Technische Universität München
Fakultät für Informatik
Prof. Tobias Nipkow, Ph.D.
Dr. Werner Meixner, Dr. Alexander Krauss

Sommersemester 2010
Lösungsblatt 12
22. Juli 2010

Einführung in die Theoretische Informatik

Hausaufgabe 1 (4 Punkte)

Zeigen Sie:

1. Sei $\Sigma = \{0, 1\}$ und α ein regulärer Ausdruck über Σ. Dann kann man in Zeit $O(|w|)$ entscheiden, ob ein Wort $w \in \Sigma^*$ ein Teilwort v hat mit $v \in L(\alpha)$.

2. Es gibt eine Turingmaschine, die ihren Kopf nie weiter als 5 Felder von der Startposition wegbewegt, aber eine unendliche Sprache akzeptiert.

Lösungsvorschlag

1. Wir können in Abhängigkeit von α einen DFA für den Ausdruck $(0|1)^*\alpha(0|1)^*$ konstruieren. Für jedes w liefert der DFA in $|w|$ Schritten das Ergebnis.

2. Falls eine Turingmaschine nur Endzustände besitzt, dann wird jedes Wort akzeptiert.

Hausaufgabe 2 (4 Punkte)

Wahr oder falsch? Begründen Sie im Folgenden Ihre Antworten möglichst knapp!

1. Für das spezielle Halteproblem $K = \{w \in \{0, 1\}^* \mid M_w[w] \downarrow\}$ und eine beliebige Sprache A gilt: Wenn $K \cap A$ entscheidbar ist, dann ist A endlich.

2. Für jede Turingmaschine M ist die Funktion

$$f_M(x) = \begin{cases} 1 & \text{falls } M \text{ auf allen Eingaben hält} \\ 0 & \text{sonst} \end{cases}$$

   berechenbar.

3. Sei $A \subseteq \Sigma^*$. Wenn χ_A total ist, dann ist A entscheidbar.

4. Sei $P = (p_1, p_2, \ldots, p_n)$ ein Post'sches Korrespondenzproblem über einem beliebigen Alphabet Σ mit $p_i = (x_i, y_i)$ und $|x_i| = |y_i|$ für alle $i \in [1, n]$.

   Dann ist P entscheidbar!

Lösungsvorschlag

1. Falsch! Begründung: Man nehme $A = \overline{K}$. Dann ist $A \cap K = \emptyset$ entscheidbar. A ist aber nicht endlich, weil sonst K entscheidbar wäre. K ist aber bekanntlich nicht entscheidbar.

2. Wahr! Begründung: Für jede Turingmaschine M ist f_M eine konstante Funktion, die überall den Wert 0 oder überall den Wert 1 hat.

3. Falsch! Begründung: Sei $A = H_0$ mit der charakteristischen Funktion χ_A. Jede charakteristische Funktion ist total, aber A ist nicht entscheibar.

4. Wahr! Begründung: Jede Lösung muss mit einem i mit $x_i = y_i$ beginnen. Dann aber ist i selbst schon eine Lösung. Falls es ein solches i nicht gibt, dann existiert also keine Lösung. Da es nur endlich viele Paare p_i gibt, ist P entscheidbar.

Hausaufgabe 3 (4 Punkte)

Geben Sie für jede der folgenden Mengen an, ob sie entscheidbar ist oder nicht. Beweisen Sie ihre Behauptungen.

1. $L_1 = \{w \in \Sigma^* \mid \varphi_w(0) = 0\}$.

2. $L_2 = \{w \in \Sigma^* \mid \varphi_w(w) = w\}$.

3. $L_3 = \{w \in \Sigma^* \mid \varphi_0(0) = w\}$.

Lösungsvorschlag

1. L_1 ist unentscheidbar. Beweis: Man setzt $F = \{\varphi \mid \exists w \in \Sigma^*.\ \varphi = \varphi_w, \varphi(0) = 0\}$. Es gilt $F \neq \emptyset$ und es gibt Funktionen $\varphi_w \notin F$. Wegen $\varphi = \varphi_w$ sind alle Funktionen aus F berechenbar. Damit ist der Satz von Rice mit $L_1 = \{w \in \Sigma^* \mid \varphi_w \in F\}$ anwendbar und es folgt die Behauptung.

2. Rice ist nicht anwendbar, weil die Eigenschaft $\varphi_w(w) = w$ keine Eigenschaft einer Funktion ist, sondern eine Relation zwischen der Funktion φ_w und der Kodierung w ist.

   Reduktion z.B. von H_0: Für w wird eine TM konstruiert, die $M_w[\epsilon]$ simuliert und bei Halten ihre ursprüngliche Eingabe zurückgibt.

3. Trivial entscheidbar, da $L_3 = \{\varphi_0(0)\}$ eine einelementige Menge ist.

Hausaufgabe 4 (4 Punkte)

Sei $\Sigma = \{0, 1\}$. Geben Sie jeweils ein Beispiel für die folgenden Objekte an. Falls kein solches Objekt existiert, begründen Sie dies.

1. Eine Funktion $f : \Sigma^* \to \Sigma^*$, die nicht berechenbar aber total ist.
2. Eine Funktion $f : \mathbb{N} \to \mathbb{N}$, die primitiv rekursiv ist, aber deren Definitionsbereich (also $\{n \in \mathbb{N} \mid f(n) \neq \bot\}$) endlich ist.
3. Eine Funktion $f : \Sigma^* \to \Sigma^*$, die total ist und für die $\{w \in \Sigma^* \mid \varphi_w = f\}$ entscheidbar ist.
4. Eine Menge $A \subseteq \Sigma^*$, die nicht rekursiv aufzählbar ist und deren Komplement ebenfalls nicht rekursiv aufzählbar ist.

Lösungsvorschlag

1. Z.B. χ_H oder jede charakteristische Funktion einer unentscheidbaren Menge.
2. Existiert nicht, da jede primitiv rekursive Funktion total ist und somit Definitionsbereich $\mathbb{N}$ hat.
3. Man nehme eine beliebige nicht-berechenbare und totale Funktion (s.o.), denn dann ist die genannte Menge leer und somit entscheidbar.
4. Man nehme $\{w \in \Sigma^* \mid \forall v.\, \varphi_w(v) \neq \bot\}$, die Menge aller terminierenden Programme.

Hausaufgabe 5 (4 Punkte)

Zeigen Sie die folgenden Aussagen durch Reduktion:

1. $H_0 \leq \{w_1 \# w_2 \# \ldots \# w_n \mid n \geq 1, w_i \in \{0,1\}^*,\ \exists i \in [1, n].\ M_{w_i}[\epsilon]\downarrow\}$.
2. $\{w_1 \# w_2 \# \ldots \# w_n \mid n \geq 1, w_i \in \{0,1\}^*,\ \forall i \in [1, n].\ M_{w_i}[\epsilon]\downarrow\} \leq H_0$.

Lösungsvorschlag

1. Wir notieren $L_1 = \{w_1 \# w_2 \# \ldots \# w_n \mid n \geq 1, w_i \in \{0,1\}^*,\ \exists i \in [1, n].\ M_{w_i}[\epsilon]\downarrow\}$.

   Für jedes w wird eine Turingmaschine M konstruiert, die bei leerer Eingabe zunächst $M_w[\epsilon]$ simuliert, und dann einfach anhält, wenn M_w anhält. Offenbar leistet $f(w) = w'$ die gewünschte Reduktion.

2. Wir notieren $L_2 = \{w_1 \# w_2 \# \ldots \# w_n \mid n \geq 1, w_i \in \{0,1\}^*,\ \forall i \in [1, n].\ M_{w_i}[\epsilon]\downarrow\}$.

   Für jedes $w = w_1 \# w_2 \# \ldots \# w_n$ wird eine Turingmaschine M konstruiert, die bei leerer Eingabe zunächst alle $M_{w_i}[\epsilon]$ nacheinander simuliert, und dann einfach anhält, wenn alle M_{w_i} angehalten haben. Offenbar leistet $f(w) = w'$ die gewünschte Reduktion.

Vorbereitung 1

1. Ist $ntime_M$ für jede deterministische Turingmaschine M berechenbar? Begründung!
2. Sei $f : \mathbb{N} \to \mathbb{N}$. Falls $NTIME(f(n))$ eine nichtentscheidbare Sprache enthält, dann ist f nicht berechenbar. Beweis!

Lösungsvorschlag

1. Eine deterministische Turingmaschine kann man als einen Spezialfall in der Klasse der nichtdeterministischen Turingmaschinen auffassen. Deshalb ist für M auch $ntime_M$ definiert. Im Unterschied zu $time_M$ ist allerdings $ntime_M$ eine totale Funktion auf Σ^*, der „Menge der Turingmaschinen“ (oder besser gesagt, der Menge der Codes aller Turingmaschinen).

   Sei nun A eine nicht entscheidbare Menge, die von einer DTM M akzeptiert wird. Wäre $ntime_M$ berechenbar, dann gäbe es eine DTM N, die $ntime_M$ berechnet, d. h. bei jeder Eingabe stoppt, also auch im Fall $ntime_M(w) = 0$. Dies würde aber bedeuten, dass man entscheiden könnte, wann M nicht hält, und damit wäre A entscheidbar. Widerspruch.

2. Sei A eine nicht entscheidbare Sprache, die von einer NTM M akzeptiert wird, so dass $ntime_M(w) \leq f(|w|)$ für alle w gilt. Von dem vorausgegangenen Beweis wissen wir, dass $ntime_M$ nicht berechenbar ist. Nun aber lernen wir, dass es keine, auch noch so große, berechenbare Funktion f gibt, mit der man $ntime_M$ abschätzen kann.

   Nehmen wir an, es gäbe eine berechenbare Funktion f mit obiger Eigenschaft. Dann könnten wir für ein beliebiges w den Wert $f(|w|)$ berechnen und die Turingmaschine M höchstens $f(|w|)$ Schritte rechnen lassen, um die Entscheidung über $w \in A$ zu erhalten. Widerspruch.

Vorbereitung 2

Finden Sie für das Problem der 3-Färbbarkeit von Graphen (3COL), sowie für das STUNDENPLAN-Problem aus TA 3 jeweils eine lösbare und eine unlösbare Instanz.

Lösungsvorschlag

Wir begnügen uns hier mit trivialen Instanzen.
3COL: Dier vollständige Graph mit 4 Knoten (K_4) ist offensichtlich nicht 3-färbbar. Jeder Graph mit 3 Knoten hingegen schon.
STUNDENPLAN Eine Lösung gibt es z.B. immer, wenn es mehr Termine als Vorlesungen gibt. Andererseits konstruieren wir mit dem Schubfachprinzip leicht eine unlösbare Instanz, in der es k Termine, aber $k + 1$ Vorlesungen gibt, und einen Studenten, der alle Vorlesungen besuchen möchte.

Tutoraufgabe 1

1. Sei $f : \mathbb{N} \to \mathbb{N}$ berechenbar. Dann gilt für alle NTM M:
$$(\forall w \in \Sigma^*.\ ntime_M(w) \leq f(|w|)) \quad \Longrightarrow \quad ntime_M \text{ berechenbar.}$$

2. Wir betrachten die Komplexitätsklasse P. Dann gibt es für jede DTM M mit $L(M) \in \mathrm{P}$ ein Polynom p, so dass $time_M(w) \leq p(|w|)$ für alle $w \in \Sigma^*$ gilt.

   Richtig oder Falsch? Begründung!

3. Ist jede in polynomieller Zeit berechenbare Funktion $f : \mathbb{N} \to \mathbb{N}$ auch polynomiell beschränkt ($\exists$ Polynom p. $\forall n.\ f(n) \leq p(n)$)? Begründung!

4. Sei p ein Polynom und M eine DTM mit $time_M(w) \leq p(|w|)$ für alle $w \in \Sigma^*$. Konstruieren Sie einen einfachen polynomiell beschränkten Verifikator für $L(M)$.

Lösungsvorschlag

1. Zur Berechnung von $ntime_M$ dient eine Turingmaschine N, die für ein beliebiges w den Wert $f(|w|)$ berechnet und dann die Schritte von M zählt bis höchstens $f(|w|)$ erreicht wird. Falls $f(|w|)$ überschritten wird, dann wird 0 ausgegeben. Andernfalls wird die für M gezählte Schrittzahl ausgegeben.

2. Falsch! $L(M) \in P$ heißt lediglich, dass eine Turingmaschine M' existiert, die die Schrankenbedingung erfüllt. Es muß aber nicht $M = M'$ gelten.

3. Nein! Falls $f(n)$ als Strichzahl codiert wird, dann mag dies gelten. Es gilt aber nicht z.B. bei Binärdarstellung. Beispiel: $f(n) = 2^n$ f ist in linearer Zeit berechenbar, f ist aber nicht polynomiell beschränkt.

4. Die Konstruktion eines polynomiellen Verifikators ist nur für nichtdeterministische Turingmaschinen interessant. Tatsächlich gibt es für deterministische Turingmaschinen stets den folgenden, trivialen Verifikator.

   Sei $L' = \{w\#\epsilon \mid w \in L(M)\} \subseteq \{w\#c \mid w \in \Sigma^*, c \in \Gamma^*\}$.
   L' wird von einer deterministischen Turingmaschine M' in polynomieller Zeit akzeptiert, falls $L(M)$ von M in polynomieller Zeit akzeptiert wird.

Tutoraufgabe 2

Seien $A, B \in \Sigma^*$ Sprachen in NP. Zeigen Sie, dass $A \cup B$, $A \cap B$ und AB ebenfalls in NP liegen.
Überlegen Sie sich hierzu, wie geeignete Zertifikate aussehen.

Lösungsvorschlag

Da A und B in NP liegen, können wir annehmen, dass es polynomiell beschränkte Verifikatoren M_A und M_B für A bzw. B gibt. Für ein $w \in A$ gibt es also ein Zertifikat c, das "beweist", dass $w \in A$ liegt und dies in Polynomialzeit überprüft werden kann. M_A akzeptiert dann das Wort $w\#c$.
Für Vereinigung, Schnitt und Komposition müssen wir uns nun jeweils überlegen, wie Zertifikate aussehen können, und wie sie in Polynomialzeit überprüft werden können.

Vereinigung: Ein Zertifikat für $w \in A \cup B$ besteht aus einer Kennung $b \in \{0, 1\}$, die angibt, ob $w \in A$ oder $w \in B$ gezeigt werden soll. Dann folgt ein Zertifikat für die Zugehörigkeit zu der jeweiligen Menge. Der Verifikator überprüft nun, welcher der beiden Fälle vorliegt und simuliert dann den Verifikator M_A bzw. M_B.

Wir konstruieren einen Verifikator $M_\cup$, so dass

$$w\#c \in L(M_\cup) \iff (c = 0c_A \wedge w\#c_A \in L(M_A)) \vee (c = 1c_B \wedge w\#c_B \in L(M_B))$$

$M_\cup$ prüft also anhand der Kennung, um was für ein Zertifikat es sich handelt, und startet dann entweder M_A oder M_B. Damit haben wir einen polynomiellen Verifikator für $A \cup B$ konstruiert.

Schnitt Ein Zertifikat für $w \in A \cap B$ besteht aus einem Paar von Zertifikaten, eins für $w \in A$ und eins für $w \in B$. Dabei benutzen wir wieder # als Trennzeichen. Ein Verifikator für $A \cap B$ muss beide Zertifikate überprüfen:

$$w\#c \in L(M_\cap) \iff c = c_A\#c_B \wedge w\#c_A \in L(M_A) \wedge w\#c_B \in L(M_B)$$

Wiederum ist der Verifikator polynomiell beschränkt, da M_A und M_B es auch sind.

Produkt Damit ein Zertifikat für $w \in AB$ möglichst einfach zu überprüfen ist, enthält es am besten auch die Aufteilung des Wortes w in zwei Teilwörter $u \in A$ und $v \in B$. Der Verifikator $M_\times$ muss prüfen, ob diese Zerlegung tatsächlich wieder das Wort ergibt, und (anhand von Zertifikaten), ob die Teilwörter in A bzw. B liegen:

$$w\#c \in L(M_\times) \iff c = u\#v\#c_A\#c_B \wedge uv = w \wedge u\#c_A \in L(M_A) \wedge v\#c_B \in L(B)$$

Alternativ wäre es auch möglich, die Zerlegung nicht im Zertifikat mitzuliefern. Der Verifikator muss dann selbst eine passende Zerlegung finden. Auch das ist aber in Polynomialzeit möglich, da es nur $|w| + 1$ Zerlegungen gibt, die man alle durchprobieren kann.

Tutoraufgabe 3

Das STUNDENPLAN-Problem, das in der Praxis allen bekannt ist, lässt sich vereinfacht wie folgt formal beschreiben:

Gegeben: Endliche Mengen S (Studierende), V (Vorlesungen) und T (Termine) und eine Relation $R \subseteq S \times V$. Dabei bedeutet $(s, v) \in R$, dass s die Vorlesung v besuchen möchte.

Problem: Gibt es eine Abbildung $f : V \to T$, so dass alle Studierenden einen überschneidungsfreien Stundenplan haben, also

$$(s, v_1) \in R \wedge (s, v_2) \in R \wedge v_1 \neq v_2 \implies f(v_1) \neq f(v_2) \; .$$

1. Zeigen Sie, dass STUNDENPLAN in NP liegt.
2. Zeigen Sie, dass STUNDENPLAN NP-hart ist, indem sie eine polynomielle Reduktion von 3COL auf STUNDENPLAN angeben.
3. Geben Sie nun eine Reduktion von STUNDENPLAN auf SAT an, die es erlaubt, das Stundenplanproblem mit Hilfe eines SAT-Solvers zu lösen.

Lösungsvorschlag

1. Um zu zeigen, dass STUNDENPLAN in NP liegt, genügt es zu zeigen, dass es effizient überprüfbare Zertifikate gibt (vgl. Satz 5.10.). Eine solches Zertifikat ist offensichtlich die gesuchte Abbildung f, die man z.B. als geordnete Liste von Terminen (einen pro Vorlesung) kodieren kann. Wir müssen uns lediglich klarmachen, dass dieses Zertifikat in Polynomialzeit überprüft werden kann. Das ist der Fall, denn man kann z.B. zur Überprüfung die Stundenpläne aller Studierenden ($|S|$ Stundenpläne mit maximal $|V|$ Einträgen) aufstellen und auf Überschneidungen prüfen.

2. Die polynomielle Reduktion funktioniert wie folgt: Gegeben eine Instanz $G = (V, E)$ von 3COL, konstruieren wir daraus ein Stundenplanproblem (S, V', T, R) wie folgt:

$$\begin{aligned} S &= E \\ V' &= V \\ T &= \{1, 2, 3\} \\ R &= \{(\{v_1, v_2\}, v) \in S \times V \mid v \in \{v_1, v_2\}\} \end{aligned}$$

   Die Knoten des Graphen werden also zu Vorlesungen und die drei Farben zu Terminen, die man den Vorlesungen zuordnet. Für jede Kante zwischen zwei Knoten erzeugt man nun einen Studenten, der genau die beiden entprechenden Vorlesungen hören will und somit verhindert, dass sie auf denselben Termin gelegt werden.

   Die oben beschriebene Transformation ist offensichtlich in Polynomialzeit berechenbar (es handelt sich ja im Wesentlichen um eine "Umbenennung" der Konzepte). Somit ist das Graphenfärbungsproblem auf das Stundenplanproblem polynomiell reduzierbar:

$$3\text{COL} \leq_p \text{STUNDENPLAN}$$

   Da von 3COL bekannt ist, dass es NP-hart ist (wurde in der Vorlesung aber nicht bewiesen), haben wir nun gezeigt, dass STUNDENPLAN NP-hart ist, und somit (mit Teil 1) NP-vollständig.

3. Eine einfache Kodierung eines gegebenen Stundenplanproblems (S, V, T, R) in SAT ist folgende:

   - Für jede Vorlesungs-Termin-Kombination benötigen wir eine Boolesche Variable, die genau dann wahr sein soll, wenn die Vorlesung zu diesem Termin stattfindet. Wir erhalten also die Variablenmenge $\{x_v^t \mid v \in V, t \in T\}$.

- Vorlesungen sollen nicht an mehreren Terminen stattfinden, was wir durch Constraints
$$F_1 = \bigwedge_{v \in V} \bigwedge_{t \in T} \bigwedge_{\substack{t' \in T \\ t' \neq t}} \neg(x_v^t \wedge x_v^{t'})$$
ausdrücken.
- Die Einschänkungen, die sich aus den Wünschen der Studierenden ergeben, beschreiben wir mit
$$F_2 = \bigwedge_{\substack{(v_1,v_2) \in V \times V \text{ mit} \\ (s,v_1) \in R, (s,v_2) \in R \\ \text{und } v_1 \neq v_2}} \bigwedge_{t \in T} \neg(x_{v_1}^t \wedge x_{v_2}^t) \ .$$
- Schließlich müssen wir noch sicherstellen, dass jede Vorlesung auch stattfindet, denn sonst wäre der leere Stundenplan immer eine Lösung:
$$F_3 = \bigwedge_{v \in V} \bigvee_{t \in T} x_v^t$$

Aus einer Belegung, die $F = F_1 \wedge F_2 \wedge F_3$ erfüllt, kann man dann leicht einen Stundenplan ablesen. Die oben beschriebene Codierung ist offensichtlich in Polynomialzeit berechenbar.

Im Prinzip haben wir mit dieser Einbettung in SAT nur noch einmal gezeigt, dass Stundenplan $\in NP$. Im Gegensatz zu Teilaufgabe 1 haben wir nun aber auch einen Weg aufgezeigt, wie man das Problem praktisch angehen kann, nämlich mit der Hilfe von effizienten Lösern für SAT. Das ist häufig schneller und einfacher, als einen problemspezifischen Algorithmus neu zu entwickeln und zu implementieren.