Institut für Theoretische Informatik
Lehrstuhl Prof. Dr. D. Wagner

Übungsblatt 4

Vorlesung Theoretische Grundlagen der Informatik im WS 20/21

Ausgabe: 23. Dezember 2020
Abgabe: 12. Januar 2021, 11:30 Uhr (digital im ILIAS)

Aufgabe 1 (3 Punkte)

In der Vorlesung wurden zwei nichtdeterministische Varianten der Turing-Maschine vorgestellt: Die klassische nichtdeterministische Turing-Maschine (NTM) hat Wahlmöglichkeiten in der Übergangsfunktion, analog zum nichtdeterministischen endlichen Automaten. Die Orakel-Turing-Maschine (OTM) lagert den Nichtdeterminismus in ein Orakelmodul aus, das zu Beginn der Berechnung ein Orakelwort vor die Eingabe schreibt. Danach wird deterministisch weitergearbeitet.

In Übung 5 haben wir die Klasse ONP definiert, die aus allen Problemen besteht, die von einer OTM in polynomialer Zeit entschieden werden können. Wir haben gezeigt, dass $\mathsf{NP} \subseteq \mathsf{ONP}$ gilt. Zeigen Sie nun: $\mathsf{ONP} \subseteq \mathsf{NP}$.

Lösung:
Wir müssen zeigen, dass für jede OTM $\mathcal{M}$ eine äquivalente NTM $\mathcal{M}'$ existiert, deren Laufzeit beschränkt ist durch ein Polynom in der Laufzeit von $\mathcal{M}$. Zu diesem Zweck modellieren wir das Orakelmodul von $\mathcal{M}$ mit Hilfe der erweiterten Übergangsfunktion einer NTM. Sei $\# \in \Gamma$ das Trennzeichen, mit dem das Orakelmodul Eingabe und Orakelwort auf dem Band trennt.

- Sei zunächst $\mathcal{M}_{\text{det}}$ der deterministische Teil von $\mathcal{M}$, also alle Zustände, Übergänge, etc., die die endliche Kontrolle betreffen. Sei s der Startzustand, Q die Zustandsmenge und δ die Übergangsfunktion von $\mathcal{M}_{\text{det}}$.
- Erweitere Q durch Hinzufügen eines neuen Zustands O (für Orakel).
- Erweitere δ wie folgt:
  - Füge für jedes $a \in \Sigma$ den Übergang (O, a, L) zu $\delta(s, a)$ hinzu.
    Dies erlaubt der NTM, in den Zustand O zu gehen.
  - Sei $\delta(O, \sqcup)$ in $\delta(O, \sqcup, L)$.
    Dies erlaubt der NTM im Zustand O, beliebig weit nach links von der Eingabe auf dem Band zu gehen.
  - Für jedes $a \in \Gamma$ sei (O, a, R) in $\delta(O, \sqcup)$.
    Dies erlaubt der NTM im Zustand O, ein beliebiges Wort in Γ^* links von der Eingabe auf das Band zu schreiben.
  - Sei $(s, \#, R)$ in $\delta(O, \sqcup)$.
    Dies erlaubt der NTM im Zustand O, das Trennzeichen direkt vor die Eingabe zu schreiben und wieder in den Startzustand s von $\mathcal{M}_{\text{det}}$ zu gehen.
- Bezeichne $\mathcal{M}'$ die so erhaltene Erweiterung von $\mathcal{M}_{\text{det}}$.

Es ist klar, dass $\mathcal{M}'$ für jede Eingabe $x \in \Sigma^*$ die Möglichkeit hat, das Orakelmodul von $\mathcal{M}$ zu imitieren und so x genau dann zu akzeptieren, wenn $x \in L(\mathcal{M})$. Außerdem ist die Laufzeit von $\mathcal{M}'$ genau die Laufzeit von $\mathcal{M}$.

Aufgabe 2 (5 Punkte)

In der Vorlesung wurde (ohne Beweis) behauptet, dass eine universelle Turing-Maschine existiert, die als Eingabe eine Gödelnummer w und ein Wort v bekommt und dann die Ausführung der Turing-Maschine T_w mit der Eingabe v simuliert. In dieser Aufgabe sollen Sie konkret die Arbeitsweise einer solchen universellen Turing-Maschine T_U beschreiben. Ihre universelle Turing-Maschine darf mehrere Bänder verwenden. Der Einfachheit halber können Sie davon ausgehen, dass ein festes Bandalphabet Γ vorgegeben ist, das sowohl T_U als auch alle von T_U simulierten Turing-Maschinen verwenden. Die Laufzeit Ihrer Simulation ist unerheblich.

Ihre Beschreibung von T_U sollte mindestens folgende Fragen beantworten:

- Wieviele Bänder hat T_U und wofür werden sie benutzt?
- Wie repräsentiert T_U die aktuelle Bandbeschriftung von T_w?
- Wie repräsentiert T_U den aktuellen Zustand der endlichen Kontrolle von T_w?
- Wie repräsentiert T_U die aktuelle Kopfposition von T_w?
- Wie identifiziert T_U den passenden Übergang von T_w für den aktuellen Zustand und die aktuelle Bandbeschriftung?
- Wie führt T_U diesen Übergang aus, d.h. wie aktualisiert sie Bandbeschriftung, Kopfposition und Zustand?

Lösung:
T_U benutzt 4 Bänder. Auf Band 1 steht die aktuelle Bandbeschriftung von T_w; der dazugehörige Kopf markiert die Kopfposition von T_w. Auf Band 2 steht die Gödelnummer w. Auf Band 3 steht unär kodiert der aktuelle Zustand von T_w, d.h. wenn T_w sich im Zustand q_i befindet, stehen auf Band 3 i Nullen. Auf Band 4 steht unär kodiert das aktuell von T_w gelesene Zeichen (also das Zeichen, auf dem Kopf 1 aktuell steht), d.h. wenn T_w das Zeichen a_j liest, stehen auf Band 4 j Nullen.

Stehe also $\mathtt{0}^i$ auf Band 3 und $\mathtt{0}^j$ auf Band 4. Dann muss Kopf 2 in der Gödelnummer den Übergang $\delta(q_i, a_j)$ finden. Gehe dazu von links nach rechts die Gödelnummer auf Band 2 durch. Prüfe für jeden Übergang, ob er mit $\mathtt{0}^i\mathtt{1}\mathtt{0}^j$ anfängt. Dazu muss lediglich überprüft werden, ob die Anzahl der Nullen mit der Anzahl auf Band 3 bzw. 4 übereinstimmt. Falls kein Übergang die richtige Form hat, lehne ab.

Sei ansonsten ein Übergang $\mathtt{0}^i\mathtt{1}\mathtt{0}^j\mathtt{1}\mathtt{0}^r\mathtt{1}\mathtt{0}^s\mathtt{1}\mathtt{0}^t$ gefunden. Also geht T_w in Zustand q_r über, schreibt a_s auf das Band und bewegt den Kopf in Richtung d_t. Simuliere diesen Schritt mit T_U: Kopf 1 schreibt a_s auf Band 1 und bewegt sich in Richtung d_t. Sei a_x das Zeichen, das Kopf 1 nun liest. Dann wird auf Band 4 $\mathtt{0}^j$ durch $\mathtt{0}^x$ ersetzt. Auf Band 3 wird $\mathtt{0}^i$ durch $\mathtt{0}^r$ ersetzt. Kopf 2 fährt zurück an den Anfang von w, um den Übergang für den nächsten Schritt zu suchen.

Aufgabe 3 (4 Punkte)

Die Komplexitätsklasse EXP ist definiert als die Menge aller Entscheidungsprobleme, die in deterministisch exponentieller Zeit gelöst werden können, d.h. in Zeit $\mathcal{O}(2^{n^c})$ für eine Konstante c. Analog dazu enthält die Klasse NEXP genau die Probleme, die sich in nichtdeterministisch exponentieller Zeit lösen lassen.

Wir wollen nun analog zur NP-Schwere den Begriff der NEXP-Schwere einführen. Dazu benötigen wir den Begriff der NEXP-Transformation. Eine NEXP-Transformation von einem Problem Π_1 in ein Problem Π_2 ist eine Funktion $f_{\mathsf{NEXP}} \colon D_{\Pi_1} \to D_{\Pi_2}$. Wie bei der polynomiellen Transformation fordern wir, dass eine Instanz $I \in D_{\Pi_1}$ genau dann eine Ja-Instanz von Π_1 ist, wenn $f_{\mathsf{NEXP}}(I)$ eine Ja-Instanz von Π_2 ist. Bei der polynomiellen Transformation wurde zusätzlich noch gefordert, dass f_{NEXP} in deterministisch polynomieller Zeit berechnet werden kann. In dieser Aufgabe sollen Sie untersuchen, ob und wie wir diese Forderung ändern müssen.

Falls eine NEXP-Transformation von Π_1 in Π_2 existiert, schreiben wir $\Pi_1 \propto_{\mathsf{NEXP}} \Pi_2$. Wir nennen ein Problem Π NEXP-schwer, wenn $\Pi' \propto_{\mathsf{NEXP}} \Pi$ für jedes Problem $\Pi' \in \mathsf{NEXP}$ gilt. In der Vorlesung wurde gezeigt: Wenn es ein NP-schweres Problem gibt, das in P liegt, dann gilt $\mathsf{P} = \mathsf{NP}$. Eine analoge Eigenschaft wollen wir auch für NEXP-Schwere haben: Wenn es ein NEXP-schweres Problem gibt, das in EXP liegt, dann gilt $\mathsf{EXP} = \mathsf{NEXP}$.

Betrachten Sie folgende möglichen Forderungen an f_{NEXP}:

(a) f_{NEXP} kann in deterministisch polynomieller Zeit berechnet werden.

(b) f_{NEXP} kann in deterministisch polynomiellem Platz berechnet werden.

(c) f_{NEXP} kann in deterministisch exponentieller Zeit berechnet werden können.

Für welche dieser Forderungen hat NEXP-Schwere die gewünschte Eigenschaft? Begründen Sie!

Lösung:
Für (a) und (b) hat NEXP-Schwere die gewünschte Eigenschaft. Wir zeigen dies für (b). Da aus polynomiellem Zeitbedarf auch polynomieller Platzbedarf folgt, gilt es auch für (a).

Sei Π' ein NEXP-schweres Problem, das in EXP liegt, d.h. es gibt einen deterministischen Algorithmus mit exponentiellem Zeitbedarf, der Π' löst. Dann lässt sich für jedes Problem $\Pi \in \mathsf{NEXP}$ ebenfalls ein deterministischer Algorithmus mit exponentiellem Zeitbedarf angeben: Betrachte eine Π-Instanz I mit $|I| = n$. Da $\Pi \propto_{\mathsf{NEXP}} \Pi'$ gilt, lässt sich I in deterministisch polynomiellem Platz in eine äquivalente Π'-Instanz $I' = f_{\mathsf{NEXP}}(I)$ transformieren. Es gilt also $|I'| \in \mathcal{O}(n^c)$ für eine Konstante c. Da aus polynomiellem Platzbedarf exponentieller Zeitbedarf folgt, ist die Transformation in Exponentialzeit möglich. Wende nun den Algorithmus zum Lösen von Π' auf I' an. Dieser hat eine Laufzeit von $\mathcal{O}(2^{|I'|^d})$ für eine Konstante d. Die Gesamtlaufzeit ist also in $\mathcal{O}(2^{n^{cd}})$ und somit exponentiell.

Für (c) funktioniert dieses Vorgehen nicht. Hier können wir nur noch $|I'| \in \mathcal{O}(2^{n^c})$ garantieren. Das ergibt eine Gesamtlaufzeit in $\mathcal{O}(2^{2^{n^c}})$, also doppelt exponentiell.

Aufgabe 4

(2 + 1 + 2 = 5 Punkte)

Betrachten Sie folgende Funktion:

$$T_{\max}(n) = \max\left(\left\{ m \in \mathbb{N} \;\middle|\; \begin{array}{l} \text{Es gibt eine TM } \mathcal{M} \text{ mit } n \text{ Zuständen und ein } x \in \Sigma^*, \\ \text{sodass } \mathcal{M} \text{ bei Eingabe } x \text{ in } m \text{ Schritten hält.} \end{array} \right\}\right)$$

Beachten Sie, dass nicht haltende Berechnungen nicht in $T_{\max}$ einfließen. Also ist $T_{\max}$ die Länge der längsten *haltenden* Berechnung, die mit n Zuständen möglich ist.

Die (starke) Goldbach-Vermutung besagt, dass jede gerade Zahl größer als 2 die Summe zweier Primzahlen ist. Bisher konnte dies nicht bewiesen werden.

(a) Zeigen Sie, dass eine TM existiert, die genau dann hält, wenn die Goldbach-Vermutung falsch ist.

(b) Tatsächlich existiert eine solche TM $\mathcal{M}_G$, die 27 Zustände benötigt. Angenommen, $T_{\max}(27)$ wäre bekannt. Geben Sie ein Verfahren an, dass dann die Goldbach-Vermutung in endlicher Zeit beweist oder widerlegt.

(c) Zeigen Sie, dass $T_{\max}$ nicht berechenbar ist.

Lösung:

(a) Gehe in aufsteigender Reihenfolge alle geraden Zahlen $n = 4, 6, 8, \ldots$ durch. Gehe für jedes n alle Primzahlen $p \leq n/2$ durch und prüfe jeweils, ob $n - p$ prim ist. Wenn ja, gehe weiter zum nächsten n. Falls für ein n alle $p \leq n/2$ ausprobiert wurden, ohne dass ein Primzahlpaar gefunden wurde, halte. In diesem Fall ist die Goldbach-Vermutung widerlegt. Falls die Goldbach-Vermutung zutrifft, terminiert dieses Verfahren nicht.

(b) Simuliere $\mathcal{M}_G$ für $T_{\max}(27)$ Schritte. Wenn $\mathcal{M}_G$ bis dahin nicht gehalten hat, ist die Goldbach-Vermutung bewiesen. Ansonsten ist sie mit Gegenbeispiel widerlegt.

(c) Angenommen, es existiert eine TM $\mathcal{M}_T$, die $T_{\max}$ berechnet. Konstruiere eine TM $\mathcal{M}_{\mathcal{H}}$, die das Halteproblem entscheidet. Seien also eine TM $\mathcal{M}$ und eine Eingabe x gegeben. Sei k die Anzahl der Zustände von $\mathcal{M}$. Berechne $T_{\max}(k)$ mithilfe von $\mathcal{M}_T$. Simuliere dann $\mathcal{M}$ auf x für $T_{\max}(k)$ Schritte. Wenn $\mathcal{M}$ bis dahin nicht gehalten hat, hält $\mathcal{M}$ für die Eingabe x nie, also lehne ab. Ansonsten akzeptiere.

Aufgabe 5

(4 Punkte)

Für $k \in \mathbb{N}$ ist ein Graph $G = (V, E)$ k-färbbar, wenn eine Funktion $\varphi : V \to \{1, 2, \ldots, k\}$ existiert, so dass für alle $(u, v) \in E$ gilt, dass $\varphi(u) \neq \varphi(v)$. Beim Entscheidungsproblem k-Color ist gefragt, ob ein gegebener Graph G k-färbbar ist. Zeigen Sie, dass 2020-Color NP-vollständig ist. Sie dürfen benutzen, dass 3-Color NP-vollständig ist.

Lösung:
2020-Color liegt in NP, weil für jeden 2020-färbbaren Graph eine entsprechende Funktion φ vollständig angegeben werden kann und in Linearzeit überprüft werden kann, ob es sich um eine gültige Färbung handelt.

Die NP-Schwere erhalten wir durch Reduktion von 3-Color auf 2020-Color. Transformiere eine Instanz $G = (V, E)$ von 3-Color wie folgt auf eine Instanz $G' = (V', E')$ von 2020-Color. Sei $X = \{1, 2, \ldots, 2017\}$ mit $X \cap V = \emptyset$. $V' = V \cup X$ und $E' = E \cup (X \times V')$. Die Instanz G' entsteht also aus G, indem eine Clique aus 2017 Knoten, die jeweils außerdem mit allen Knoten von G verbunden sind, hinzugefügt wird. Eine 3-Färbung von G lässt sich zu einer 2020-Färbung von G' erweitern, indem die 2017 Knoten in X jeweils eine eigene Farbe erhalten. Umgekehrt muss eine 2020-Färbung von G' jedem Knoten der Clique eine andere Farbe zuweisen. Da alle Knoten der Clique mit allen Knoten aus V verbunden sind, bleiben damit für die Knoten in V nur drei Farben übrig. Die 2020-Färbung von G' beschreibt also direkt eine 3-Färbung von G.

Aufgabe 6 (2 + 2 = 4 Punkte)

Gegeben sei ein Entscheidungsproblem Π, das mindestens eine Ja-Instanz und eine Nein-Instanz hat. Wir definieren das Entscheidungsproblem Π^* wie folgt: Die Instanzen von Π^* sind Paare (I_1, I_2) von Π-Instanzen mit der Eigenschaft, dass genau eine davon eine Ja-Instanz von Π ist. Es gilt zu entscheiden, ob I_1 die Ja-Instanz ist.

Wir definieren außerdem das Entscheidungsproblem $\overline{\overline{\Pi}}$, dessen Instanzen beliebige Paare (I_1, I_2) von Π-Instanzen sind. Es soll entschieden werden, ob (I_1, I_2) eine gültige Instanz von Π^* ist, also ob genau eine der beiden Instanzen eine Ja-Instanz von Π ist.

Zeigen Sie:

(a) Wenn Π in NP liegt, dann liegt Π^* in NP $\cap$ co-NP.

(b) Wenn Π in NPC liegt, dann ist $\overline{\overline{\Pi}}$ NP-schwer.

Lösung:

(a) Ist I_1 eine Ja-Instanz von Π, so taugt ein Zeuge dafür auch als Zeuge dafür, dass (I_1, I_2) eine Ja-Instanz von Π^* ist. Ist I_1 eine Nein-Instanz von Π, so ist I_2 eine Ja-Instanz von Π, und ein Zeuge dafür taugt auch als Zeuge dafür, dass (I_1, I_2) eine Nein-Instanz von Π^* ist.

(b) Reduziere Π wie folgt auf $\overline{\overline{\Pi}}$. Sei I_1 eine Instanz von Π und sei I_2 eine feste Nein-Instanz von Π. Dann ist I_1 eine Ja-Instanz von Π genau dann, wenn (I_1, I_2) eine gültige Instanz von Π^* ist.

Aufgabe 7 (5 Punkte)

Das Set-Splitting-Problem ist wie folgt definiert: Gegeben sind eine Grundmenge S und eine Menge $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ von Teilmengen $A_i \subseteq S$. Gesucht ist eine Partition $S_0 \dot{\cup} S_1 = S$, sodass kein A_i vollständig in S_0 oder S_1 enthalten ist.

Zeigen Sie, dass SET-SPLITTING NP-vollständig ist. Nutzen Sie 3-SAT zur Reduktion.

Lösung:
Wir zeigen zunächst SET-SPLITTING $\in \mathsf{NP}$. Für eine gegebene Partition (S_0, S_1) lässt sich in Zeit $\mathcal{O}(|S|)$ überprüfen, ob $S_1 \cap S_2 = \emptyset$ und $S_1 \cup S_2 = S$. Außerdem muss für jedes A_i geprüft werden, dass $S_0 \cap A_i$ und $S_1 \cap A_i$ nicht leer sind. Das geht in Zeit $\mathcal{O}(\Sigma_{i=1}^{n} |A_i||S|)$.

Wir zeigen nun $\mathsf{3\text{-}SAT} \propto$ SET-SPLITTING. Gegeben sei eine $\mathsf{3\text{-}SAT}$-Instanz bestehend aus einer Variablenmenge U und einer Klauselmenge C. Wir konstruieren nun eine SET-SPLITTING-Instanz $(S, \mathcal{A})$. Sei $\overline{U} = \{\overline{x} \mid x \in U\}$ die Menge der negativen Literale für U. Wähle $S = U \cup \overline{U} \cup \{f\}$, wobei f ein neues Element ist. Für jede Variable $x \in U$ konstruieren wir eine Teilmenge $A_x = \{x, \overline{x}\}$. Für jede Klausel $c = (x \vee y \vee z)$ mit $x, y, z \in \overline{U} \cup U$ konstruieren wir eine Teilmenge $A_c = \{x, y, z, f\}$. Die Transformation ist offensichtlich polynomial.

Wir zeigen, dass (U, C) genau dann lösbar ist, wenn $(S, \mathcal{A})$ lösbar ist.

- $\Rightarrow$: Betrachte eine erfüllende Belegung von (U, C). Wir partitionieren S in die beiden Mengen $S_0 = \{x \in U \mid x = \mathsf{falsch}\} \cup \{\overline{x} \in \overline{U} \mid x = \mathsf{wahr}\} \cup \{f\}$ und $S_1 = \{x \in U \mid x = \mathsf{wahr}\} \cup \{\overline{x} \in \overline{U} \mid x = \mathsf{falsch}\}$. Offensichtlich gilt $S_0 \dot{\cup} S_1 = S$. Betrachte eine Variable $x \in U$ und die dazugehörige Menge $A_x = \{x, \overline{x}\}$. Eins der beiden Literale wertet zu falsch aus und ist in S_0 enthalten, das andere entsprechend in S_1. Betrachte eine Klausel $c = (x \vee y \vee z) \in C$ und die dazugehörige Menge $A_c = \{x, y, z, f\}$. Da die Klausel erfüllt ist, wertet mindestens eins der Literale x, y, z zu wahr aus und ist in S_1 enthalten, während f in S_0 enthalten ist. Also ist kein A_i vollständig in S_0 oder S_1 enthalten.
- $\Leftarrow$: Betrachte eine Lösung (S_0, S_1) von $(S, \mathcal{A})$. Sei o.B.d.A. $f \in S_0$. Betrachte eine Variable $x \in U$. Setze x auf falsch, falls $x \in S_0$, ansonsten auf wahr. Die Menge $A_x = \{x, \overline{x}\}$ erzwingt, dass x und $\overline{x}$ nicht im selben S_i enthalten sind, also sind alle Literale in S_0 mit falsch belegt und alle in S_1 mit wahr. Betrachte eine Klausel $c \in C$ und die dazugehörige Menge $A_c = \{x, y, z, f\}$. Da $f \in S_0$, muss mindestens eines der Literale x, y, z in S_1 sein. Dieses Literal wurde auf wahr gesetzt, also ist die Klausel erfüllt.