Lösungsblatt 7

Aufgabe 1

Zeigen Sie: PCP_Σ ist genau dann entscheidbar, wenn Σ ein unäres Alphabet ist.

Lösung

$\underline{\mathrm{PCP}_\Sigma \text{ entscheidbar} \implies |\Sigma| = 1}$

Da ein Alphabet Σ per Definition nichtleer ist, ist diese Aussage äquivalent zu

$$|\Sigma| \geq 2 \implies \mathrm{PCP}_\Sigma \text{ unentscheidbar.}$$

Diese wurde bereits in der Vorlesung gezeigt.

$\underline{|\Sigma| = 1 \implies \mathrm{PCP}_\Sigma \text{ entscheidbar}}$

Sei $P = ((x_1, y_1), \ldots, (x_k, y_k))$ eine gegebene PCP_Σ-Instanz. Wegen $|\Sigma| = 1$ gilt für alle Indizes $i_1, \ldots, i_n \in [k]$:

$$x_{i_1} \ldots x_{i_n} = y_{i_1} \ldots y_{i_n} \iff |x_{i_1}| + \ldots + |x_{i_n}| = |y_{i_1}| + \ldots + |y_{i_n}|.$$

Also ist für eine Lösung nur die Länge der x_i und y_i relevant und nicht ihre Reihenfolge innerhalb der Wörter $x_{i_1} \ldots x_{i_n}$ und $y_{i_1} \ldots y_{i_n}$.

Wir unterscheiden zwei Fälle.

- $\underline{\text{Fall 1:}}$ $\forall i \in [k]\colon |x_i| > |y_i| \vee \forall i \in [k]\colon |x_i| < |y_i|$.

  Dann ist P unlösbar.

- $\underline{\text{Fall 2:}}$ $\exists i, j \in [k]\colon (|x_i| \geq |y_i| \wedge |x_j| \leq |y_j|)$.

  Dann gilt für $\ell = |y_j| - |x_j|$ und $m = |x_i| - |y_i|$:

  $$|x_i^\ell x_j^m| = \ell|x_i| + m|x_j| = |y_j||x_i| - |y_i||x_j| = \ell|y_i| + m|y_j| = |y_i^\ell y_j^m|.$$

  Somit ist $(\underbrace{i, \ldots, i}_{\ell \text{ mal}}, \underbrace{j, \ldots, j}_{m \text{ mal}})$ für obige ℓ und m eine Lösung für P.

Also kann $\chi_{\mathrm{PCP}_\Sigma}$ sehr leicht mit folgendem Algorithmus berechnet werden:

Eingabe: PCP_Σ-Instanz $P = ((x_1, y_1), \ldots, (x_k, y_k))$

```
if ∀i ∈ [k]: |x_i| > |y_i| ∨ ∀i ∈ [k]: |x_i| < |y_i|
    return 0
else
    return 1
```

Man sieht, dass dieser Algorithmus für jede Eingabe terminiert, von einer Turingmaschine ausgeführt werden kann und, aufgrund der obigen Überlegungen, korrekt ist. □

Aufgabe 2

Seien $\Sigma = \{a, b\}^*$ ein Alphabet, M_1 die 1-Bahnd DTM

$$M_1 = (\{0, 1, 2, 3, 4\}, \Sigma, \Sigma \cup \{m, \square\}, \delta, 0, \square, \{4\})$$

mit

q	$\delta(q, a)$	$\delta(q, b)$	$\delta(q, m)$	$\delta(q, \square)$
0	$(0, a, R)$	$(1, m, L)$		
1	$(2, m, R)$		$(1, m, L)$	
2		$(1, m, L)$	$(2, m, R)$	$(3, \square, L)$
3			$(3, m, L)$	$(4, \square, N)$

und L die von M_1 akzeptierte Sprache.

1. Geben Sie L in Mengenschreibweise an.
2. Geben Sie $\text{time}_{M_1}(x)$ für alle $x \in L$ an.
3. Geben Sie eine 2-Band-DTM M_2 für L an, sodass für alle $x \in \Sigma^*$ gilt:

   $$\text{time}_{M_2}(x) \leq |x| + 1.$$

4. Gibt es eine Mehrband-DTM M für L, sodass die Ungleichung

   $$\text{time}_M(x) \leq |x|$$

   für mindestens ein $x \in L$ gilt? Begründen Sie Ihre Antwort.

Lösung

1. $L = \{a^m b^m \mid m \geq 1\}$.

   Bemerkung: Unter

   `www.turingmachinesimulator.com/shared/yyubczjlyb`

   kann M_1 simuliert werden.

2. Für alle $x \in L$ gilt:

   $$\text{time}_{M_1}(x) = \frac{|x|}{2} + \left(\sum_{i=1}^{|x|+1} i \right) + 1 = \frac{|x|}{2} + \frac{(|x|+1)(|x|+2)}{2} + 1 = \frac{|x|^2}{2} + 2|x| + 2.$$

3. Sei $M_2 = (\{0, 1, 2, 3\}, \Sigma, \Sigma \cup \{\square\}, \delta, 0, \square, \{3\})$ mit

q	$\delta(q, a, \square)$	$\delta(q, b, \square)$	$\delta(q, \square, a)$
0	$(1, a, a, R, R)$		
1	$(1, a, \square, R, R)$	$(2, b, \square, R, L)$	
2		$(2, b, \square, R, L)$	$(3, \square, a, N, N)$

   Intuitive Bedeutung der Zustände:

0: Startzustand. Markiere (hier mit a) den Anfang des Eingabewortes auf Band 2.

1: Suche das erste b auf Band 1. Bewege den Lese-Schreib-Kopf auf Band 2 mit.

2: Suche auf Band 1 das erste Leersymbol nach dem Eingabewort. Bewege solange den Lese-Schreib-Kopf auf Band 2 nach links bis zur Markierung. Akzeptiere, falls beide Lese-Schreib-Köpfe gleichzeitig das gesuchte Zeichen finden.

3: Endzustand.

Bemerkung: Unter

`www.turingmachinesimulator.com/shared/volyiyjxlq`

kann M_2 simuliert werden.

4. Nein. Angenommen, es gäbe eine solche Turingmaschine M und ein solches Wort $x \in L$. Wegen $\text{time}_M(x) \leq |x|$ würde M in einen Endzustand gehen, ohne das erste Leersymbol nach dem Eingabewort gelesen zu haben. Somit würde M auch alle Wörter aus $\{xy \mid y \in \Sigma^+\}$ akzeptieren, obwohl sie nicht in L enthalten sind.

Aufgabe 3

Sie sind Doktorand und arbeiten gerade an Ihrer Dissertation über Komplexitätstheorie. Plötzlich reißt ein Kollege die Bürotür auf und platzt voller Entsetzen heraus:

> *„Eine Turingmaschine, die nur $n + 1$ Schritte machen darf, kann das Eingabewort nur einmal von links nach rechts durchlaufen und danach, wenn sie das erste Leersymbol nach dem Eingabewort sieht, in einem Schritt entscheiden, ob sie das Wort akzeptiert oder nicht. So ähnlich arbeiten doch endliche Automaten. Enthält dann $\mathsf{TIME}(n + 1)$ genau die regulären Sprachen?“*

Wie reagieren Sie darauf?

Lösung

Tatsächlich gilt $\mathsf{REG} \subseteq \mathsf{TIME}(n+1)$, da jeder deterministische endliche Automat zu einer Turingmaschine äquivalent ist, die den Lese-Schreib-Kopf immer nur nach rechts bewegt. Die umgekehrte Inklusion ist jedoch falsch. Ein einfaches Gegenbeispiel liefert Aufgabe 2, Teil 3: Für $L = \{a^m b^m \mid m \geq 1\}$ wurde nämlich $L \in \mathsf{TIME}(n + 1)$ gezeigt, obwohl L bekanntlich nicht regulär ist, d. h. $L \in \mathsf{TIME}(n + 1) \setminus \mathsf{REG}$.

Aufgabe 4

Sie schreiben wieder an Ihrer Dissertation, bis Ihr Kollege wieder in Ihr Büro stürmt und stammelt:

> *„Enthält vielleicht $\mathsf{TIME}(n)$ genau die regulären Sprachen?“*

Wie reagieren Sie diesmal darauf?

Lösung

Bei dieser Aussage stimmen sogar beide Richtungen der Inklusion nicht.

Gegenbeispiel für $\mathsf{TIME}(n) \not\subseteq \mathsf{REG}$

Für $L = \{a^m b^m w \mid m \geq 1 \wedge v \in \{a,b\}^*\}$ kann analog zu Aufgabe 2, Teil 3 eine 2-Band-Turingmaschine gebaut werden, die in $2m$ Schritten überprüft, ob das Eingabewort von der Form $a^m b^m w$ für ein $w \in \{a,b\}^*$ ist. Mit dem Pumping-Lemma können wir jedoch zeigen, dass L nicht regulär ist. Also ist $L \in \mathsf{TIME}(n) \setminus \mathsf{REG}$.

Gegenbeispiel für $\mathsf{REG} \not\subseteq \mathsf{TIME}(n)$

Die Sprache $L = \{a\}^*\{b\}^*$ ist bekanntlich regulär, aber kann nicht von einer deterministischen Turingmaschine M mit $\text{time}_M(x) \leq |x|$ für alle $x \in \{a,b\}^*$ akzeptiert werden. Die Argumentation hierfür verläuft analog zu der aus Aufgabe 2, Teil 4. Somit ist $L \in \mathsf{REG} \setminus \mathsf{TIME}(n)$.

Aufgabe 5

Wir erweitern die Menge der aussagenlogischen Formeln um Konstanten true und false und zweistellige Junktoren $\oplus$ (XOR) und $\barwedge$ (NAND). Dabei gilt $\mathcal{A}(\text{true}) = 1$ und $\mathcal{A}(\text{false}) = 0$ für jede Belegung $\mathcal{A}$, sowie

$$\mathcal{A}(F \oplus G) = \begin{cases} 0, & \text{falls } \mathcal{A}(F) = \mathcal{A}(G) \\ 1, & \text{falls } \mathcal{A}(F) \neq \mathcal{A}(G) \end{cases}$$

$$\mathcal{A}(F \barwedge G) = \begin{cases} 0, & \text{falls } \mathcal{A}(F) = 1 \text{ und } \mathcal{A}(G) = 1 \\ 1, & \text{falls } \mathcal{A}(F) = 0 \text{ oder } \mathcal{A}(G) = 0, \end{cases}$$

für jede Belegung $\mathcal{A}$, die für alle Variablen in F und G definiert ist. In dieser Aufgabe unterscheiden wir zwischen folgenden drei Sorten aussagenlogischer Formeln:

- *Boolesche Formeln* können Variablen und die Junktoren $\neg$, $\wedge$ und $\vee$ enthalten.
- *XOR-Formeln* können Variablen, Konstanten und den Junktor $\oplus$ enthalten.
- *NAND-Formeln* können Variablen, Konstanten und den Junktor $\barwedge$ enthalten.

Zeigen Sie:

1. Folgendes Problem liegt in P.

   XOR-SAT
   Eingabe: Eine XOR-Formel.
   Frage: Ist F erfüllbar?

2. Folgendes Problem ist NP-vollständig.

   NAND-SAT
   Eingabe: Eine NAND-Formel.
   Frage: Ist F erfüllbar?

Hinweis: Sie können davon ausgehen, dass das folgende Problem NP-vollständig ist:

SAT
Eingabe: Eine boolesche Formel.
Frage: Ist F erfüllbar?

Lösung

1. Für beliebige XOR-Formeln F, G und H können folgende Äquivalenzen sehr leicht gezeigt werden:

   (1) $((F \oplus G) \oplus H) \equiv (F \oplus (G \oplus H))$

   (2) $(F \oplus G) \equiv (G \oplus F)$

   (3) $(F \oplus F) \equiv \text{false}$

   (4) $(F \oplus \text{false}) \equiv F$

   (5) $\text{true} \oplus \text{false} \equiv \text{true}$

   Sei nun F eine gegebene XOR-Formel. Wegen (1) können die Klammern in F weggelassen werden und wegen (2) können die in F vorkommenden Variablen und Konstanten beliebig sortiert werden. Wegen (3) und (4) können Vorkommen derselben Variable oder Konstante paarweise gelöscht werden. Diesen letzten Schritt kann man so lange widerholen, bis jede Variable und jede Konstante entweder genau einmal oder gar nicht vorkommt. Sind am Schluss beide Konstanten vorhanden, so kann man mit (5) durch true ersetzen. Bleibe keine Konstante übrig, so kann man einfach nach (4) false hinzufügen. Die dadurch entstehende Formel F' hat dann die Form

   $$F' = c \oplus x_{i_1} \oplus x_{i_2} \oplus \ldots \oplus x_{i_k},$$

   für ein $k \in \mathbb{N}$ und ein $c \in \{0, 1\}$. Für k und c ergeben sich 4 Fälle:

   (i) $c = \text{false}$ und $k = 0$. Dann ist $F' = 0$ unerfüllbar.

   (ii) $c = \text{true}$ und $k = 0$. Dann ist $F' = 1$ allgemeingültig und somit erfüllbar.

   (iii) $c = \text{false}$ und $k \geq 1$. Dann ist $\mathcal{A}\colon \{x_1, x_2, \ldots\} \to \{0, 1\}$ mit

   $$\mathcal{A}(x_i) = \begin{cases} 1, & \text{falls } i = i_1 \\ 0 & \text{sonst} \end{cases}$$

   ein Modell für F'.

   (iv) $c = \text{true}$ und $k \geq 1$. Dann ist $\mathcal{A}\colon \{x_1, x_2, \ldots\} \to \{0, 1\}$ mit $\mathcal{A}(x_i) = 0$ ein Modell für F'.

   Aus diesen Überlegungen folgt, dass eine XOR-Formel F genau dann efüllbar ist, wenn true oder eine Variable in F ungerade oft vorkommt. Diese Eigenschaft kann von einer deterministischen Turingmaschine sehr leicht in polynomieller Zeit überprüft werden. □

2. NAND-SAT $\in$ NP

Mit *guess and check*: Die nichtdeterministische Turingmaschine rät eine Belegung für F und überprüft in polynomieller Zeit, dass diese ein Modell ist.

NAND-SAT ist NP-hart

Wir zeigen: SAT $\leq_p$ NAND-SAT. Für eine gegebene boolesche Formel F konstruieren wir induktiv eine äquivalente NAND-Formel $f(F)$ wie folgt:

$$f(F) = \begin{cases} F, & \text{falls } F \text{ atomar} \\ (f(G) \barwedge \text{true}), & \text{falls } F = \neg G \text{ für eine boolesche Formel } G \\ ((f(G) \barwedge f(H)) \barwedge \text{true}), & \text{falls } F = G \wedge H \text{ für boolesche Formeln } G, H \\ ((f(G) \barwedge \text{true}) \barwedge (f(H) \barwedge 1)), & \text{falls } F = G \vee H \text{ für boolesche Formeln } G, H. \end{cases}$$

$f(F)$ kann in polynomieller Zeit (in der Länge der Codierung von F) berechnet werden und es gilt:

$$F \text{ ist erfüllbar} \iff f(F) \text{ ist erfüllbar.}$$

Somit ist SATin Polynomialzeit auf NAND-SAT reduzierbar. □

Bemerkung: Die Wahl $f(F) = (f(G) \barwedge f(G))$ für den Fall $F = \neg G$, ist zwar korrekt, aber die dadurch entstehende Funktion ist im Allgemeinen nicht in polynomiller Zeit berechenbar, denn sie verdoppelt die Anzahl an Vorkommen der Teilformel $f(G)$. Betrachte beispielsweise die Formel $F = \neg\neg \ldots \neg x_1$ mit n Negationen. Dann kommt x_1 genau 2^n mal in $f(F)$ vor.