ATFS 2001

16.08.2001 22:14

Übungen – „HowTo"

– nicht offiziell, nicht fehlerfrei, nicht komplett! –

Die Übungen sowie die offiziellen Musterlösungen sind zu finden unter
http://www-i2.informatik.rwth-aachen.de/Vorlesung/ATFS01/

INHALTSVERZEICHNIS

Übung 1

Aufgabe 1: Induktion

a) zu zeigen: $(w^R)^R = w$

wir beginnen mit der Induktion beim leeren Wort ε, und hängen dann immer einen Buchstaben a an. Außerdem benutzen wir den Satz $\boxed{(uv)^R = v^R u^R}$.

IA: $(\varepsilon^R)^R = \varepsilon$. Jetzt nur noch „ausklammern“:
IS: $((wa)^R)^R = (a^R w^R)^R = w^{RR} a^{RR} = wa$.

b) zu zeigen: $(w^R)^n = (w^n)^R$

IA: $(w^R)^0 = (w^0)^R$
IS: $(w^R)^{n+1} = w^R (w^R)^n = w^R (w^n)^R = (w^n w)^R = (w^{n+1})^R$

Aufgabe 2: Sprachklassen

a) wenn L_1 eine Teilsprache von L_2 ist, so ist L_1^* natürlich auch eine Teilsprache von L_2^*: L_1 kann nie mehr Worte haben als L_2. Somit kann auch eine Aneinanderreihung von Worten aus L_1 immer auch in L_2 dargestellt werden.
- *formalen Beweis auf Musterlösung beachten!*

b) Beliebige Folgen von Wörtern aus den Sprachen L_1 und L_2 umfassen natürlich auch alle Folgen, die jweils nur Wörter aus L_1 bzw. L_2 verwenden.

Aufgabe 3: Reguläre Ausdrücke

Wir betrachten 0-1-Folgen

a) Wörter, in denen auf 00 immer 11 folgt:
(0011 Ú 01 Ú 1)* (0 Ú e)

b) Gerade viele Nullen oder durch 3 teilbar viele Einsen:
(1*01*01*)* Ú (0*10*10*10*)* (äquivalent zur Musterlösung)

c) Folgen von „01“ oder „10“:
(10)* Ú (01)* = $(10 \vee 01)^*$

Aufgabe 4: Sternhöhe

Sternhöhe = Verschachtelungstiefe mit Sternen („*“)

			Allgemein:
a)	((ab)* ∨ (ba)*)*	= (ab ∨ ba) *	
b)	(a(b*c)*)*	= ε ∨ a (a ∨ b*c)*	
c)	((abc)*ab)*	= ε ∨ (abc ∨ ab)*ab	(A*B)* = ε ∨ (A ∨ B) B

Bei b und c kann man entweder ε erzeugen, ansonsten MUSS der B-Teil mindestens 1x vorkommen, ansonsten beliebige Folgen von A und B.

Übung 2

Aufgabe 1: Potenzmengenautomat

ein nichtdeterministischer Automat (NFA) ist gegeben durch ein Tupel:

<Zustandsmenge, Eingabealphabet, Transitionen, Startzustand, Endzustände>

Die Transitionstafel des **Potenzmengenautomates** bekommt man folgendermaßen:

	a	B
Startzustand + alle Zustände, die man per ε erreichen kann	Wohin komme ich von den links stehenden Zustände mit a?	Wohin komme ich von den links stehenden Zustände mit b?
Hier alle Kombinationen behandeln, die einmal unter „a“ oder „b“ vorkommen.		

In der ersten Spalte stehen dann alle „Potenzzustände“ (z.B. {q0, q1}. Diese alle hingemalt, jeweils einen Kreis drum, korrekt nach Tabelle Pfeile ziehen → fertig.
- *siehe Beispiel in der Musterlösung!*

Aufgabe 2:

… ????

Aufgabe 3: DFAs konstruieren

a) geg.: 2 DFA’s, die jeweils Sprache 1 und Sprache 2 erkennen.
ges.: ein DFA, der beide Sprachen erkennt.
Ansatz: Zusammenfassen: Alle Zustände, alle Transitionen. Die Endzustände von Automat eins bekommen nun alle ε-Transitionen zum Startzustand von Automat 2. Fertig.

b) geg.: 1 DFA
ges.: DFA, der alle Präfixe erkennt (von abcd also z.B. a, ab und abc).
Ansatz: Endzustände sind jetzt alle Zustände, von denen aus man zu einem der originalen Endzustände kommt. (d.h. das Wort ließe sich so vervollständigen).

Übung 3

Aufgabe 1: Algotithmus von Thompson

Mit dem Algotithmus von Thompson kann man aus einer **RegEx**[1] einen **DFA** machen.

- $\mathfrak{A}(\Lambda) :=\longrightarrow \underset{q_0}{\bullet} \quad \circledcirc_{q_f}$
- $\mathfrak{A}(a) :=\longrightarrow \underset{q_0}{\bullet} \xrightarrow{a} \circledcirc_{q_f}$
- $\mathfrak{A}(\alpha \vee \beta) :=\longrightarrow \underset{q_0}{\bullet}$ (ε → ∘ ∼ 𝔄(α) ∼ ◎ → ε; ε → ∘ ∼ 𝔄(β) ∼ ◎ → ε) $\circledcirc_{q_f}$
- $\mathfrak{A}(\alpha\beta) :=\longrightarrow \underset{q_0}{\bullet} \sim \mathfrak{A}(\alpha) \sim \circ \sim \mathfrak{A}(\beta) \sim \circledcirc_{q_f}$ (*Beachte:* q_0 Quelle, q_f Senke)
- $\mathfrak{A}(\alpha^*) :=\longrightarrow \underset{q_0}{\bullet} \xrightarrow{\varepsilon} \circ \sim \mathfrak{A}(\alpha) \sim \circledcirc \xrightarrow{\varepsilon} \circledcirc_{q_f}$ (ε über ∘ ∼ 𝔄(α) ∼ ◎; ε unter ◎ → ∘)

Auf deutsch:

1.) Bei „oder“ einfach verzweigen
2.) Bei „hintereinanderhängen“ einfach die beiden Automaten hintereinanderhängen
3.) Bei „Stern“ die Möglichkeit des Rücksprungs geben.

ACHTUNG: Beim Zusammenfügen von Automaten IMMER beachten, dass man vorher und nachher einen ε-Schritt macht, um zu verhinden, dass man ggf. wieder zurück gehen kann !!

Aufgabe 2: W_{ij}^k - Mengen

W_{ij}^k = Die Wege[2], vom Zustand i zum Zustand j zu kommen, wobei man dazwischen höchstens die Zustände 0 bis k besuchen darf. Beispiel:
W_{45}^3 = Alle Wege von q_4 nach q_5, wobei die Zustände 0, 1, 2 und 3 auf dem Weg besucht werden dürfen.
Diese kann man sich jetzt rekursiv aufbauen. Beispiel bei 4 Zuständen:
$\boxed{W_{ab}^0 \rightarrow W_{ab}^1 = W_{ab}^0 \cup W_{a1}^0 (W_{11}^0)^* W_{1b}^0}$: von a nach 1, dort belibig bleiben, dann nach b.
$\boxed{W_{ab}^2 \rightarrow W_{ab}^2 = W_{ab}^1 \cup W_{a2}^1 (W_{22}^1)^* W_{2b}^1}$: von a nach 2, dort belibig bleiben, dann nach b.
Hier darf jetzt jeweils der Zustand 1 schon mitbenutzt werden!
Die Wege jeweils per ∨ verknüpfen, bzw. aneinanderhängen. So bekommt man $W_{\text{Anfang, Ende}}^{\text{alle}}$

[1] RegEx = Regular Expression = regulärer Ausdruck
[2] Wege = Die Beschriftungen aller gegangenen Wege aneinandergehängt.

Aufgabe 3: Pumping-Lemma für reguläre Sprachen.

Mit dem Pumping-Lemma kann man zeigen, dass eine Sprache nicht regulär ist.

Reguläre Sprache = man kann sie mit einem Automaten (NFA oder sogar DFA) darstellen = man muss nicht „zählen“, bzw. Werte zwischenspeichern.

So ist z.B. $\mathbf{a^n b^n}$ *keine* reguläre Sprache.

Das Pumping-Lemma besagt:
Zu jeder regulären Sprache existiert eine Schranke k, so dass zu jedem Wort, das mindestens k Buchstaben hat, eine Zerlegung „**uvw**“ existiert, wobei

- **v** nicht leer ist
- „**uv**“ maximal k Buchstaben hat
- **uw** und $\mathbf{uv^i w}$ und auch in der Sprache liegt

Beispiel $a^n b^n$: wir wählen das Wort $a^k b^k$: damit ist n.V. $uv = a^k = a^{k-|v|}v$. Wähle nun z.B. i=0: → Widerspruch: $a^{k-|v|}b^k$ liegt nicht in der Sprache, (da v>0).

a) $\mathbf{L_1 = \{a^m b a^n b a^{m+n}\}}$: Wähle n=0: Kann nicht regulär sein, da man hier zählen müste.
Beweis: wähle das Wort $\mathbf{a^m bba^m}$:
Wähle als Schranke m: → uv=a^x, da v>0: $\mathbf{a^{|m-v|}bb^m}$ liegt in L → WIDERSPRUCH!

b) L_2 = alle Palindrome. Kann auch nicht regulär sein, da man die erste Worthälfte irgendwo „zwischenspeichern“ müsste. Beweis: Wähle das Wort: $a^i bba^i$: ,
weiter wie oben.

c) zu 1.: da das Wort kürzer als n ist, gibt es nur endlich viele Wörter → regulär.
zu 2.: die letzten n Stellen haben gleich viele a’s und b’s. Lösung: man konstruiere einen NFA: Man bleibt so lange im Zustand q_0, bis man den n’t-letzten Buchstaben hat. Ab dann hat man alle endlich vielen Möglichkeiten in je einen Pfad gepackt. (bei n=4 sind das schon 2*2*2*2 = 16 Pfade).

Übung 4

Aufgabe 1: Schnitt-Automat

a) konstruiere einen Automaten, der den Schnitt zweier Sprachen versteht.

Bilde das Kreuzprodukt aller Zustände: $\mathbf{Q \rightarrow Q_1 \acute{} Q_2}$
Neue Transitionen: $\mathbf{\delta((q^1, q^2), a) \rightarrow \delta(q^1, a), \delta(q^2, a)}$**:** wir geben das Eingabesymbol also beiden Automaten gleichzeitig zur Verarbeitung.

a) Schnittmengenautomat:
   Akzeptiere alle Zustände, in denen beide Zustände Endzustände sind.
b) Vereinigungsautomat:
   Akzeptiere alle Zustände, in denen einer der Zustände ein Endzustand ist.

Aufgabe 2: Homomorphismus, strukturelle Induktion

formeller Beweis: siehe Musterlösung.

a) Jeder Buchstabe hat ein eindeutiges Bild. Somit ist auch die Zielsprache wieder in der gleichen Sprachklasse.
b) Gegenbeispiel: Die Abbildung bildet alles auf ε ab. Dann isz z.B. $a^n b^n$ nicht in der gleichen Sprachklasse wie ε.

Aufgabe 3: Ableitungen

a) d_{wn} = es ist eine Ableitung in n Schritten möglich.
b) Sind reguläre Ausdrücke selbst regulär?
   Wir bilden die Klammern auf die Klammern ab, alles andere auf ε. Die nun entstehenden „korrekten Klammerausdrücke“ sind bekanntlich nicht regulär.

Aufgabe 4: Markierungsalgorithmus

0					
1					
2					
3					
4	X	X	X	X	
	0	**1**	**2**	**3**	**4**

0					
1	X				
2	-	X			
3	X	-	X		
4	X	X	X	X	
	0	**1**	**2**	**3**	**4**

1.) Markiere alle Zustandspaare „Endzustand – nicht Endzustand“.
2.) Betrachte alle noch nicht markierten Zustandspaare A B, und schaue, welche Zustände man von A und B (zusammen) erreichen kann. (B C). Wenn B C markiert ist, markiere auch A B. So oft durch alle Stellen durchlaufen, bis sich nichts mehr ändert. Die jetzt nicht markierten Zustandspaare sind äquivalent und können zusammengefasst werden.

Übung 5

Aufgabe 1: Ableitungsbäume

a) **Links**ableitung: zuerst das am weitesten **links** stehende Nichtterminalsymbol ersetzen.
b) **Rechts**ableitung: zuerst das am weitesten **rechts** stehende Nichtterminalsb. ersetzen.
c) Ableitungsbaum: eine einzige Ableitung als Baum.
d) Welche Sprache erzeugt G? → Erst Linksableitung erzeugen, am Besten in Greibach-Normalform. (s.u.)

Aufgabe 2: Kontextfreie Grammatiken

(wdh.: kontextfreie Grammatik = links steht nur genau ein Nichtterminalsymbol)

a) zuerst steht eine beliebige a/b-Folge, und dann stehen noch mal so viele a's, wie in der ersten Folge vorkommen:
   S → aSa | bS | ε
b) Zuerst beliebig viele a's, dann eins-bis dreimal so viele b's:
   S → aSb | aSbb | aSbbb | ε
c) s. Musterlösung

Aufgabe 3: kontextfreie und reguläre Grammatiken

a) einfach alle rechten Seiten der Produktionsregeln umdrehen – fertig. dann natürlich immer noch regulär.

Übung 6

Aufgabe 1: Eindeutigkeit einer Grammatik

eindeutig = es gibt genau eine Linksableitung (bzw. Rechtsableitung).
s. Musterlösung: zuerst in RegEx umwandeln.

Aufgabe 2:

… ?

Aufgabe 3:

3a: Leerheit

man sucht für alle Terminalsymbole einen „Weg nach oben“ zu S. Ist dieser vorhanden, kann die Grammatik nicht leer sein. (ohne Gewähr!)

3b: Prüfe, ob Wort in Sprache liegt

Das gewünschte Wort als Zustandspfad aufmalen. Dann für jedes Terminalsymbol schauen, als welche NT's sie sich darstellen lassen, und dies – wie in der Musterlösung – in den Graphen eintragen. Wenn am Ende vom Start- zum Endzustand ein Pfeil führt (es gibt also einen Weg), liegt das Wort in der Sprache.

3c: Grammatik ε-Regel-frei machen (6/3c)

Beispiel:

S → S+A | A
A → (S) | B
B → aB | ε

Wir wollen nun erreichen, dass:

1.) das Startsymbol S nie mehr auf rechten Seiten vorkommt
2.) rechts kein ε mehr steht ausser beim Startsymbol S

zu 1):
wir führen einfach eine Regel S' → S | ε ein, und definieren S' als unser neues Startsymbol.
(Das ε natürlich nur, wenn das leere Wort auch in der Sprache liegt.)
(ε darf hier ja nur genau in dieser Zeile „ S' → …“ vorkommen.)

zu 2.)

1.) wir suchen per „Rückwärtssuche“ alle Symbole, die sich irgendwie auf ε ableiten lassen. Das ist bei uns B, A, S.
2.) überall, wo ein B, A oder S auf der rechten Seite vorkommt, müssen wir den entsprechenden Ausdruck durch alle Variationen ersetzen, in denen eines der Symbole

gestrichen ist oder nicht (also durch ε ersetzt ist:)
S+A wird zu S+A | S+ | +A | +
ACHTUNG: nicht nur streichen, sondern auch die „alte Variante“ 1x drin lassen !!!

die Lösung lautet also komplett:

S’ → S | ε
S → S+A | S+ | +A | + | A
A → (S) | () | B
B → aB | a

3d: Kettenregeln entfernen (6/3d)

Eine Kettenregel ist eine Regel der Form A → B,
also links genau ein Nichtterminalsymbol, und rechts auch genau ein Nichtterminalsymbol.

Diese „Ketten“ sind sehr leicht zu entfernen: Man ersetzt einfach das „B“ durch das, worauf „B“ abgeleitet wird. Wir setzen also von unten nach oben ein:

S’ → S+A | S+ | +A | + | (S) | () | aB | a | ε
S → S+A | S+ | +A | + | (S) | () | aB | a
A → (S) | () | aB | a
B → aB | a

Beachte: (eigentlich total trivial): Wenn wir einen Regelkomplex folgender Form haben:

A → B | C | D
B→ E
C → E
D → E
E → A (wichtig – muss auch da sein, da so ein Zirkelschluss da ist):

können wir den ganzen Ausdruck weglassen und alle Variablen A,B,C,D,E z.B. durch “Z” ersetzen, A,B,C,D,E hier ja alles das gleiche sind.

3e: Umwandlung in Chomsky-Normalform (6/3e)

Chomsky bedeutet:
Links steht immer genau 1 Nichtterminalsymbol, rechts stehen entweder genau 2 Nichtterminalsymbole oder ein Terminalsymbol. Das ε darf nur bei „S → ε“ vorkommen, dann darf aber S auf keiner rechten Regelseite vorkommen.
Es gibt als ausschließlich die Regeln der Form:

A → BC (oder auch A → AB)
A → a
S → ε.

BEISPIELE:

B → aB wird zu B → AB, A → a.

B → ABC wird zu B → AX X → BC

Unser Beispiel: (wir nehmen die kettenregel-freie Version):
!!! Hier ist alles unterstrichene ein Nichtterminalsymbol !!!

S' → SY | S+ | +A | + | (Z | () | aB | a | ε
S → SY | S+ | +A | + | (Z | () | aB | a
Y –> +A
A → (Z | () | aB | a
+ → +
Z → S)
B → AB | a
a → a
S → S
(→ (
) →) - ohne Gewähr !

3f: Greibach-Normalform (6/3f)

Regeln der Form:
A → aBCD….
A → a
S → ε
d.h.,
- **links** nur 1 NTS[3],
- **rechts** ε, genau ein TS[4] und dahinter beliebig viele (oder kein) NTS.

Dies lässt sich folgendermaßen erreichen: BEISPIEL:

Geg.: A → Ax | Ay | z. Unsere Worte beginnen also mit z, und dann kommen beliebige Folgen von x und y.

Lösung: A → zT, T → xT | yT | x | y

Vorher bauten wir unsere Worte von rechts auf: zuerst beliebige x-y-Folgen, dann ein z. Jetzt machen wir es von links: Zuerst ein z, dann beliebige x-y-Folgen.
→ fertig (das war ja einfach …) ☺

[3] NTS = Nichtterminalsymbol,
[4] TS = Terminalsymbol

Aufgabe 4: Pumping-Lemma für kontextfreie Sprachen(6/4)

hiermit kann man zeigen, ob eine Sprache kontextfrei ist.

Kontextfreie Sprache = es existiert eine Grammatik, so dass auf der linken Regelseite steht genau nur ein Nichtterminalsymbol.

Das Pumping-Lemma für CFL besagt:
Zu jeder kontextfreien Sprache existiert eine Schranke k, so dass zu jedem Wort, das mindestens k Buchstaben hat, eine Zerlegung „**uvwxy**“ existiert, wobei

- **vx** nicht leer ist
- „**vwx**“ maximal k Buchstaben hat
- $\mathbf{uv^iwx^iy}$ und auch in der Sprache liegt

Beispiel $a^nb^nc^n$: wir wählen das Wort $a^kb^kc^k$:
Das soll unterteilt werden in uvwxy. Das Wort ist ja 3k Zeichen lang, und da vx nicht leer ist, ist $|uwy|<3k$. Da vwx n.V. maximal k Buchstaben hat, kann vx nicht gleicheztig ein a und ein c enthalten.

…. wer weiß hier weiter? → s@s-inf.de !

Übung 7

Aufgabe 1: Kellerautomaten (7/1)

1.) Wir befinden uns am Ende in einem Endzustand und die Kellerinschrift ist egal
2.) Am Ende ist der Keller leer aber der Zustand ist egal,
3.) Wir befinden uns am Ende in einem Endzustand und der Keller ist leer.

… any ideas? → s@s-inf.de !

Aufgabe 2: Konstruktion eines Kellerautomaten (NFA !) (7/2)

a)

Der Kellerautomat ist das Tupel:
$A_G = <$
- Zustandsmenge,
- Eingabealphabet
- Kelleralphabet
- d - Transitionen
- Startzustand
- Kellerstartsymbol
- Endzustandsmenge >

Die d-Transitionen sind folgendermaßen definiert:

d (Zustand, Eingabe, Kellerinschrift) = { (Neuer Zustand, schreibe auf Keller) }.

In der Mengenklammer können mehrere neue „Zustand-schreibe“ – Tupel stehen, da der Kellerautomat immer nichtdeterministisch[5] ist.

Anfang: das zur parsende Wort steht auf dem Eingabeband.

Wir bauen unsere Kellerautomat-Regeln folgendermaßen auf:

Gegeben: Grammatik:

A → ABCdef | GHI
B → JKL

Für alle Trminalsymbole folgende Regeln anlegen:
d(q, d, d) = {(q,e)}
d(q, e, e) = {(q,e)}

[5] nichtdeterministisch = man muss alle möglichen Pfade durchprobieren, bis ggf. einer passt.

d(q, f, f) = {(q,e)}: Diese Zeilen für alle Nichtterminalsymbole schreiben.
also: wir löschen das gleiche Zeichen vom Eingabeband und vom Kellerband.

Für alle Nichtterminalsymbole folgende Regeln anlegen:
d(q, e , A) = {(q, ABCdef), (q, GHI)}
d(q, e , B) = {(q, JKL) }: Diese Zeilen für alle rechten Seiten schreiben.
Wir lesen ein Nichtterminalsymbol (linke Seite) vom Keller, und schreiben eine der rechten Seiten auf den Keller.

d(q, e , x) = ? , falls x Terminalsymbol. (also Eingabe leer, aber Keller nicht).
d(q, e, x) = ? , falls e ungleich x. (also Eingabe + Keller nicht leer, aber Symbole verschieden).

b): siehe Musterlösung.

Übung 8

Aufgabe 1: (8/1)

Dies ist anschaulich leicht zu erklären:
Das Wort wird hier erkannt, sobald der Keller leer ist. Wir beginnen auch mit einem leeren Keller.

Wenn v = uw ist, muss nach Erkennung von u, also mitten im Wort v, der Keller leer sein. Daher kann es nicht sein, dass u und v zwar in der Sprache sind, aber nicht w:
w würde ja am Ende von u beginnen und am Ende von v aushören; an beiden Stellen ist der Keller leer, also müsste auch das Wort in der Sprache liegen. → q.e.d.

Aufgabe 2: (8/2)

… any ideas? → s@s-inf.de !

Aufgabe 3: (8/3)

Gegeben:
1.) Eine Grammatik in Chomsky-Normalform, d.h. rechts steht entweder ein Nichtterminalsymbol, 2 Terminalsymbole oder ε.
2.) Ein Wort der **Länge n**.

Frage: Ist das Wort in der Sprache?

Algorithmus:
Male eine Tabelle **n´n** und schreibe die linke Seite der Terminalsymbole auf die Diagonale (also die Nichtterminalsymbole, aus denen sich die entsprechenden Terminalsymbole ableiten lassen).
Berechne nun alle Felder über der Diagonalen wie bei Matrixmultiplikation rekursiv:

X1	X2	X3	X4	E
				Y1
				Y2
				Y3
				Y4

E = X1 Y1 + X2 Y2 + X3 Y3 + X4 Y4

X1 ist dabei das erste belegte Feld in der Reihe, Y1 ist das Feld unter dem zu berechnenden Wert E.

Aufgabe 4: (8/4)

ECFG = erweiterte reguläre Grammatik = benutze reguläre Expressions.

1.) alle ∨ durch | ersetzen.
2.) Ausklammern: if A then B (else C ∨ D) → if A then B else C ∨ if A then B else D
3.) * ersetzen: A* → B sowie B→AB | ε
4.) statt „? “ schreibe „L“. (auch ? * muss ersetzt werden wie alle anderen *-Symbole!)

Übung 9

Aufgabe 1: Typ 0- , 1-, 2-, 3- Grammatiken (9/1)

Typ 0: beliebige Grammatik. (aBcDe → dEfGh)
Typ 1: „**kontextsensitiv**“: wie Typ 0, aber links stehen weniger Buchstaben als rechts.
Typ 2: „**kontextfrei**“: links steht nur genau ein Nichtterminalsymbol.
Typ 3: „**regulär**“ / „einseitig linear“: rechts stehen beliebig viele Teminalsymbole, und immer rechts (oder wahlweise immer links) davon steht maximal ein Nichtterminalsymbol.
Also: Abcde … oder bcdeA (in einer Grammatik aber immer nur links oder rechts!)

a) Typ 2
b) im Zweifel in RegEx umwandeln und dann zurück in eine Grammatik, hier ist es natürlich noch viel einfacher (siehe Musterlösung)

Aufgabe 2: Normieren

a)
1.) Rechts alle [a] durch [A] ersetzen, und Regeln [A → a] hinzufügen.
2.) Rechts und links darf maximal ein Nichtterminalsymbol gleich sein.
aus AB → BA mache also:
AB → AB
AB → AB
AB → AB
AB → AB *- Beispiel siehe Musterlösung!*

b): … ?

Aufgabe 3: Permutationen / Kontrollsymbole

a) Erzeuge eine Typ 0 – Grammatik, die auch alle Permutationen[6] der Wörter versteht.

Idee: wir füren „Kontrollsymbole“ CDE ein, und packen CD vor und E hinter unser Wort (das am Anfang ja nur aus S besteht):

CD E

Wir dürfen jetzt D immer nach rechts schieben, bis es auf ein Nichtterminalsymbol stößt. Dann muss das erst in ein Terminalsymbol gewandelt werden, und D kann weiter nach rechts wandern:

C D .E - im ersten Bereich stehen nur Nichtterminalsymbole!

C kann jetzt beliebig durch den linken Raum wandern, und 2 benachbarte Symbole vertauschen. Wie bei „Bubble-Sort“ kann dabei jedes Symbol an jede beliebige Stelle gebracht werden. Damit kann dann jede Permutation erreicht werden.

- *Beachte das Beispiel in der Musterlösung!*

b) Typ 1- Grammatik = rechts stehen mindestens so viele Symbole wie links.

Kann hier nicht sein wegen $C_1C_2C_3 \rightarrow \varepsilon$ (*siehe Musterlösung*).

[6] Permutationen = beliebige Durcheinanderwürfelungen der Buchstaben des Wortes

Übung 10

Aufgabe 1: Turingmaschine

Notation für Turingmaschinen:
Tupel: Alter Zustand – lese Zeichen – neuer Zustand – schreibe Zeichen - Bewegung

Turingmaschine selbst:
A = <{
- Zustände,
- Eingabealphabet,
- Arbeitsalphabet
- Startzustand
- Blank (wie heisst das Blank?)
- Endzustand
- Transitionen }>

Erkannt sollen werden: a-Folgen der Länge 1, 2, 4, 8, 16, 32, 64, …

Idee für unsere TM: Ersetze jedes 2. a durch einen Strich. Breche ab, wenn hinten ein a steht. Akzeptiere das Wort, wenn nur noch am Ende genau ein a steht.

```
Aaaaaaaa
-a-a-a-a
---a---a
-------a
--------  OK
```

```
aaaaaaa
-a-a-a-  falsch
```

```
aaaaaa
-a-a-a
---a--  falsch
```

```
aaaaa
-a-a-   falsch
```

```
aaaa
-a-a
---a  OK
```

```
aaa
-a-   falsch
```

```
aa
-a  OK
```

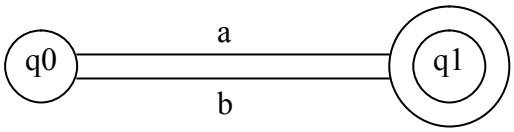
```
a  OK
```

```
Turingmaschine:

Q0 a  Q1 -  R  // erstes a durch - ersetzen
Q1 #  QE #  N  // OK!
Q1 -  Q1 -  R  // Striche nach rechts überspringen
Q1 a  Q2 a  R  // zweites a gefunden
Q2 -  Q2 -  R  // Striche nach rechts überspringen
Q2 a  Q3 a  R  // 2. a gefunden - nicht ersetzen.
Q3 -  Q3 -  R  // Striche nach rechts überspringen
Q3 a  Q2 a  R  // nach rechts

…. etc.
```

Aufgabe 2: Turingmaschine und reguläre Sprachen

zu zeigen: ein DFA lässt sich durch eine nur nach rechts gehende TM darstellen.

Gegeben: z.B. folgender Automat:



der alle die Wörter a(ba)* akzeptiert.

Daraus mache folgende TM:

q0 a q1 a R
q1 b q0 b R
q1 _ qE _ R (qE = akzeptierender Endzustand)

Die Zustände des Automaten werden also die Zustände der TM, das Eingabewort kommt vom Band.

Aufgabe 3: Nichtdeterministische Turingmaschine

gegeben: Ein Band, auf dem ganz viele stehen, und vielleicht irgendwo (rechts oder links von der aktulellen Position) auch ein $. Wir stehen auf einer Box ( ).
gefragt: Ist irgendwo so ein $?

Lösung: Da unsere TM nichtdeterministisch sein kann, gehen wir vom Startzustand aus spontan entweder nach rechts oder nach links.
Da NTM's immer „hellsehen können", weiss die TM vorher, in welchen sie gehen muss.

δ(q0,) = {(q0, , R), (q0, , L)}
δ(q0, $) = (qf, $, N) // fertig!

Übung 11

Aufgabe 1: deterministische Turingmaschine

schreibe die TM aus Aufgabe 9.3 deterministisch.

Lösung: Wir gehen abwechselnd nach rechts und links, immer 1 Zeichen weiter als vorher: das erste Zeichen, das nicht „X“ ist, ersetzen wir durch ein X und kehren um.

Aufgabe 2:

… ?

Viel Glück bei der Klausur! ☺