Karlsruher Institut für Technologie

aifb
Institut für Angewandte Informatik
und Formale Beschreibungsverfahren

13.02.2018

Lösung zur Klausur über den Stoff der Vorlesung „Grundlagen der Informatik II“ (90 Minuten)

Name: _____ Vorname: _____

Matr.-Nr.: _____ Semester: _____ (WS 2017/18)

Ich bestätige, dass ich die folgenden Angaben gelesen und mich von der Vollständigkeit dieses Klausurexemplars überzeugt habe (Seiten 1-16).

_____

Unterschrift des o. g. Klausurteilnehmers bzw. der o. g. Klausurteilnehmerin

Anmerkungen:

1. Legen Sie bitte Ihren Studierendenausweis bereit.
2. Bitte tragen Sie **Name**, **Vorname** und **Matr.-Nr.** deutlich lesbar ein.
3. Die folgenden **11 Aufgaben** sind vollständig zu bearbeiten.
4. Folgende Hilfsmittel sind zugelassen: **keine.**
5. Täuschungsversuche führen zum Ausschluss von der Klausur.
6. Unleserliche oder mit Bleistift geschriebene Lösungen können von der Klausur bzw. Wertung ausgeschlossen werden.
7. Die Bearbeitungszeit beträgt 90 Minuten.

Nur für den Prüfer :

1	2	3	4	5	6	7	8	9	10	11	-	-	-	-	-	gesamt
(7)	(9)	(12)	(12)	(8)	(10)	(7)	(6)	(6)	(7)	(6)						(90)

Aufgabenübersicht

Aufgabe 1	**7 Punkte**
2018-H-01	**Multiple-Choice**

/ 7

Kreuzen Sie für die folgenden Aussagen an, ob diese jeweils **wahr**, **falsch** oder nach heutigem Kenntnisstand nicht beantwortbar (**unbekannt**) sind. Für falsche Antworten werden **keine** Punkte abgezogen.

Lösung:

	Wahr	Falsch	Unbekannt
Deterministische Kellerautomaten akzeptieren eine kleinere Sprachklasse als nichtdeterministische Kellerautomaten.	☒	☐	☐

Erklärung: $L = \{w \in \{0,1\}^\star \mid w = vv'\}$ ist eine Sprache, die von nichtdeterministischen, aber nicht von deterministischen Kellerautomaten erkannt wird.

NP-vollständige Probleme sind die schwierigsten Probleme, die es gibt.	☐	☒	☐

Erklärung: Es gibt Probleme, die nicht einmal berechenbar sind und sogar Probleme, die nicht einmal semientscheidbar sind. Das sind in gewisser Weise die schwierigsten Probleme, die es gibt. Zwischen NP und den nicht berechenbaren Problemen gibt es aber noch unendlich viele weitere Klassen, die schwierigere Probleme enthalten als NP. Wahrscheinlich enthalten schon so „einfache“ Klassen wie $PSPACE$ echt schwierigere Probleme als alle Probleme in NP.

Es gilt $NP \subseteq P$.	☐	☐	☒

Erklärung: Das berühmte P/NP-Problem ist noch offen; es könnte sein, dass $P = NP$ gilt

Mit irrationalen Zahlen wie $\sqrt{2}, e, \pi, i$ usw. kann man im Rechner nicht exakt rechnen.	☐	☒	☐

Erklärung: Man kann im Rechner symbolisch rechnen, dabei werden die Zahlen aber nicht explizit dargestellt, sondern allgemeine Rechenregeln angewendet: $\sqrt{x} \cdot \sqrt{x} = x, e^{i\pi} = -1$ usw. Dennoch sind symbolische Rechnungen exakt (im Gegensatz zu vielen expliziten Darstellungen wie FPZ oder GPZ), und im Prinzip können beliebige abzählbare Teilmengen von $\mathbb{R}$ abgebildet werden. Grundsätzlich können alle Rechenoperationen, die mit Stift und Papier möglich sind, auch im Rechner durchgeführt werden.

Es gibt für jede beliebige Teilmenge $R \subseteq \mathbb{R}$ der reellen Zahlen eine Zahlendarstellung, die alle Zahlen aus R explizit und exakt im Rechner repräsentiert.	☐	☒	☐

Erklärung: Eine exakte und explizite Repräsentation, d. h. mit Auflistung aller Nachkommastellen, ist schon für rationale Zahlen nicht möglich. Aber auch symbolisch können überabzählbare Teilmengen von $\mathbb{R}$ nicht vollständig repräsentiert werden.

Assemblersprachen haben eine niedrigere Berechnungsmächtigkeit (in Bezug auf die Berechenbarkeit von Funktionen) als objektorientierte Sprachen.	☐	☒	☐

Erklärung: Alle typischen Programmiersprachen sind Turing-vollständig.

Turing-vollständige Programmiersprachen besitzen die höchste Berechnungsmächtigkeit, die es gibt.	☐	☐	☒

Erklärung: Die Churchsche These legt diesen Schluss nahe, aber beweisen kann man sie nicht. Es könnte in 100 Jahren jemand eine Sprache erfinden, die mehr berechnen kann als die heutigen Sprachen. (Es gibt Konzepte wie analoge Computer, die in gewisser Weise Dinge können, die normale Computer nicht können, aber dafür haben sie andere Restriktionen. Insbesondere „berechnen“ sie in unserem Sinne nichts.)

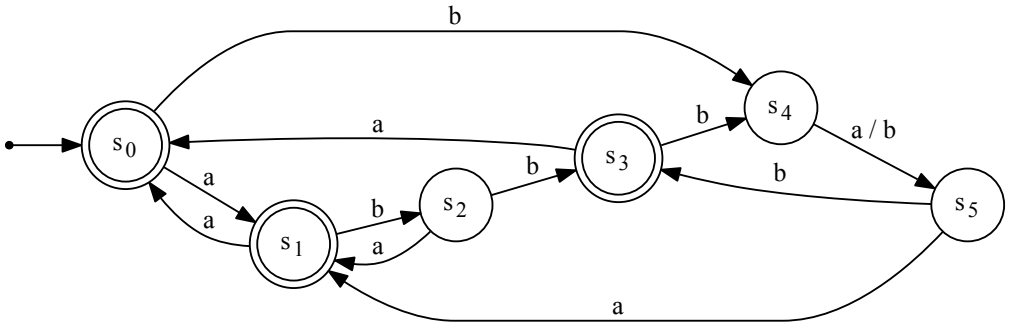
Aufgabe 2 **9 Punkte**

2018-H-02 **Minimierung endlicher Automaten**

/ 9

Gegeben sei folgender deterministischer endlicher Automat $A = (E, S, \delta, s_0, F)$. Durch das abgebildete Zustandsüberführungsdiagramm sei δ definiert.

δ :



(a) Minimieren Sie A und geben Sie den minimierten Automaten A' vollständig an. Geben Sie insbesondere die Minimierungstabelle und ein Zustandsüberführungsdiagramm δ' an.

/ 5

Lösung:

- Übergangstabelle für A:

	a	b
s_0	s_1	s_4
s_1	s_0	s_2
s_2	s_1	s_3
s_3	s_0	s_4
s_4	s_5	s_5
s_5	s_1	s_3

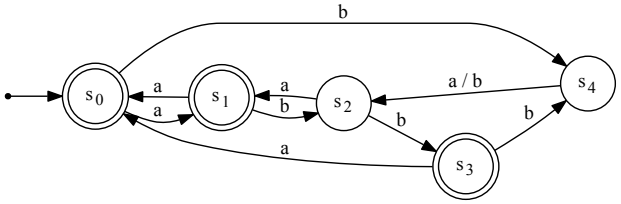
- Dreieckstabelle:

s_1	$\times_2$				
s_2	$\times_0$	$\times_0$			
s_3	$\times_3$	$\times_2$	$\times_0$		
s_4	$\times_0$	$\times_0$	$\times_1$	$\times_0$	
s_5	$\times_0$	$\times_0$	$-$	$\times_0$	$\times_1$
	s_0	s_1	s_2	s_3	s_4

- Übergangstabelle für A':

	a	b
s_0	s_1	s_4
s_1	s_0	s_2
s_2	s_1	s_3
s_3	s_0	s_4
s_4	s_2	s_2

- Der minimierte Automat A' lautet wie folgt:
  $A' = (\{a, b\}, \{s_0, s_1, s_2, s_3, s_4\}, \delta', s_0, \{s_0, s_1, s_3\})$



(b) Geben Sie die Mengen aller zueinander k-äquivalenten Zustände für $k \in \{0, 1, 2\}$ und die Mengen äquivalenter Zustände des endlichen Automaten A an. Verwenden Sie hierfür die nachfolgende Tabelle. Geben Sie auch einelementige Mengen an.

/ 4

Lösung:

0−Äquivalenz	$\{s_0, s_1, s_3\}, \{s_2, s_4, s_5\}$
1−Äquivalenz	$\{s_0, s_1, s_3\}, \{s_2, s_5\}, \{s_4\}$
2−Äquivalenz	$\{s_0, s_3\}, \{s_1, s_5\}, \{s_2\}, \{s_4\}$
Äquivalenz	$\{s_0\}, \{s_1\}, \{s_2, s_5\}, \{s_3\}, \{s_4\}$

Skript für den Ausgangsautomaten:

```
fsm:
(s0, a) | (s2, a) | (s5, a) => s1;
(s0, b) | (s3, b) => s4;
(s1, a) | (s3, a) => s0;
(s1, b) => s2;
(s2, b) | (s5, b) => s3;
(s4, a) | (s4, b) => s5;
--declarations--
s0=s0;
F=s3,s0,s1;
--declarations-end--
```

SKRIPT ID-30035

Skript für den minimierten Automaten:

```
fsm:
(s0, a) | (s2, a) => s1;
(s0, b) | (s3, b) => s4;
(s1, a) | (s3, a) => s0;
(s1, b) | (s4, a) | (s4, b) => s2;
(s2, b) => s3;
--declarations--
s0=s0;
F=s3,s0,s1;
--declarations-end--
```

SKRIPT ID-30037

Aufgabe 3 **12 Punkte**

2018-H-03 **Monotone Grammatiken / Pumping-Lemma**

/ 12

Gegeben sei die Sprache

$$L = \{w \in \{a,b,c,d,e\}^\star \mid w = a^m b^n c^{m+n} d^m e^n \text{ mit } m,n \in \mathbb{N}_+\}$$

Bspw. gilt:

$$abccde, aabcccdde, \underbrace{aaaaa}_{m=5}\underbrace{bbbbbb}_{n=6}\underbrace{ccccccccccc}_{m+n=11}\underbrace{ddddd}_{m}\underbrace{eeeeee}_{n} \in L$$

$$\lambda, a, b, abc, cde, aaccdd, bbccee \notin L$$

(a) Geben Sie eine **monotone oder kontextsensitive** Grammatik G an mit $L(G) = L$. Geben Sie G vollständig an.

/ 6

Lösung:

„Brute-force“:

$$\begin{aligned}
G &= (\{B,C,D,E,S\}, \{a,b,c,d,e\}, P, S)\\
P &= \{S \to BCES \mid aCDS \mid aCDBCE,\\
&\quad Ba \to aB,\\
&\quad CB \to BC,\\
&\quad Ca \to aC,\\
&\quad DB \to BD,\\
&\quad DC \to CD,\\
&\quad Da \to aD,\\
&\quad EB \to BE,\\
&\quad EC \to CE,\\
&\quad ED \to DE,\\
&\quad Ea \to aE,\\
&\quad aB \to ab,\\
&\quad bB \to bb,\\
&\quad bC \to bc,\\
&\quad cC \to cc,\\
&\quad cD \to cd,\\
&\quad dD \to dd,\\
&\quad dE \to de,\\
&\quad eE \to ee\}
\end{aligned}$$

SKRIPT ID-28679

Etwas schlauere Lösung:

$$
\begin{aligned}
G &= (\{B, C, D, S, T\}, \{a, b, c, d, e\}, P, S) \\
P &= \{S \to aCSD \mid aCDT, \\
&\quad T \to BCe \mid BCTe, \\
&\quad CB \to BC, \\
&\quad Ca \to aC, \\
&\quad DB \to BD, \\
&\quad DC \to CD, \\
&\quad aB \to ab, \\
&\quad bB \to bb, \\
&\quad bC \to bc, \\
&\quad cC \to cc, \\
&\quad cD \to cd, \\
&\quad dD \to dd, \\
&\quad eD \to De\}
\end{aligned}
$$

SKRIPT ID-31185

(b) Zeigen Sie mithilfe des Pumping-Lemmas für EA-Sprachen, dass L **nicht rechtslinear** ist.

/ 4

Lösung: Pumpwort $a^n bc^{n+1} d^n e \in L$, Teilwort xy kann nur a's enthalten, gepumpt mit $i \neq 1$ ergibt sich $a^k bc^{n+1} d^n e \notin L$, da $k \neq n$.

(c) Kreuzen Sie an, welche Aussagen **nach den Ergebnissen aus (a) und (b)** wahr bzw. falsch bzw. nicht beantwortbar (unbekannt) sind. Für falsche Antworten werden **keine** Punkte abgezogen.

/ 2

Hinweise:

- Nehmen Sie die Aussagen aus der Fragestellung zu (a) und (b) als wahr an, auch wenn Sie die Teilaufgabe(n) nicht gelöst haben.
- Nehmen Sie an, dass außer diesen beiden Aussagen keine Kenntnisse über L verfügbar sind.

Lösung:

	Wahr	Falsch	Unbekannt
L ist vom Typ 0	☒	☐	☐

Erklärung: Jede Sprache, für die eine Grammatik G existiert, ist vom Typ 0.

L ist vom Typ 1	☒	☐	☐

Erklärung: Die Grammatik G aus (a) ist monoton, also ist L vom Typ 1.

L ist vom Typ 2	☐	☐	☒

Erklärung: Nach den Ergebnissen aus (a) und (b) wissen wir nur, dass L mindestens vom Typ 1 und ganz sicher nicht vom Typ 3 ist.

L ist vom Typ 3	☐	☒	☐

Erklärung: Vom Typ 3 kann L nicht sein, da nach (b) das Pumping-Lemma nicht erfüllt ist.

Aufgabe 4 **12 Punkte**

2018-H-04 **Turingmaschinen**

/ 12

Gegeben sei wieder die Sprache L aus Aufgabe 3:

$$L = \{w \in \{a,b,c,d,e\}^\star \mid w = a^m b^n c^{m+n} d^m e^n \text{ mit } m,n \in \mathbb{N}_+\}$$

Gegeben sei außerdem die Turingmaschine

$$T = (\{a,b\}, \{a,\ldots,e,A,B,\star\}, \{s_0,\ldots,s_4,s_X,s_{z1},s_{z2}\}, \delta, s_0)$$

die auf Eingabe $w = a^m b^n$ hin die „zugehörige“ Ausgabe $f_T(w) = a^m b^n c^{m+n} d^m e^n \in L$ erzeugt. T berechnet also folgende Funktion:

$$f_T : \{a\}^\star \times \{b\}^\star \to \{a,b,c,d,e\}^\star : f_T(a^m b^n) = a^m b^n c^{m+n} d^m e^n \text{ für } m,n \in \mathbb{N}_+$$

δ	a	b	c	d	e	A	B	$\star$
s_0	(s_X, a, N)							
s_X	(s_1, A, R)	(s_1, B, R)	(s_{z2}, c, L)					
s_1	(s_1, a, R)	(s_1, b, R)	(s_1, c, R)				(s_1, B, R)	(s_{z1}, c, L)
s_2	(s_2, a, R)		(s_2, c, R)	(s_2, d, R)			(s_2, B, R)	(s_{z2}, d, L)
s_3	(s_3, a, R)	(s_3, b, R)					(s_4, b, R)	
s_4			(s_4, c, R)	(s_4, d, R)	(s_4, e, R)		(s_4, B, R)	(s_{z2}, e, L)
s_{z1}	(s_{z1}, a, L)	(s_{z1}, b, L)	(s_{z1}, c, L)			(s_X, A, R)	(s_X, B, R)	
s_{z2}	(s_{z2}, a, L)	(s_3, b, R)	(s_{z2}, c, L)	(s_{z2}, d, L)	(s_{z2}, e, L)	(s_2, a, R)	(s_{z2}, B, L)	$(s_3, \star, R)$

SKRIPT ID-C29916

(a) Geben Sie die Konfigurationsfolgen von T bei folgenden Eingaben an:

- $w = a$:

/ 5

Lösung:

Tape	Transition	*Konfiguration*
$\hat{a}$	$(s_0, a) \to (s_X, a, N)$	(λ, s_0, a)
$\hat{a}$	$(s_X, a) \to (s_1, A, R)$	$\vdash (\lambda, s_X, a)$
$A\hat{\star}$	$(s_1, \star) \to (s_{z1}, c, L)$	$\vdash (A, s_1, \star)$
$\hat{A}c$	$(s_{z1}, A) \to (s_X, A, R)$	$\vdash (\lambda, s_{z1}, Ac)$
$A\hat{c}$	$(s_X, c) \to (s_{z2}, c, L)$	$\vdash (A, s_X, c)$
$\hat{A}c$	$(s_{z2}, A) \to (s_2, a, R)$	$\vdash (\lambda, s_{z2}, Ac)$
$a\hat{c}$	$(s_2, c) \to (s_2, c, R)$	$\vdash (a, s_2, c)$
$ac\hat{\star}$	$(s_2, \star) \to (s_{z2}, d, L)$	$\vdash (ac, s_2, \star)$
$a\hat{c}d$	$(s_{z2}, c) \to (s_{z2}, c, L)$	$\vdash (a, s_{z2}, cd)$
$\hat{a}cd$	$(s_{z2}, a) \to (s_{z2}, a, L)$	$\vdash (\lambda, s_{z2}, acd)$
$\hat{\star}acd$	$(s_{z2}, \star) \to (s_3, \star, R)$	$\vdash (\lambda, s_{z2}, \star acd)$
$\star\hat{a}cd$	$(s_3, a) \to (s_3, a, R)$	$\vdash (\lambda, s_3, acd)$
$\star a\hat{c}d$		

Eine Angabe der Konfigurationsfolge (rechte Spalte) ist laut Aufgabenstellung gefordert, diese Tabelle liefert noch ein paar mehr Informationen.

- $w = b$:

  Lösung: Bei Eingabe b bleibt T sofort stehen, da es von s_0 aus keine Folgekonfiguration gibt. Die Konfigurationsfolge ist: (λ, s_0, b).

(b) Warum ist es – streng nach Definition von Turingmaschinen – in Ordnung, dass sich T für diese (in Bezug zu f_T so ähnlichen) Eingaben a bzw. b so unterschiedlich verhält?

Hinweis: Überlegen Sie, inwiefern diese beiden Eingaben „untypisch“ für die angegebene Funktion f_T sind.

/ 2

Lösung: Weil beide Eingaben für f_T undefiniert sind und damit auch die Verhaltensweise von T undefiniert ist, also beliebig sein darf.

(c) Erklären Sie für jeden der Zustände in S **kurz**, welche Wirkung er im Verlauf der Rechnung hat. Beschreiben Sie außerdem **kurz** das Gesamtverhalten von T.

/ 5

Lösung:

Hinweis: Die folgenden Erklärungen sind sehr ausführlich. Für die volle Punktezahl muss nicht jeder Sonderfall exakt durchexerziert worden sein. Es reicht, wenn klar wird, dass die grundsätzliche Wirkung jedes Zustands und der Turingmaschine insgesamt verstanden worden ist.

- s_0: **Lösung:** Wechselt nur beim Lesen von a nach s_X, damit keine Eingaben akzeptiert werden, die keine a's enthalten. Nach Begründung aus (b) wäre es allerdings auch in Ordnung, direkt in s_X zu starten. Der Fall, dass es keine b's gibt, wird ja auch nicht abgefangen.

- s_X: **Lösung:** Markiert durch ein großes A (bzw. später durch ein großes B, wenn wir bei den b's angekommen sind), dass das aktuelle Zeichen abgearbeitet ist. Wechselt danach nach s_1, um für dieses Zeichen (egal ob es sich um ein a oder ein b handelt) hinten ein c zu schreiben. Wenn der Kopf allerdings auf c steht, ist das ein Signal, dass der erste Teil der Rechnung, der c's schreibt abgeschlossen ist. Der zweite Teil beginnt in Zustand s_2, vorher muss aber zuerst mit s_{z2} zurück nach links gelaufen werden.
- s_1: **Lösung:** Läuft ganz nach rechts (d, e, A können (noch) nicht vorkommen, daher die leeren Felder), um hinten ans Wort ein c zu schreiben. Wechselt daraufhin nach s_{z1}, um in einer Schleife für das nächste a bzw. b ebenfalls ein c zu schreiben. Sowohl s_X als auch s_1 und s_{z1} können dabei für a und b verwendet werden, da immer dasselbe getan, nämlich hinten ein c geschrieben werden muss.
- s_2: **Lösung:** Schreibt, analog zu s_1 hinten ein d an das Wort. Da b, e, A nicht vorkommen können, bleiben diese Felder leer. Da s_2 nur von s_{z2} aus besucht werden kann, muss sich nicht um das Ersetzen von A durch a gekümmert werden – anders als in s_3, s. u. Nachdem d geschrieben wurde, wird durch s_{z2} zurückgelaufen, um die Schleife fortzusetzen.
- s_3: **Lösung:** Schreibt zusammen mit s_4 analog zu s_1 und s_2 für jedes B ein e hinten ans Wort. Da in diesem Fall, anders als bei s_2, s. o. und s. u., das B nicht schon in s_{z2} durch b ersetzt werden kann, ist diese Aufteilung in zwei Zustände nötig. s_3 läuft lediglich bis zum ersten B (das als nächstes bearbeitet werden soll) ersetzt dieses durch b und wechselt dann nach s_4.
- s_4: **Lösung:** Läuft vollends nach rechts (a, b, A können nicht vorkommen) und schreibt ein e. Wechselt nach s_{z2}, um die Schleife für die verbleibenden B's fortzusetzen.
- s_{z1}: **Lösung:** Läuft nach links, bis ein A oder ein B gelesen wird, wechselt dann nach s_X, wo entschieden wird, ob weiter c's geschrieben werden oder der zweite Teil begonnen werden soll.
- s_{z2}: **Lösung:** Läuft nach links, bis entweder ein b oder ein A gelesen werden. Im ersten Fall müssen die verbliebenen b's abgearbeitet werden (Folgezustand s_3), im zweiten die a's (Folgezustand s_2; in diesem Fall gibt es im Wort keine b's, sondern nur B's, daher wurde über diese hinweggegangen; außerdem wird in diesem Fall gleich A durch a ersetzt, was den Prozess in s_2 vereinfacht).

  Der Grund für das Akzeptieren von Wörtern, die keine b's enthalten, liegt übrigens darin, dass beim Lesen von c in diesem Zustand nicht unterschieden werden kann, ob wir regulär von rechts nach links über die c's laufen oder ob wir direkt von s_3 aus kommen, in welchem Fall das eigentlich ungültig ist. (Natürlich könnte man eine Sonderbehandlung dieser Fälle durch weitere Zustände einbauen, aber das wollten wir Ihnen aus Komplexitätsgründen ersparen.)
- Gesamtverhalten von T: **Lösung:** Zuerst wird nacheinander jedes a durch A ersetzt und dabei ein c hinten ans Wort gesetzt. Danach wird durch dieselben Zustände

analog mit den b's verfahren. Im zweiten Teil wird zunächst jedes A zurück nach a umgewandelt und dabei ein d hinten ans Wort geschrieben. Danach wird analog mit den B's verfahren.

Aufgabe 5	**8 Punkte**
2018-H-05	**Berechenbarkeit**

/ 8

Sei U^X ein **universeller Automat** mit folgenden Eigenschaften:

- Für $X \in \{TM, LBA, KA, EA\}$ ist U^X ein Automat vom Typ X.
- Als Eingabe erhält U^X:
  - einen **Automaten** A vom Typ X und
  - ein Wort w.
- U^X akzeptiert seine Eingabe genau dann, wenn A die Eingabe w akzeptiert.

Zum Beispiel wäre U^{TM} eine universelle Turingmaschine.

(a) Geben Sie die Sprache $L(U^X)$ formal an.

/ 2

Lösung:

$L\left(U^X\right) = \{\langle A, w\rangle \mid A \text{ akzeptiert } w\}$: Die Menge aller Kodierungen $\langle A, w\rangle$ von Automaten A und zugehöriger Eingabe w, so dass A bei der Eingabe von w in einem akzeptierenden Endzustand hält.

(b) Argumentieren Sie, warum U^{EA} nicht existieren kann.

Hinweise:

- Überlegen Sie, welcher Sprachklasse $L(U^{EA})$ angehören müsste und welche aus der Vorlesung bekannte Eigenschaft dieser Sprachklasse dann nicht erfüllt sein kann.
- Versuchen Sie, eine nicht notwendigerweise formale, aber schlüssige Begründung zu finden. Formulieren Sie umgangssprachlich präzise.

/ 4

Lösung:

Die Sprache $L(U^{EA})$ müsste das Pumping Lemma für rechtslineare Sprachen erfüllen, da U^{EA} ein endlicher Automat ist. Unabhängig davon, wie die Kodierung gewählt würde, müsste eine Eingabe $\langle A, w\rangle$ in $xyz = \langle A, w\rangle$ zerlegt werden können, sodass die Pump-Eigenschaften erfüllt sind. Ab einer bestimmten Größe n der Kodierung müsste also ein Teil des vorderen Teils y (mit $|xy| \leq n$) auf sehr vielfältige ($\forall i \neq 1$) und regelmäßig Weise aufgebläht werden können, während der hintere Teil z unverändert bleibt. Es gibt viele Möglichkeiten der Kodierung, aber es ist offensichtlich (intuitiv) unmöglich, die Kodierung so zu gestalten, dass so eine Aufblähung nicht zu einem ungültigen Codewort führen kann.

Beispielsweise könnte eine einfache Kodierung zuerst das Wort w, dann den Automaten A enthalten (oder umgekehrt): $\langle A, w\rangle = \langle A\rangle\#w$ (oder $\langle A, w\rangle = w\#\langle A\rangle$). In diesem Fall würde ab einer bestimmten Größe von w nur die Eingabe gepumpt werden, nicht aber der Automat, was dazu führen kann, dass sich das Akzeptanzverhalten von A ändern müsste – weil das gepumpte w nicht mehr akzeptiert wird oder jetzt erst akzeptiert wird, dies aber durch die Simulation nicht gewährleistet werden kann, weil A ja nicht verändert wurde (oder entsprechend umgekehrt).

Jegliche Zerlegung von $\langle A, w\rangle$ und anschließendem Aufpumpen führt dazu, dass entweder die Automatendefinition oder das Eingabewort w verändert werden oder beides. Hierdurch werden notgedrungen auch unzulässige Kodierungen w' erzeugt[1]. Folglich existiert kein U^{EA}.

(c) Philosophieren Sie **sehr kurz**, ob ein U^{KA} bzw. ein U^{LBA} existieren kann oder nicht.

Hinweise:

- Beachten Sie die Hinweise unter (b). Eventuell reicht eine kurze Reflexion, warum Ihre Argumentation dort auch hier gilt – oder eben nicht gilt.

Lösung:

Analog zur Argumentation aus (b) könnte man folgern, dass das PPL auch für KA nicht erfüllt werden kann. Für LBA ist dagegen kein PPL bekannt (zumindest aus der Vorlesung; in der Realität gibt es ähnliche Konstrukte auch für LBA), also könnte es U^{LBA} nach diesen Kenntnissen theoretisch geben.

Auch andere schlüssige Argumentationsketten sind in Ordnung; im Prinzip kann man je nach Voraussetzung sowohl beide Automaten für denkbar als auch beide für unmöglich erklären. Es geht nur um die richtige Begründung.

[1] Diese Argumentation ist intuitiv richtig, aber formal müsste man noch genauer argumentieren. Es könnte immer noch Kodierungen geben, die auf so clevere Weise Eingabe und Automat miteinander verzahnen, dass das Pumpen möglich ist. Das scheint undenkbar – und es stimmt auch nicht, aber nach der bisherigen Argumentation haben wir es nicht ganz ausgeschlossen. Eine wasserdichtere Argumentation findet sich hier: https://www.quora.com/Can-a-finite-state-machine-be-universal.

/ 2

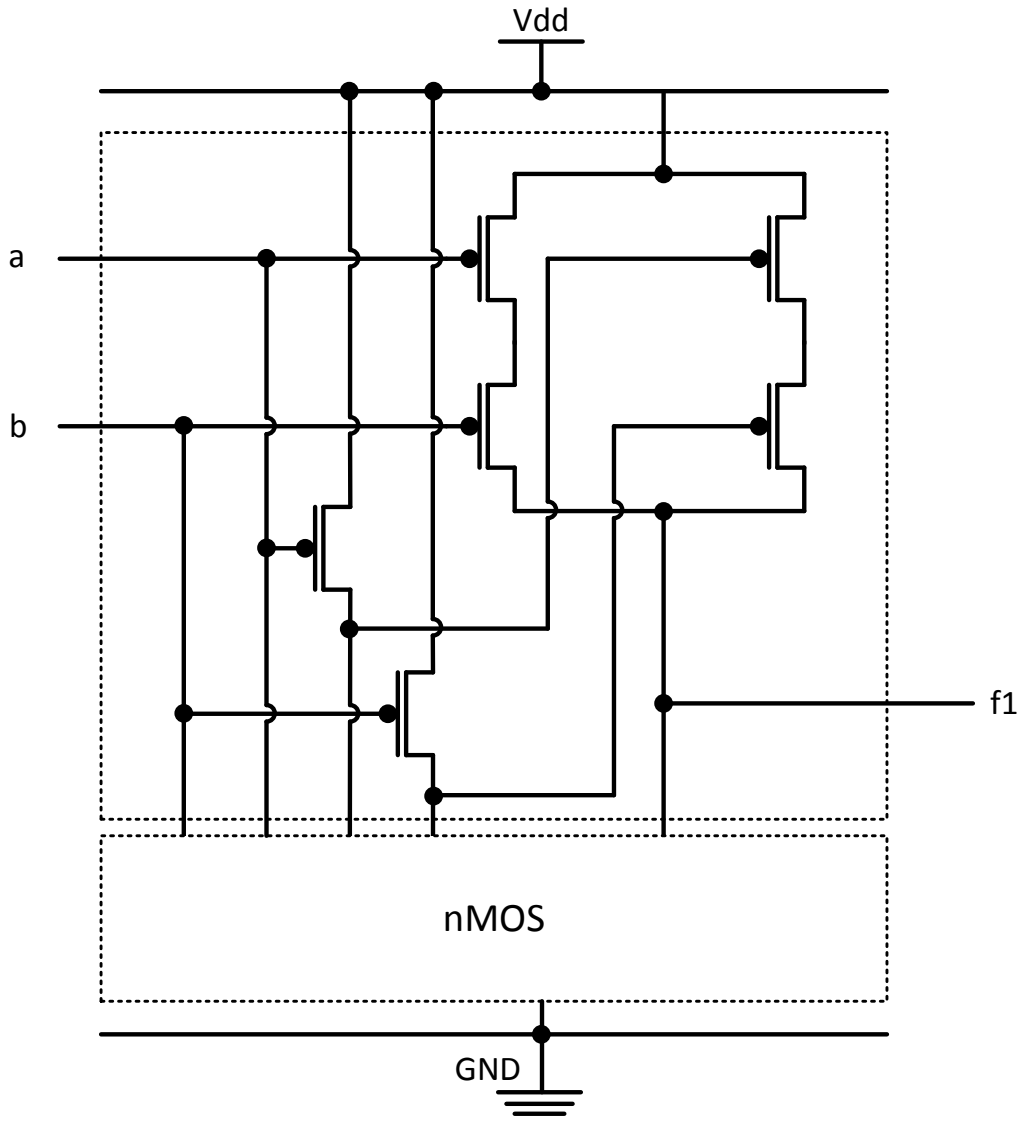
Aufgabe 6	**10 Punkte**
2018-H-06	**CMOS / Schaltnetze**

/ 10

(a) Zeichnen Sie den **pMOS-Teil** eines Äquivalenzgatters: Das Ausgangssignal f_1 muss also genau dann auf Eins gesetzt werden, wenn beide Eingangssignale a und b den gleichen Wert haben.

/ 6

Lösung:

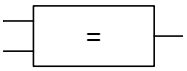


(b) Benutzen Sie das in Teilaufgabe (a) beschriebene Äquivalenzgatter, um ein **Schaltnetz** zu entwerfen, das die vier Eingangssignale w, x, y und z auf Parität prüft. Das Ausgangssignal f_2 muss genau dann auf Eins gesetzt werden, wenn eine gerade Anzahl an Einsen anliegt. Auch keine Eins gilt als gerade Anzahl. Sie dürfen das Äquivalenzgatter als Blackbox mit zwei Eingängen und einem Ausgang verwenden.

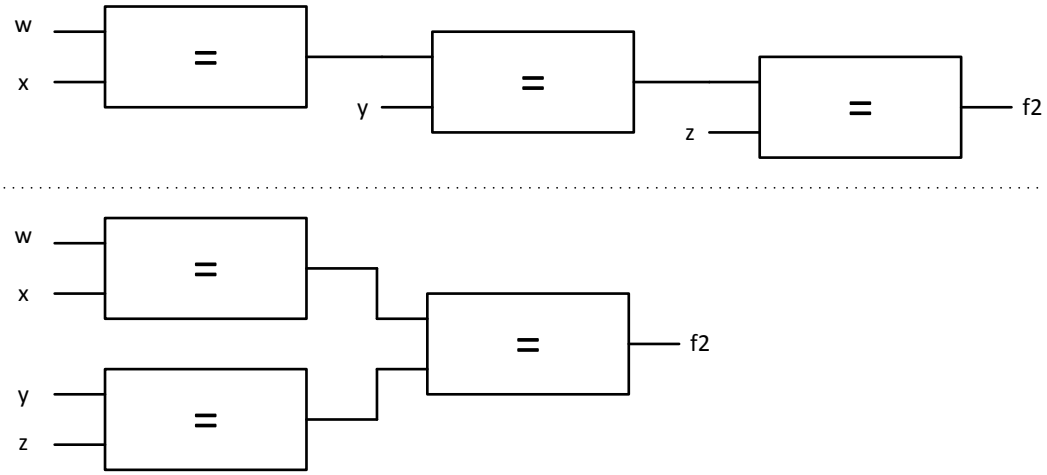
Hinweis: Sie dürfen auch andere aus der Vorlesung bekannte Gatter verwenden. Dies ist aber nicht notwendig.

/ 4

Symbol für Äquivalenzgatter:



Lösung:



Aufgabe 7	**7 Punkte**
2018-H-07	**Huffman**

/ 7

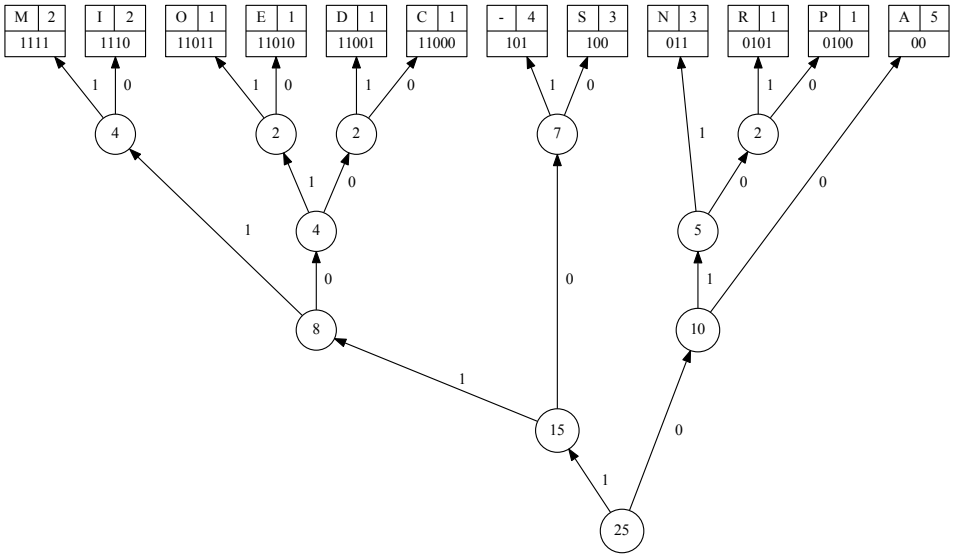
Gegeben sei der folgende Text aus 25 Zeichen:

MENS_SANA_IN_CAMPARI_SODA

Erzeugen Sie zu der durch den Text gegebenen Wahrscheinlichkeitsverteilung eine Huffman-Kodierung. Tragen Sie dazu die Häufigkeiten der Zeichen in die untere Zeile der ersten Tabelle ein, erstellen Sie einen Huffman-Baum mit Angabe der Häufigkeiten an den Knoten und geben Sie in der zweiten Tabelle für jedes Zeichen eine dem Baum entsprechende Kodierung an.

M	I	O	E	D	C	_	S	N	R	P	A
2	2	1	1	1	1	4	3	3	1	1	5

Lösung (Beispiel):



SKRIPT ID-28731

Aufgabe 8 **6 Punkte**

2018-H-08 **Kodierung – Fano-Bedingung**

/ 6

(a) Seien A und B Alphabete, $c : A \to B^3$ eine Blockkodierung und $c^\star$ die natürliche Fortsetzung von c. Zeigen Sie, dass

$$c \text{ injektiv} \iff c^\star \text{ injektiv}.$$

/ 2

Hinweis: Die natürliche Fortsetzung meint die Erweiterung eines Codes von einzelnen Zeichen zu ganzen Codewörtern. Injektivität von $c^\star$ bedeutet, dass jedes Codewort eindeutig dekodiert werden kann.

Lösung:

„⇒“ Aus der Injektivität von c folgt, dass keine Elemente $a_1, a_2 \in A$ existieren für die gilt: $c(a_1) = c(a_2)$. Da c zusätzlich eine Blockkodierung ist und damit nur Codewörter der gleichen Länge erzeugt, kann kein Codewort Präfix eines anderen Codewortes sein. (Um die Fano-Bedingung zu verletzen müssten dann zwei Elemente aus A auf das gleiche Codewort abgebildet werden, was im Widerspruch zur Injektivität von c steht.) Daraus folgt, dass die Fano-Bedingung für c erfüllt ist. Aus der Vorlesung ist bekannt, dass c injektiv und c erfüllt die Fano-Bedingung $\Rightarrow c^\star$ injektiv.

„⇐“ Aus der Vorlesung ist bekannt, dass $c^\star$ injektiv $\Rightarrow c$ injektiv.

(b) Seien $A := \{x, y, z\}$ und $B := \{0, 1\}$ Alphabete. Definieren Sie eine **injektive** Kodierung

$$c : A \to B^\star$$

sodass c die Fano-Bedingung **nicht** erfüllt, aber ihre natürliche Fortsetzung $c^\star$ dennoch **injektiv** ist. Begründen Sie Ihre Antwort kurz.

/ 4

Hinweis: Gesucht ist also eine Kodierung, die die Fano-Bedingung nicht erfüllt, aber trotzdem eindeutig dekodiert werden kann.

Lösung:

$$c(x) = 0$$
$$c(y) = 01$$
$$c(z) = 11$$

Die Kodierung c verletzt die Fano-Bedingung, da $c(x)$ Präfix von $c(y)$ ist. Die natürliche Fortsetzung $c^\star$, ist allerdings injektiv, da für jedes Codewort $w = \{c(x), c(y), c(z)\}^\star$ eindeutig entschieden werden kann, ob eine einzelne 0 ein x kodiert oder Teil der Kodierung von y ist. Dies liegt daran, dass eine 1 niemals einzeln in w auftreten kann, sondern nur in der Sequenz 01.

Aufgabe 9 **6 Punkte**

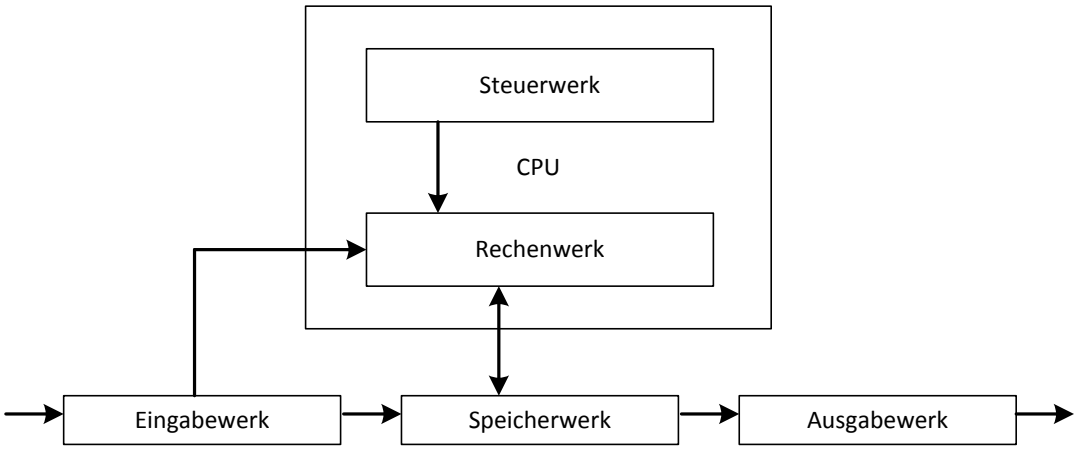
2018-H-09 **Rechnerarchitektur**

/ 6

(a) In der Vorlesung wurde das klassische Konzept eines Universalrechners nach Burks, Goldstine und von Neumann vorgestellt. Kennzeichnen Sie im gegebenen Schaubild die einzelnen Komponenten dieses Konzepts und zeichnen Sie den Datenfluss ein. Beschreiben Sie **kurz** die Funktion jeder Komponente.

/ 4

Lösung:



- Steuerwerk: Steuerung des Programmablaufs
- Rechenwerk: Ausführung arithmetischer und logischer Operationen
- Speicherwerk: Speicherung von Programmen und Daten
- Eingabe-/Ausgabewerk: Eingabe von Programmen/Daten in den Speicher, Ausgabe von Ergebnissen

(b) Das Konzept des Befehlsphasen-Pipelinings führt zu einer erheblichen Beschleunigung der Programmausführung. Es können dabei aber auch Probleme auftreten. Nennen Sie zwei und geben Sie je einen Ansatz zu deren Lösung.

/ 2

Lösung:

- Laden falscher Befehle nach bedingten Sprüngen
  - spekulative Programmausführung
- Zugriff auf veraltete Werte wenn Operanden gelesen werden, bevor der vorherige Befehl sie mit einem aktuelleren Wert überschrieben hat
  - Einfügen von Leer-Befehlen (NOP - No Operation)

Aufgabe 10 **7 Punkte**

2018-H-10 **Programmiersprachen**

/ 7

In der Vorlesung haben wir gesehen, dass alle gängigen Programmiersprachen dieselbe Ausdrucksmächtigkeit haben, also einander simulieren können. Im Folgenden seien gegeben:

- eine **GOTO-Sprache** (das sei eine übliche Assemblersprache, die Sprünge in Abhängigkeit des aktuellen Akkumulator-Werts erlaubt: `JNZ label`) und
- eine **WHILE-Sprache** (das sei eine höhere Sprache wie Java, die allgemeine WHILE-Schleifen unterstützt: `while (b) { * do something * }`).

(a) Gegeben sei ein Programmausschnitt in der GOTO-Sprache:

/ 5

```
label:  LOAD R1
        ADD R2
        STORE R1
        LOAD R2
        SUB 1
        STORE R2
        JNZ label  // „JUMP NOT ZERO“
```

Simulieren Sie das Programm durch die WHILE-Sprache, d. h. geben Sie ein äquivalentes WHILE-Programm an.

Hinweise:

- Nutzen Sie möglichst Java-Notation oder einen verständlichen Pseudocode.
- Der Akkumulator sei über `ACC` erreichbar, die Register über ihre Namen `Ri`.

Lösung:

```
while (R2 != 0) {
    R1 += R2;
    R2 -= 1;
}
```

Alternative: `R1 += R2 * (R2 + 1) / 2;`

(b) Was müsste im gegebenen Programmausschnitt verändert werden, damit am Ende die Fakultät `R2!` in `R1` steht?

Hinweis: Es sind **an zwei Stellen** Veränderungen nötig. Tragen Sie diese einfach im gegebenen Code ein.

/ 2

Lösung:

- Zu Beginn:
  `LOAD #1`
  `STORE R1`
- Statt `ADD R2` ⇒ `MULT R2`.

Aufgabe 11 **6 Punkte**

2018-H-11 **Betriebssysteme**

/ 6

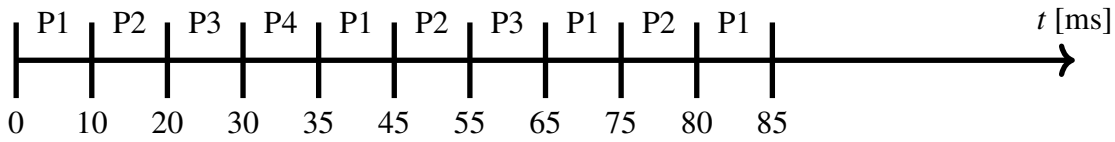
(a) Betrachten Sie die Prozesse P1 bis P4, die in die Warteschlange eines Prozessors zur Bearbeitung eingereiht werden.

Prozesse	CPU-Zeit in ms
P1	35
P2	25
P3	20
P4	5

Teilen Sie den Prozessen Rechenzeit gemäß dem Round-Robin Verfahren zu. Die Zeitscheibe sei dabei in feste Zeitspannen der Länge $Z = 10$ ms unterteilt. Veranschaulichen Sie Ihr Ergebnis auf dem gegebenen Zeitstrahl.

/ 3

Lösung:



(b) Nehmen Sie nun an, Prozess P1 wird **blockiert**, nachdem er für 5 ms **aktiv** war. Nachdem weitere 40 ms vergangen sind, wird der Zustand von P1 wieder auf **bereit** gesetzt. Teilen Sie den Prozessen aus Aufgabenteil (a) unter dieser Voraussetzung Rechenzeit gemäß dem Round-Robin Verfahren zu mit einer Zeitscheibenlänge $Z = 10$ ms. Veranschaulichen Sie Ihr Ergebnis auf dem gegebenen Zeitstrahl.

/ 3

Lösung: