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

Übungsblatt 4

Vorlesung Theoretische Grundlagen der Informatik im WS 16/17

Ausgabe 29. November 2016
Abgabe 8. Dezember 2016, 11:00 Uhr (im Kasten im UG von Gebäude 50.34)

Bitte nutzen Sie den WebInScribe Deckblattgenerator
und heften Sie das Deckblatt an Ihr Übungsblatt.
`https://webinscribe.ira.uka.de/deckblatt/index.php?course=10588`.

Aufgabe 1 (2 Punkte)

Es seien zwei Sprachen $A, B \subset \Sigma^\star$ geben, wobei B regulär ist. Impliziert $A \propto B$ das A ebenfalls regulär ist? Ist also A eine reguläre Sprache, wenn A polynomial transformierbar in B ist?

Lösung:
Angenommen A ist regulär. Dann existiert eine Funktion $f : \Sigma^\star \to \Sigma^\star$ und eine deterministische Turing-Maschine die f berechnet, so dass für jedes Wort $w \in \Sigma^\star$, $w \in A$ genau dann gilt, wenn $f(w) \in B$.

Wir zeigen, dass es eine nicht reguläre Sprache A existiert, die polynomial transformierbar in B ist. Dazu wählen wir $A = \{0^n 1^n \mid n \in \mathbb{N}\}$ und $B = \{0^i 1^j \mid i, j \in \mathbb{N}\}$ und f wie folgt.

$$f(w) = \begin{cases} w & w \in A \\ 10 & w \notin A \end{cases}$$

Da $A \subset B$ gilt für jedes $w \in A$ offensichtlich $f(w) \in B$. Anderseits ist das Wort 10 nicht in B enthalten. Somit gilt für jedes $w \in \Sigma^\star$, $w \in A$ genau dann, wenn $f(w) \in B$.

Es bleibt zu zeigen, dass eine polynomiale deterministische Turing-Maschine $\mathcal{M}$ existiert die f berechnet. In der Vorlesung wurde eine Turing-Maschine $\mathcal{M}'$ vorgestellt, die in polynomieller Zeit entscheidet ob ein Wort in A liegt. Zur Berechnung von f kann $\mathcal{M}$ die TM $\mathcal{M}'$ simulieren. Das Wort w bleibt auf dem Band stehen, wenn $w \in A$, andernfalls schreibt $\mathcal{M}$ das Wort 10 auf das Band.

Damit ist gezeigt das f eine polynomiale Transformation von A in B ist. Da A nicht regulär ist, gilt die Implikation nicht.

Aufgabe 2 (2 + 2 + 2 = 6 Punkte)

Zeigen Sie:

(a) Die Menge der semi-entscheidbaren Sprachen ist unter dem Kleenschen Abschluss abgeschlossen. Beschreiben Sie dazu für eine semi-entscheidbare Sprache L die Funktionsweise einer deterministischen Turingmaschine $\mathcal{M}$, die $L^\star$ akzeptiert.
Hinweis: In der Übung wurde dieselbe Aussage für entscheidbare Sprachen bewiesen.

(b) Seien $L_1, L_2 \subseteq \Sigma^\star$ Sprachen mit $L_1 \cap L_2 = \emptyset$, sodass L_1^c und L_2^c jeweils semi-entscheidbar sind. Dann existiert eine entscheidbare Sprache L_3, so dass gilt $L_1 \subseteq L_3$ und $L_2 \subseteq L_3^c$.

(c) Die Sprache $L_{TM} = \{(\langle \mathcal{M}_1 \rangle, \langle \mathcal{M}_2 \rangle) \mid L(\mathcal{M}_1) = L(\mathcal{M}_2)\}$ ist unentscheidbar.
Anmerkung: Bemerken Sie den Unterschied zu Aufgabe 6 des dritten Übungsblatts.

Lösung:

(a) Sei $\mathcal{M}$ eine Turingmaschine, die L akzeptiert. Da L nur semi-entscheidbar ist, kann nun nicht einfach ein einzelne Präfixe auf Zugehörigkeit zu L prüfen, da diese Berechnung nicht notwendigerweise terminiert. Stattdessen geht die deterministische Turingmaschine $\mathcal{M}'$ folgendermaßen vor. Zunächst generiert sie alle möglichen Teilwortzerlegungen der Eingabe w, wovon endlich viele existieren. Zum Beispiel würde abc in $\{abc\}, \{a, bc\}, \{a, b, c\}, \{ab, c\}$ zerlegt werden. Für jede der entstehenden Teilwortzerlegungen wird dann der Reihe nach für jedes Teilwort x ein einziger Schritt der Turingmaschine $\mathcal{M}$ mit Eingabe x ausgeführt. Akzeptiert $\mathcal{M}$ das Teilwort x, so wird mit dem nächsten Teilwort identisch verfahren. Wurden alle Teilwörter einer Zerlegung akzeptiert, so akzeptiert $\mathcal{M}'$ die Eingabe w, denn $w \in L^\star$. Ansonsten wird die nächste Teilwortzerlegung bearbeitet. So ist also jede Teilwortzerlegung nach endlich vielen Schritten von $\mathcal{M}'$ an der Reihe.

Umgekehrt gilt, dass aus $w \in L^\star$ die Existenz einer Teilwortzerlegung folgt, so dass jedes Teilwort in L liegt. Dann wird jedes der endlich vielen Teilworten nach endlich vielen Schritten von $\mathcal{M}$ erkannt. Deshalb wird auch die Gültigkeit der Teilwortzerlegung nach endlich vielen Schritten von $\mathcal{M}'$ erkannt.

(b) Sei $\mathcal{M}_1^c$ eine Turingmaschine, die L_1^c akzeptiert, und sei analog $\mathcal{M}_2^c$ eine Turingmaschine, die L_2^c akzeptiert. Sei $\mathcal{M}$ eine Turingmaschine, die L_3 folgendermaßen entscheidet. Aus $L_1 \cap L_2 = \emptyset$ folgt $L_2 \subseteq L_1^c$. Die Turingmaschine $\mathcal{M}$ simuliert also $\mathcal{M}_1^c$, und lehnt die Eingabe ab, falls diese von $\mathcal{M}_1^c$ akzeptiert wird. Dadurch werden Wörter in L_2 von $\mathcal{M}$ abgelehnt. Außerdem folgt aus $L_1 \cap L_2 = \emptyset$ auch $L_1 \subseteq L_2^c$. Die Turingmaschine $\mathcal{M}$ simuliert also parallel[1] zu $\mathcal{M}_1^c$ außerdem $\mathcal{M}_2^c$, und akzeptiert die Eingabe, falls diese von $\mathcal{M}_2^c$ akzeptiert wird. Dadurch werden Wörter in L_1 von $\mathcal{M}$ akzeptiert. Für $w \in L_1^c \cap L_2^c$ ist es mit Hinblick auf die geforderten Eigenschaften unerheblich, ob $\mathcal{M}$ das Wort w akzeptiert oder ablehnt. Wegen $L_1^c \cup L_2^c = \Sigma^\star$ hält $\mathcal{M}$ aber in jedem Fall. Also ist $L_3 = L(\mathcal{M})$ tatsächlich eine entscheidbare Sprache mit den geforderten Eigenschaften.

(c) Angenommen, L_{TM} wäre entscheidbar. Dann existiert eine Turingmaschine $\mathcal{M}$, die L_{TM} entscheidet. Das Halteproblem $\mathcal{H} = \{wv \mid T_w \text{ hält auf der Eingabe } v\}$ ist aus der Vorlesung als unentscheidbar bekannt. Konstruiere zu einer Instanz wv eine Turingmaschine $\mathcal{M}_{wv}$, die bei Eingabe von v die Turingmaschine T_w simuliert und genau dann akzeptiert, wenn T_w stoppt. Bei allen anderen Eingaben akzeptiert $\mathcal{M}_{wv}$. Sei ferner $\mathcal{M}_\star$ eine Turingmaschine, die alle Eingaben akzeptiert. Dann entscheidet $\mathcal{M}$ bei Eingabe von $(\langle \mathcal{M}_{wv} \rangle, \langle \mathcal{M}_\star \rangle)$, ob die

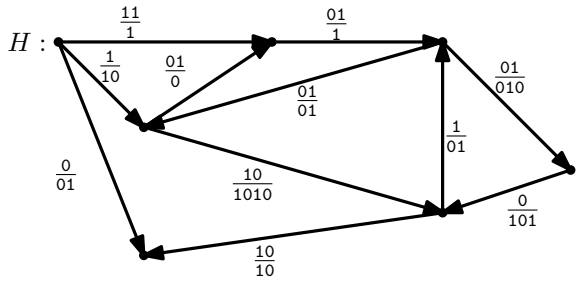
[1] Zum Beispiel durch Verwendung zweier Bänder und Köpfe.

Turingmaschine T_w auf der Eingabe v hält, was ein Widerspruch zur Unentscheidbarkeit des Halteproblems ist.

Aufgabe 3 (1 + 3 + 1 = 5 Punkte)

Der ebenso geniale wie auch erfolgreiche Wissenschaftler und Superbösewicht Doktor Meta hat Post! Elsa hat ihm auf seinen Brief geantwortet und möchte sich zu einer unkonkreten und riskanten Zeit mit ihm in seinem Hauptquartier treffen. Fieberhaft feilt Doktor Meta an der Abendplanung. Er braucht etwas, das sowohl anspruchsvoll, als auch ansprechend ist. Nach langem Grübeln fällt sein Blick auf sein Weltherrschaftsdomino©. Das Spiel besteht aus einer Menge von Dominosteinen und einem kniffligen gerichteten Graphen. Jeder Kante des Graphen ist ein Dominostein zugeordnet. Auf jedem Dominostein steht sowohl auf der oberen als auch der unteren Hälfte ein endliches Wort aus dem Alphabet Σ. Ziel des Spiels ist es, einen nichtleeren Pfad[2] zu finden, sodass die beiden Konkatenationen der Wörter auf der oberen bzw. unteren Hälfte der Dominosteine entlang des Pfads identisch sind.

(a) Nachdem Elsa eines unvorhersehbaren Abends eintrifft, ist sie begeistert von Metas Idee. Lösen sie mit ihr folgende Instanz des Weltherrschaftsdominos:



(b) Turing-Man – halb Mensch, halb Turingmaschine – versucht, das Date zu stören! Er will den beiden den Spaß verderben, indem er die Instanzen vor Elsa löst und ihr die Lösung verrät. Beweisen Sie, dass Turing-Man das Problem gar nicht entscheiden kann!

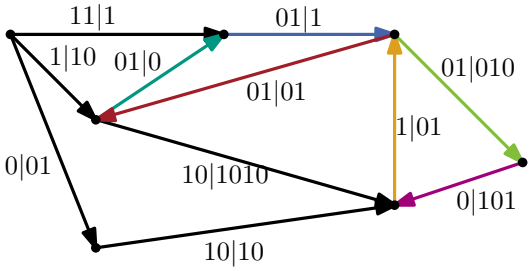
(c) Turing-Man ist sehr ausdauernd und ist sich sicher: wenn es eine Lösung gibt, dann wird er Sie finden. Hat er recht? Warum?

Lösung:

Sei Σ eine Menge von Symbolen und sei $G = (V, E)$ ein gerichteter Graph, wobei jeder Kante zwei Worte $x_e, y_e \in \Sigma^\star$ zugeordnet ist. Sei $p = \langle e_1, e_2, \dots e_k \rangle$ ein nicht notwendiger einfacher Pfad (Kanten dürfen Mehrfach besucht werden). Mit $x(p)$ und $y(p)$ bezeichnen wir die Wörter $x_{e_1} x_{e_2} \dots x_{e_k}$ bzw. $y_{e_1} y_{e_2} \dots y_{e_k}$.

[2] Der nicht einfach sein muss und auch keinen speziellen Anfangs- oder Endknoten haben muss.

(a)



(b) Wir zeigen, dass das Problem nicht-entscheidbar ist. Wir führen die Annahme, dass das Problem entscheidbar ist zum Widerspruch dazu, dass das Post'sche Korrespondenzproblem nicht-entscheidbar ist.

Angenommen, das Problem ist entscheidbar. Dann gibt es eine Turing-Maschine, die für eine beliebige Instanz entscheided, ob ein Pfad p mit $x(p) = y(p)$ existiert.

Sei $K = ((x_1, y_1), (x_2, y_2), \dots (x_n, y_n))$ eine beliebige Instanz für das Post'sche Korrespondenzproblem. Wir definieren eine Instanz G für unser Problem, die genau dann lösbar ist, wenn das Post'sche Korrespondenzproblem auf K lösbar ist.

Sei $\mathcal{K}_n$ der vollständige gerichtete Graph mit n Knoten und Schleifen. Sei E_i die Menge der ausgehenden Kanten von dem Knoten $v_i \in V$. Wir ordnen jeder Kante $e \in E_i$ das Tupel (x_i, y_i) zu.

Sei $i_1, i_2, \dots, i_k$ die Folge von Indizes mit $x_{i_1} \dots x_{i_k} = y_{i_1} \dots y_{i_k}$. Dann gilt für den Pfad $p = \langle (i_1, i_2), (i_2, i_3), \dots, (i_{k-1}, i_k), (i_k, i_k) \rangle$, dass die Wörter $x(p)$ und $y(p)$ übereinstimmen.

Sei $p = \langle e_1 = (u_1, u_2), e_2 = (u_2, u_3), \dots, e_k = (u_{k-1}, u_k) \rangle$ ein Pfad mit $x(p) = y(p)$. Dann erfüllt die Folge $u_1, u_2, \dots, u_k$ das Post'sche Korrespondenzproblem auf K.

(c) Das Problem ist semi-entscheidbar. Es können alle Pfade einer bestimmten Länge aufgezählt werden, und zu jedem Pfad kann überprüft werden, ob er eine gültige Lösung ist. Falls eine Lösung existiert, wird sie deshalb irgendwann gefunden und der Algorithmus terminiert. (Falls keine Lösung existiert, terminiert dieser Algorithmus jedoch nicht notwendigerweise).

Aufgabe 4

(1 + 1 + 1 + 2 = 5 Punkte)

```
foreach v ∈ V do
    v.dist ← ∞
    v.pred ← null
s.dist ← 0
Q ← V
while Q ≠ ∅ do
    u ← vertex in Q with smallest dist
    Q ← Q \ {u}
    foreach v ∈ V with (u, v) ∈ E do
        dist' ← u.dist + (u, v).weight
        if v ∈ Q and dist' < v.dist
         then
            v.dist ← dist'
            v.pred ← u
p ← t
u ← t
while u ≠ null do
    u ← u.pred
    p ← u, p
return p
```

(a) Formulieren Sie für einen gerichteten, gewichteten Graphen $G = (V, E)$ und zwei Knoten $s, t \in V$ das Problem, einen kürzesten Pfad von s nach t zu finden als Optimierungsproblem, als Optimalwertproblem und als Entscheidungsproblem.

(b) Ihnen steht die Implementierung von Dijkstras Algorithmus links zur Verfügung. Welches Problem löst diese Implementierung? Wie können Sie damit die beiden anderen Probleme lösen?

(c) Sie möchten nun nur das Optimalwertproblem lösen. Können Sie die Implementierung von Dijkstras Algorithmus aus Teil (b) vereinfachen, um Laufzeit einzusparen? Begründen Sie!

(d) Sie haben nun nur einen Algorithmus gegeben, der das Optimalwertproblem löst, z.B. den aus Teil (c). Wie können Sie daraus einen Algorithmus konstruieren, der das Optimierungsproblem löst?

Lösung:

(a) *Optimierungsproblem*: Finde einen Pfad kürzester Länge von s nach t.

Optimalwertproblem: Finde die Länge eines kürzesten Pfades von s nach t.

Entscheidungsproblem: Existiert ein Pfad der Länge höchstens k von s nach t?

(b) Für beide Probleme wird zunächst das Optimierungsproblem ausgeführt. Um das Optimalwertproblem zu lösen, kann einfach die Länge des Pfades ausgegeben werden. Um das Entscheidungsproblem zu lösen, kann ausgegeben werden, ob die Länge des Pfades kleiner oder gleich dem Parameter k ist.

(c) Im Gegensatz zum Optimierungsproblem muss beim Optimalwertproblem kein Pfad berechnet werden, sondern nur dessen Länge. Man benötigt also kein Vorgängerattribut, und es muss nach Finden von t kein Pfad über die Vorgänger rekonstruiert werden.

(d) Nutze den Optimalwertalgorithmus, um die Länge k des kürzesten Pfades zwischen s und t zu finden. Wähle eine Kante e aus und führe dann den Optimalwertalgorithmus erneut auf diesem Graph aus. Gibt der Algorithmus k aus, kann die Kante e nicht auf dem kürzesten Pfad liegen. Ansonsten muss die Kante e auf dem kürzesten Pfad liegen. In diesem Fall wird ihr Gewicht auf ihr ursprüngliches Gewicht zurückgesetzt. Führe dieses Verfahren für jede Kante aus. Die Kanten des kürzesten Pfades sind dann genau die Kanten im Graph, deren Gewicht höchstens k ist.

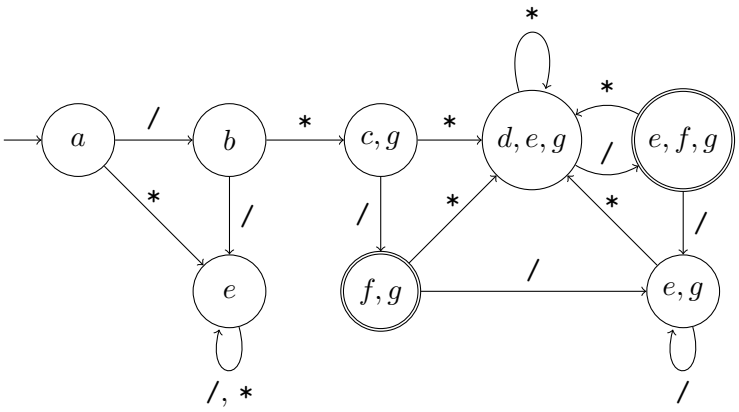
Aufgabe 5

(2 + 4 + 3 + 1 = 10 Punkte)

Ein Kommilitone von Ihnen behauptet, dass er ein alternatives Verfahren zur Konstruktion von Äquivalenzklassenautomaten gefunden hat. Statt nach möglicherweise langen Zeugen suchen zu müssen, betrachtet er immer nur einzelne Zeichen. Zu einem deterministischen endlichen Automaten $\mathcal{A} = (Q, \Sigma, \delta, s, F)$ geht er folgendermaßen vor. Im ersten Schritt partitioniert er die Zustandsmenge in zwei Mengen $Q \setminus F$ und F. In jedem weiteren Schritt wählt er zunächst ein Zeichen $a \in \Sigma$. Jede Menge[3] $[q]$ trennt er genau dann weiter auf, wenn für zwei Zustände $q_1, q_2 \in [q]$ nach dem vorherigen Schritt galt, dass $[\delta(q_1, a)] \neq [\delta(q_2, a)]$. Solch einen Schritt wiederholt er, bis sich bei keinem Zeichen $a \in \Sigma$ weitere Trennungen ergeben. Mit den entstandenen Mengen und dem Verfahren aus der Vorlesung konstruiert er dann den Äquivalenzklassenautomaten.

(a) Führen Sie das Verfahren für folgenden Automaten (vgl. Aufgabe 3 (c) des zweiten Übungsblatts) durch, indem Sie die unten stehende Tabelle ausfüllen. Finden Sie dieselben Mengen wie die Äquivalenzklassen aus Aufgabe 3 (c) des zweiten Übungsblatts?

[3] Disjunkte Mengen $\{\ldots, q, \ldots\} = [q]$ können durch einen Repräsentanten q identifiziert werden.



Lösung:

Schritt	Zeichen a	Partition nach Trennung durch a
1		$\{\{a\},\{b\},\{e\},\{c,g\},\{d,e,g\},\{e,g\}\}$, $\{\{f,g\},\{e,f,g\}\}$
2	/	$\{\{a\},\{b\},\{e\},\{e,g\}\}$, $\{\{c,g\},\{d,e,g\}\}$, $\{\{f,g\},\{e,f,g\}\}$
3	*	$\{\{a\},\{e\}\}$, $\{\{b\},\{e,g\}\}$, $\{\{c,g\},\{d,e,g\}\}$, $\{\{f,g\},\{e,f,g\}\}$
4	/	$\{\{a\}\}$, $\{\{e\}\}$, $\{\{b\}\}$, $\{\{e,g\}\}$, $\{\{c,g\},\{d,e,g\}\}$, $\{\{f,g\},\{e,f,g\}\}$
5	*	$\{\{a\}\}$, $\{\{e\}\}$, $\{\{b\}\}$, $\{\{e,g\}\}$, $\{\{c,g\},\{d,e,g\}\}$, $\{\{f,g\},\{e,f,g\}\}$
6	/	$\{\{a\}\}$, $\{\{e\}\}$, $\{\{b\}\}$, $\{\{e,g\}\}$, $\{\{c,g\},\{d,e,g\}\}$, $\{\{f,g\},\{e,f,g\}\}$

(b) Folgern Sie aus $[q_1] \neq [q_2]$ induktiv die Existenz eines Zeugen w, der q_1 und q_2 trennt.

Lösung:
Induktionsanfang: Nach dem Anfangsschritt gibt es zwei Mengen $Q \setminus F$ und F. Diese werden durch den Zeugen ε getrennt. *Induktionsannahme*: Nach dem i-ten Schritt folgt aus $[q_1] \neq [q_2]$ die Existenz eines Zeugen w, der q_1 und q_2 trennt. *Induktionsschluss*: Betrachte jetzt den Schritt $i+1$ und zwei Zustände $q_1, q_2 \in Q$ mit $[q_1] \neq [q_2]$. Galt schon im Schritt i der Zusammenhang $[q_1] \neq [q_2]$, existiert nach Induktionsannahme ein Zeuge, der q_1 und q_2 trennt. Ansonsten galt im Schritt i dass $[q_1] = [q_2]$. Das bedeutet, dass a die Zustände q_1 und q_2 im Schritt $i+1$ getrennt hat. Das ist der Fall genau dann, wenn im Schritt i galt, dass $[\delta(q_1, a)] \neq [\delta(q_2, a)]$. Nach Induktionsannahme existiert dann ein Zeuge w, der $\delta(q_1, a)$ von $\delta(q_2, a)$ trennt. Dann ist aw ein Zeuge, der im Schritt $i+1$ die Zustände q_1 und q_2 trennt.

(c) Zeigen Sie, dass am Ende des Verfahrens $[q_1] = [q_2]$ impliziert, dass kein Zeuge existiert, der q_1 und q_2 trennt.

Lösung:
Sei am Ende des Verfahrens $q_1, q_2 \in Q$ mit $[q_1] = [q_2]$. Wähle ein beliebiges Wort $w = a_1 a_2 \cdots a_k$ mit $a_i \in \Sigma$ für $1 \leq i \leq k$. Es gilt $[\delta(q_1, a_1)] = [\delta(q_2, a_1)]$, denn sonst würden die Mengen $[q_1]$ und $[q_2]$ durch a_1 weiter aufgetrennt werden. Genauso gilt $[\delta(\delta(q_1), a_1), a_2)] = [\delta(\delta(q_2, a_1), a_2)]$, denn sonst würden die Mengen $[\delta(q_1, a_1)]$ und $[\delta(q_2, a_1)]$ durch a_2 weiter aufgetrennt werden. Es folgt $[\delta(\ldots, a_k)] = [\delta(\ldots, a_k)]$. Da schon im ersten Schritt akzeptierende von nicht akzeptierenden Zuständen getrennt wurden, kann w also q_1 und q_2 nicht trennen. Da w beliebig gewählt ist kann also kein Zeuge existieren, der q_1 und q_2 trennt.

(d) Ist das Verfahren Ihres Kommilitonen korrekt?

Lösung:
Ja, denn aus (b) folgt, dass nicht-äquivalente Zustände in unterschiedlichen Mengen liegen, und aus (c) folgt, dass Zustände, die in derselben Menge liegen äquivalent sind. Das Verfahren berechnet also genau die (eindeutigen) Äquivalenzklassen und ist ansonsten identisch zu dem Verfahren aus der Vorlesung.