Prof. T. Nipkow
K. Kappelmann, J. Rädle, L. Stevens **ABGABE 20.07.2020, 23:59 (Σ 6 P.)** Technische Universität München
Lehrstuhl für Logik und Verifikation

Einführung in die Theoretische Informatik

Sommersemester 2020 – Übungsblatt Lösungsskizze 12

AUFGABE 12.1.

0,5 + 0,5 + 1 P

Im Folgenden dürfen Sie zusätzlich zu den Basisfunktionen der primitiven Rekursion die erweiterte Komposition und das erweiterte rekursive Definitionsschema benutzen. Die in der Vorlesung behandelten primitiv rekursiven Funktionen und die Funktionen aus Tutoraufgabe 12.4 können ebenfalls verwendet werden. LOOP-Programme sind nicht erlaubt.
Zeigen Sie:

(a) Wenn die Funktion $f(x, y)$ primitiv rekursiv ist, dann ist auch die Funktion $g(x, z) = \sum_{i=0}^{z} f(x, i)$ primitiv-rekursiv.

(b) Die Funktion $\mathrm{mod}(x, y)$ die den Rest der Ganzzahldivision $\frac{x}{y}$ auf natürlichen Zahlen berechnet, ist primitiv-rekursiv. (Im Fall $y = 0$ soll die Funktion 0 zurückgeben)

(c) Die Funktion

$$\begin{aligned} f(0) &= 1 \\ f(1) &= 1 \\ f(n+2) &= f(n+1) + f(n) \end{aligned}$$

ist primitiv-rekursiv. (*Hinweis:* Benutzen Sie die Cantorsche Paarungsfunktion)

Lösungsskizze

(a) Da wir erstmal nur Funktionen rekursiv auf dem ersten Argument definieren können, definieren wir zunächst die Hilfsfunktion

$$\begin{aligned} g'(0, x) &= f(x, 0) \\ g'(z+1, x) &= g'(z, x) + f(x, z+1) \end{aligned}$$

Und dann $g(x, z) = g'(z, x)$.

(b) Es gilt $\mathrm{mod}(m+1, n) = \mathrm{mod}(m, n) + 1$, außer im Fall $\mathrm{mod}(m, n) = n - 1$. Dann ist $m + 1$ ein Vielfaches von n, und somit gilt $\mathrm{mod}(m + 1, n) = 0$.

$$\begin{aligned} \mathrm{mod}(0, n) &= 0 \\ \mathrm{mod}(m+1, n) &= \mathrm{ifthen}(n, \mathrm{ifthen}(n \dot{-} (\mathrm{mod}(m, n) + 1), \mathrm{mod}(m, n) + 1, 0), 0) \end{aligned}$$

(c) Da diese Definition nicht dem Schema der primitiven Rekursion folgt, müssen wir einen Umweg gehen. Wir verwenden die Cantorsche Paarungsfunktion c um eine Funktion g anzugeben, so dass $g(n) = \langle f(n), f(n+1) \rangle$:

$$\begin{aligned} g(0) &= c(1, 1) \\ g(n+1) &= c(p_2(g(n)), p_1(g(n)) + p_2(g(n))) \end{aligned}$$

Dann können wir $f(n) = p_1(g(n))$ definieren.

AUFGABE 12.2.

1 + 1.5P

In der Vorlesung wurden einige Approximationsalgorithmen für NP-schwere Probleme gezeigt. Wir wollen in dieser Aufgabe zeigen, dass manche Probleme unter der Annahme $\mathsf{P} \neq \mathsf{NP}$ schwer zu approximieren sind. Um für ein Problem A zu zeigen, dass es keinen d-Approximationsalgorithmus geben kann, geht man folgendermaßen vor: Wir nehmen zunächst zum Widerspruch an, dass ein solcher Approximationsalgorithmus $\mathcal{A}$ existiert. Dann wählen wir ein geeignetes NP-schweres Entscheidungsproblem B, sodass wir für jede Probleminstanz b von B in polynomieller Zeit eine Probleminstanz a von A erzeugen können. Wenn wir nun $\mathcal{A}$ auf a anwenden und anhand des Lösungswerts ablesen können, ob die Antwort auf b "ja" oder "nein" ist, dann können wir B in polynomieller Zeit entscheiden. Das steht im Widerspruch zur NP-Schwere von B und damit kann es $\mathcal{A}$ nicht geben.

Wenden Sie diese Vorgehensweise an, um die folgenden Aussagen zu zeigen:

(a) Es kann keinen d-Approximationsalgorithmus mit $d < \frac{3}{2}$ für BIN PACKING geben. *Tipp:* Betrachten Sie die Reduktion von PARTITION auf BIN PACKING auf Folie 413.

(b) Wir betrachten das Problem TSP aus der Vorlesung mit der Modifikation, dass die Dreiecksungleichung nicht gilt. Zeigen Sie, dass es für kein $d \in \mathbb{N}$ einen d-Approximationsalgorithmus für TSP geben kann. *Tipp:* Betrachten Sie eine Reduktion von HAMILTONKREIS auf TSP.

Lösungsskizze

(a) Wie in der Vorlesung sei $a_1, \ldots, a_n$ die Eingabe für eine Instanz für PARTITION. Wir setzen $a_1, \ldots, a_n$ als Objekte und $b \coloneqq \Sigma_{i=1}^{n} a_i/2$ als Behältergröße und erhalten ein Instanz von BIN PACKING. Die Größe dieser Instanz ist polynomiell in der ursprünglichen PARTITION-Instanz. Angenommen es gäbe einen polynomiellen Approximationsalgorithmus $\mathcal{A}$ für BIN PACKING mit $d < \frac{3}{2}$. Aufgrund von $d < \frac{3}{2}$ muss $\mathcal{A}$ die Objekte in 2 Behälter packen, falls das möglich ist. Ansonsten verwendet $\mathcal{A}$ genau so wie die Optimallösung mindestens 3 Behälter. Damit können wir PARTITION entscheiden. Widerspruch!

(b) Sei $M \in \{0,1\}^{n \times n}$ der Eingabegraph unserer HAMILTONKREIS-Instanz. Angenommen es gibt einen polynomiellen d-Approximationsalgorithmus $\mathcal{A}$ für TSP für ein beliebiges aber festes $d \in \mathbb{N}$. Wir konstruieren den Graphen M' als eine Matrix deren Einträge überall 1 sind. Weiterhin definieren die Gewichtsfunktion

$$c_d(i,j) \coloneqq \begin{cases} 1 & \text{wenn } M_{i,j} = 1, \\ d * n + 1 & \text{sonst.} \end{cases}$$

Damit erhalten wir eine Instanz von TSP, die polynomiell in M ist. Beachte, dass d konstant im Hinblick auf die Eingabelänge ist. Falls die optimale Lösung nun nur Kanten mit Gewicht 1 verwendet, dann gibt $\mathcal{A}$ höchstens einen Wert von $d * n$ aus. Mit der Definition von $c_d(i,j)$ folgt, dass der ursprüngliche Graph M einen Hamiltonkreis enthält. Wenn die optimale Lösung hingegen eine Kante mit Gewicht $d*n+1$ verwendet, dann gibt $\mathcal{A}$ einen Wert von mindestens $d*n+1$ aus und M enthält keinen Hamiltonkreis. Wir können HAMILTONKREIS also entscheiden. Widerspruch!

AUFGABE 12.3. (*If you're lost, you can look and you will find me... time after time*) 1 + 0.5 + 0.5 + 0.5 + 0.5P

Wir führen nun Hausaufgabe 11.3 fort. Unser Ziel ist es zu zeigen, dass mehr Zeit/Speicher auch wirklich mehr Power bedeutet.

(a) Zeigen Sie: Es existieren unendlich viele, unterschiedliche Zeitkomplexitätsklassen. Verwenden Sie hierfür die Ergebnisse aus Hausaufgabe 11.3 (b).

(b) Zeigen Sie: Es existiert kein Polynom p, sodass $\mathsf{P} = \mathsf{TIME}(p(n))$ (man sagt auch, die Klasse P kollabiert nicht).

(c) (**Bonus**) In der Vorlesung wurde P unter anderem als die Menge der "leichten Probleme" bezeichnet. Diskutieren Sie diese Aussage im Licht des vorherigen Ergebnisses.

(d) (**Bonus**) Sei $f : \mathbb{N} \to \mathbb{N}$ berechenbar und total. Zeigen Sie: Es existiert eine entscheidbare Sprache $L' \notin \mathsf{SPACE}(f(n))$.

(e) (**Bonus**) Folgern Sie: Es existieren unendlich viele, unterschiedliche Speicherkomplexitätsklassen.

Lösungsskizze

(a+e) Sei $f : \mathbb{N} \to \mathbb{N}$ total, strikt monoton und $f \in \mathsf{TIME}\big(\mathcal{O}(f(n))\big) \cap \mathsf{SPACE}\big(\mathcal{O}(f(n))\big)$.[1] Beachte, dass die unter HA 11.3 (b) und HA 12.3 (d) definierten Prozeduren für L und L' dann zeigen, dass L bzw. L' in $\mathsf{TIME}(C \cdot f(n)^3)$ bzw. $\mathsf{SPACE}(C' \cdot f(n)^{f(n)^2})$ für geeignete Konstanten C, C' liegen.[2]
Sei nun $T_0 \coloneqq \mathsf{TIME}(f(n))$ und T_{i+1} die aus Anwendung von Lemmas HA 11.3 (b) auf T_i erhaltene Zeitklasse. Wir haben $T_i \subset T_{i+1}$ für alle $i \in \mathbb{N}$. Somit ist $\mathcal{T} \coloneqq \{T_i \mid i \in \mathbb{N}\}$ eine unendliche Familie aus unterschiedlichen Zeitkomplexitätsklassen.
Analoge Konstruktion liefert eine unendliche Menge an unterschiedlichen Speicherkomplexitätsklassen $\mathcal{S}$ beginnend mit $S_0 \coloneqq \mathsf{SPACE}(f(n))$.

(b) Starten wir den Prozess aus Aufgabe (a) mit einem solchen polynom p, enthält $\mathcal{T}$ eine unendliche Menge an polynomiellen Zeitkomplexitätsklassen die $\mathsf{TIME}(p(n))$ strikt enthalten. Somit folgt $\mathsf{P} \neq \mathsf{TIME}(p(n))$ für alle Polynome p.

(c) Die Aussage ist mit vorsicht zu genießen. Wenn ein Problem in P liegt, bedeutet das nicht unbedingt, dass das Problem auch in der Praxis gut lösbar ist. Insbesondere haben wir gezeigt, dass polynomielle Probleme existieren mit beliebig hohen Grad.
Allerdings sind die so konstruierten Probleme oft sehr theoretischer Natur und in der Praxis selten relevant. Die im Alltag anzutreffenden, polynomiellen Entscheidungsprozeduren besitzen oft einen kleinen Grad.

(d) Der Beweis erfolgt analog zur Hausaufgabe 11.3 (b) durch Diagonalisierung. Definiere

$$L' \coloneqq \{w \in \{0,1\}^* \mid M_w[w] \text{ akzeptiert nicht mit } \leq f(|w|) \text{ Speicher}\}.$$

[1] Diese Annahmen sind notwendig, damit f speicher- und platzkonstruierbar ist und die konstruierten TMs für L und L' $f(n)$ somit in passender Zeit und Speicher berechnen können.

[2] Jede hoch genuge Abschätzung tut hier ihren Job, aber es muss eine obere Schranke gegeben werden, damit die Sprachen auch wirklich in einer neuen, nach oben beschränkten Komplexitätsklasse liegen.

L' ist entscheidbar: Bei Eingabe w' darf $M_{w'}[w']$ höchstens $f(|w'|)$ viel Speicher verbrauchen. Somit darf $M_{w'}[w']$ nur endlich viele Konfigurationen erreichen: $\mathcal{O}\big(|\Gamma|^{f(|w'|)} \cdot |Q| \cdot f(|w'|)\big)$, wobei Γ das Bandalphabet und Q Zustandsmenge von $M_{w'}$ ist. Wir simulieren also $M_{w'}[w']$ und betrachten folgende Fälle:

(i) Sobald $M_{w'}[w']$ mehr als $f(|w'|)$ Speicher verbraucht, akzeptieren wir.
(ii) Sobald $M_{w'}[w']$ akzeptiert, lehnen wir ab.
(iii) Sobald $M_{w'}[w']$ ablehnt, akzeptieren wir.
(iv) Sobald $M_{w'}[w']$ eine Konfiguration doppelt besucht, hängt $M_{w'}$ in einer Schleife (da $M_{w'}$ deterministisch ist). $M_{w'}$ wird in diesem Fall also nicht terminieren, weswegen wir in diesem Fall akzeptieren können.

$L' \notin \mathsf{SPACE}(f(n))$: Angenommen $L' \in \mathsf{SPACE}(f(n))$. Dann gibt es eine Gödelnummer w' für eine DTM $M_{w'}$ mit $L(M_{w'}) = L'$ und $\mathsf{space}_{M_{w'}}(x) \leq f(|x|)$ für alle $x \in \Sigma^*$. Insbesondere gilt daher $\mathsf{space}_{M_{w'}}(w') \leq f(|w'|)$ (1). Nun gilt:

$$\begin{aligned} w' \in L(M_{w'}) = L' &\iff M_{w'} \text{ akzeptiert } w' \text{ und } \mathsf{space}_{M_{w'}}(w') \leq f(|w'|) && \text{(Def. } w' \in L(M_{w'}) \text{ und (1))} \\ &\iff M_{w'} \text{ akzeptiert } w' \text{ mit maximal } f(|w'|) \text{ Speicher} \\ &\iff w' \notin L' = L(M_{w'}) && \text{(Def. } L'\text{).} \end{aligned}$$

Widerspruch!

Die Zauberhafte Welt der Informatik

Die hier geführten Beweise sind sehr "grob". In der Tat kann man die Resultate um einiges verbessern und den sogenannten Zeithierarchie Satz als auch Speicherhierarchie Satz erhalten.
Desweiteren kann man die hier eingeführten Speicherkomplexitätstheorie weiterführen und beispielsweise die Klassen PSPACE und NPSPACE analog zu P und NP definieren:

- $\mathsf{PSPACE} := \bigcup_{p \text{ Polynom}} \mathsf{SPACE}(p(n))$
- $\mathsf{nspace}_M(w)$: die minimale Anzahl an verschiedenen, besuchten Zellen eines Berechnungslaufes der NTM M bei Eingabe w falls $w \in L(M)$ und 0 sonst.
- $\mathsf{NSPACE}(f(n)) := \{A \subseteq \Sigma^* \mid \exists \mathsf{NTM}\, M.\ L(M) = A \wedge \forall w \in \Sigma^*.\ \mathsf{nspace}_M(w) \leq f(|w|)\}$.
- $\mathsf{NPSPACE} := \bigcup_{p \text{ Polynom}} \mathsf{NSPACE}(p(n))$

Ähnlich zu $\mathsf{P} \subseteq \mathsf{NP}$, ist es einfach zu sehen, dass $\mathsf{PSPACE} \subseteq \mathsf{NPSPACE}$. Anders als $\mathsf{NP} \overset{?}{\subseteq} \mathsf{P}$, ist die Frage ob $\mathsf{NPSPACE} \overset{?}{\subseteq} \mathsf{PSPACE}$ überraschenderweise jedoch bereits positiv beantwortet worden! Der Satz von Savitch beweist auf elegante Art und Weise durch eine Reduktion auf ein Grapherreichbarkeitsproblem, dass $\mathsf{NPSPACE}(f(n)) \subseteq \mathsf{SPACE}(f(n)^2)$ für $f \in \mathsf{TIME}\big(\Omega(\log(n))\big) \cap \mathsf{SPACE}\big(\Omega(\log(n))\big)$ gilt.

> Es gibt nichts praktischeres als eine gute Theorie.
>
> — Ludwig Boltzmann