_{i+2} \dots w_n$. Bei $i = 0$ ist u leer, bei $i = n$ ist v leer.
2: Prüfe ob gilt: $u \in L_1 \wedge v \in L_2$. Falls ja, brich ab und gebe ja zurück. Falls nein, inkrementiere i und gehe zu Schritt 1.

Die Abbruchbedingung ist durch i=n gegeben.
Von Interesse ist der zweite Schritt. Hier wird für zwei Wörter die Zugehörigkeit zu der jeweiligen Sprache geprüft, hier ergibt sich eine Laufzeit von $p_1(|u|) + p_2(|v|) \leq p_1(|w|)+p_2(|w|)$. Wie schon bei der Vereinigung handelt es sich dabei wieder um ein Polynom. Da wir aber genau $n+1$ Unterteilungen haben können, muss das ganze nun $(n+1)$-mal durchgeführt werden, also ergibt sich als obere Schranke für die Gesamtlaufzeit $(n+1) \cdot (p_1(|w|)+p_2(|w|))$. Das ist aber wieder ein Polynom.

Durchschnitt: Zu zeigen: $L_\cap = L_1 \cap L_2 \in \mathbb{P}$

Für ein Wort $w \in \Sigma^*$ muss man nun prüfen ob gilt: $w \in L_1 \wedge w \in L_2$, ähnlich wie bei der Vereinigung. Um das zu testen, ob $w \in L_1$ brauchen wir $p_1(|w|)$ Zeit, um zu testen ob $w \in L_2$ brauchen wir $p_2(|w|)$ Zeit, also insgesamt wieder $p_1(|w|) + p_2(|w|)$ Zeit.

Für $\mathbb{NP}$:

Man erinnere sich zunächst daran, dass nichtdeterministische Turing–Maschinen Sprachen nur akzeptieren, nicht notwendigerweise entscheiden. Seien $L_1, L_2 \in \mathbb{NP}$, dann kann die Zeit, die zum Akzeptieren auf einer NTM von L_1 benötigt wird, durch ein Polynom $p_1(n)$ und die Zeit für L_2 durch ein Polynom $p_2(n)$ mit $x \in \Sigma^*$ und $n = |x|$ abgeschätzt werden.

Vereinigung: Hier kann man analog zu der Abgeschlossenheit in $\mathbb{P}$ argumentieren. Allerdings kann man ein wenig optimieren und die beiden Berechnungen „parallel“ durchführen lassen. Falls einer der beiden Berechnungszweige „ja“ antwortet, wird akzeptiert. Somit ergibt sich als Gesamtlaufzeit $\max\{p_1(n), p_2(n)\}$.

Durchschnitt: Hier kann man einfach die beiden NTM's wie folgt hintereinanderschalten. Eine NTM arbeitet bei Eingabe w zunächst wie die NTM für L_1 und

zwar bis zum Erreichen eines akzeptierenden Zustandes auf einem Berechnungszweig, danach wie die NTM für L_2 bei Eingabe w. Die Länge eines kürzesten akzeptierenden Berechnungszweiges kann durch $p_1(n)+p_2(n)$ abgeschätzt werden.

Kleene-Stern: Idee: Eine NTM kann die Zerlegung von w raten, die dazu führt, dass $w \in L_1^*$ ist und dann für einzelnen Teilwörter nachrechnen, dass sie in L_1 sind.
Genauer: Macht man sich am besten erstmal klar, wieviele Zerlegungen ein Wort der Länge n haben kann. Ein solches Wort hat als Buchstabenindizes die Menge $\{1, 2, 3, \ldots, n\}$. Wir spalten das Wort nun auf, in dem wir die Indizes angeben, an denen gesplittet werden soll. Beispielsweise steht die Menge $\{3, 6, 7\}$ für „Zerlege w in 4 Teilwörter u, v, x, y mit $u = w_1w_2$, $v = w_3w_4w_5$, $x = w_6$, $y = w_7w_8 \ldots w_n$“.
Was wir also betrachten müssen, ist die Menge aller Teilmengen $\{2, 3, \ldots, n\}$. Dabei handelt es sich um die Potenzmenge einer endlichen Menge mit $n - 1$ Elementen und die hat bekanntlich die Mächtigkeit 2^{n-1}. Hinweis: Die leere Menge entspricht der Zerlegung $w = w$. Daher gehen wir mit einer NTM wie folgt vor:

1: Die NTM „rät“ eine günstige Zerlegung für die Indizes → Voller binärer Berechnungsbaum hat nach $(n-1)$ Schritten 2^{n-1} Blattknoten, diese entsprechen den möglichen Zerlegungen.
2: Danach prüft die NTM für so einen Knoten, ob die dadurch repräsentierten Teilwörter alle in L_1 liegen. Dies kann für eine feste Zerlegung einfach sequentiell geschehen, für die einzelnen Teilwörter mit der NTM für L_1. Da es höchstens n Teilwörter gibt und jedes höchtens die Länge n, lässt sich dieser Test in $n \cdot p_1(n)$ durchführen, falls w tatsächlich in L_1^* liegt. Dies zusammen mit der Ratephase ist immer noch polynomiell.

Aufgabe 3 brute–force–Primzahltest nicht in $\mathbb{P}$ (2 Punkte)

Warum hat der Brute–Force–Primzahltest, der für Zahlen p in Binärdarstellung alle möglichen Teiler $q \leq \sqrt{p}$ testet, keine polynomielle Laufzeit?

Lösung: Hier kann man erstmal bemerken, dass die Länge der binär codierten Eingabe logarithmisch zum Wert der Eingabe wächst. Das Problem ist hier, dass man zu einer gegebenen Zahl p alle Zahlen bis $\sqrt{p}$ testen muss. Dies sind aber schlichtweg zu viele, selbst wenn man den Modulo in konstanter Zeit berechnen könnte. Der Grund hierfür ist, dass die Wurzelfunktion exponentiell schnell wächst im Verhältnis zum Logarithmus. Un da die Eingabelänge „nur“ logarithmisch wächst, erhöht sich die Anzahl der durchzuführenden Vergleiche exponentiell.

Aufgabe 4 Sprache in $\mathbb{P}$ (2 Punkte)

Zeigen Sie, dass die Sprache $\{< a, b, c, p > |\ a, b, c, p \in \mathbb{N} \text{ und } a^b \equiv c \bmod p\}$ für binär kodierte Eingabe in der Komplexitätsklasse $\mathbb{P}$ liegt.

Lösungshinweis: Man macht sich zunächst klar, dass etwa $b-1$ mit a zu multiplizieren um a^b auszurechnen nicht mehr polynomiell ist, denn die Darstellungsgröße von b in der Eingabe ist $\log_2 b$. Man kann sich aber den Algorithmus zum schnellen Potenzieren mittels wiederholten Quadrierens zu Nutze machen und dessen Verträglichkeit mit der modulo–Rechnung, findet sich in jedem Algorithmenbuch, z.B. Corman, Leiserson, Rivest auf S.829. Der Basistrick ist folgender: Wenn b eine Zweierpotenz ist, so kann man a^b durch $\log_2 b$ oftes Quadrieren von a berechnen. Damit haben wir eine Schleife, die $\log_2 b$ oft durchlaufen wird und nicht wie zuvor $b-1$ mal.

Aufgabe 5 Ein Graphenproblem (6 Punkte)

Beweisen Sie, dass das Entscheidungsproblem, ob ein ungerichteter Graph 2–färbbar ist, in der Komplexitätsklasse $\mathbb{P}$ liegt.
Beweisen Sie weiterhin, dass das Entscheidungsproblem, ob ein ungerichteter Graph 3–färbbar ist, in der Komplexitätsklasse $\mathbb{NP}$ liegt.
Hinweis: Ein Graph ist k–färbbar, wenn es eine Zuordnung von den Knoten zu k Farben gibt, so dass beliebige zwei benachbarte Knoten verschiedene Farben haben. 2–Färbbarkeit ist äquivalent zu der Eigenschaft, dass alle Kreise im Graphen gerade Länge haben.

Lösung: Wurde im Tutorium besprochen.