Fragen, Antworten und Kommentare zur aktuellen Vorlesung

Bitte nehmen Sie an der anonymen Lehrevaluation unter https://forms.gle/pb3gFZN7rreEUHDr9 spätestens bis 5.6 teil. Die von mir kommentierten Ergebnisse stehen im letzten Fragen&Antworten-Dokument des Semesters.

Das Video zur Lösung der Aufgabe 39 finden Sie unter: https://youtu.be/lICMrmjcJHw . Natürlich dürfen in Ihrer Lösung zu 39 a) auch weitere Zustände vorkommen, auch wenn sie nicht nutzbar sind. Bei b fehlt bei der leeren Menge eine Schleife mit a, b.

Das Video zur Lösung der Aufgabe 40 finden Sie unter: https://youtu.be/zgL1RYheaD4 . In Aufgabe 40a) fehlt ein Übergang z1 a z0. Bei 40c) sollte $\{a^n b^m \mid n > 0, m = 0 \text{ oder } m = 1\}$ stehen (oder eine vergleichbare Notation, wie z. B. ein regulärer Ausdruck a*(a + ab) (kommt später))

Frage: Müssen wir in der Klausur exakt nach den vorgegebenen Algorithmen vorgehen?

Antwort: Nur wenn dies im Detail, z. B, durch die Vorgabe einer zu füllenden Tabellenstruktur vorgegeben ist (Aufgaben 4 und 8). Wenn Sie Fehler machen, könnte es aber schwieriger sein Teilpunkte zu vergeben, wenn mir der verkürzte Rechenweg nicht klar oder die Abkürzung schlicht falsch ist.

Frage: Was wird sich an den Klausuraufgaben inhaltlich ändern?

Antwort: Gehen Sie davon aus, dass die Aufgabenstellungen identisch sind, nur alle Beispieldaten geändert werden. Da dadurch die Aufgaben etwas einfacher oder schwerer werden, können sich die Punktzahlen leicht ändern. Bei Fragen wie „gilt Eigenschaft XY“ kann es natürlich sein, dass diese Eigenschaft in der Probeklausur gilt, in der echten Klausur nicht oder umgekehrt.

Frage: Eine Frage gibt es noch, mit der eingabe von bspw 39b auf der Testumgebung, weiß ich nicht wie ich das eintragen soll? Es habe zwar das Kapitel druchgelesen gehabt, aber für die Probeklausur2.atm fehlt mir ein beispiel was und wie ich es eintragen soll.

Antwort: Die Benennung der Zustände ist für die Testumgebung uninteressant und wird nicht überprüft, Sie können natürlich Namen wie z1_z2 wie bei der Potenzmengenkonstruktion benutzen. Der Test prüft nicht ganz genau, ob Ihre Berechnung korrekt ist, da nur die Sprachäquivalenz geprüft wird. Wollen Sie das Ergebnis noch genauer prüfen, müssten Sie den Automaten aus 39b in die Testumgebung eingeben und von dieser den deterministischen Automaten berechnen lassen und ausgeben und die Zustandsnamen vergleichen.

Frage: Was passiert genau im Automat A3 in Folie 240 mit dem Wort a?

Antwort: Nach a kann man in z1 sein, so dass als nächste Zeichen b oder c möglich sind. Nach a kann man aber auch in z2 sein, da $a\varepsilon = a$. Es wurde also intern nichtdeterministisch entschieden nach z2 zu gehen, danach kann als Zeichen nur b folgen. Ähnlich zu z2 kann man nach a auch in z3 sein und da kann nur c folgen.

Frage: Ich verstehe nicht genau, was in den Tests zur Automaten-Erstellung passiert.

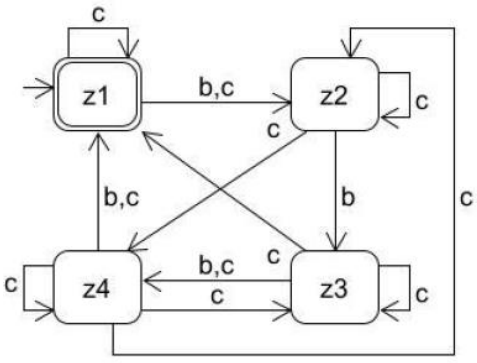
Antwort: Das ist wahrscheinlich, da hier, anders als sonst üblich nicht nur einzelne Beispiele, sondern die gesamte Struktur getestet, genauer verifiziert, wird. Bei einer Verifikation ist die Korrektheit nachgewiesen und nicht nur anhand von Beispielen plausibel gemacht worden. Um, nicht im Test die Lösung direkt einzubauen, wird ein anderer Ansatz zur Sprachbeschreibung benutzt. Dies sind reguläre Ausdrücke, aus denen ein eindeutiger minimaler endlicher Automat berechnet wird, was allerdings erst etwas später in der Vorlesung vorkommt. Statt Lesbarkeit steht in diesem Fall die garantierte Korrektheit im Mittelpunkt.

Die Tests bei den Sprachen der Automaten haben damit den wesentlichen Vorteil, dass Sie die Lösungen vollständig auf Korrektheit überprüfen können. Das ist bei kontextfreien Grammatiken (und Turing-Maschinen) nicht möglich. Es gibt kein Programm was für gegebene beliebige zwei kontextfreie Grammatiken überprüfen kann ob sie die gleiche Sprache beschreiben. Dies ist als eines der unentscheidbaren Probleme bekannt. Das bedeutet auch, dass die gegebenen Tests nicht präzise sind. Wenn Sie eine Grammatik schreiben, die genau die Worte erzeugt, die in den Tests gefordert sind, wird diese Grammatik alle Tests erfüllen obwohl sie keinen tieferen Sinn hat.

Frage: Wie funktioniert das mit dem deterministischen Automaten mit den 2^n Zuständen?

Antwort: Ist recht trickreich. Ein erster Teilansatz ist, dass es zunächst ein Zeichen gibt mit dem einfach von einem Zustand zum nächsten wieder zum Anfang geschaltet werden kann. Im Beispiel ist das das Zeichen b. Damit sind {z1}, {z2}, {z3} und {z4} in der Potenzmengenkonstruktion erreichbar

Dann müssen durch die Potenzmengenkonstruktion alle weiteren Elemente der Potenzmenge erreicht werden können. Dazu nimmt man sich einen existierenden Zustand und ein neues Zeichen und geht mit diesem zu allen Zuständen der Potenzmenge. Im Beispiel



z1 -c-> {z1,z2} da mit b immer weitergeschaltet wird, ist damit auch {z2,z3}, {z3,z4}, {z4,z1} über die Potenzmengenkonstruktion erreichbar, da z. B. {z1,z2} -b-> {z2,z3}.

Es fehlt bei den zweielementigen Elementen der Potenzmenge noch {z1,z3} (und damit auch {z2,z4}). Dies könnte mit einem neuen Zeichen erreicht werden. Um die Zahl der Zeichen zu reduzieren reicht es, hier nicht z1 zu nutzen, also wird

z2 -c->{z2,z4} ergänzt.

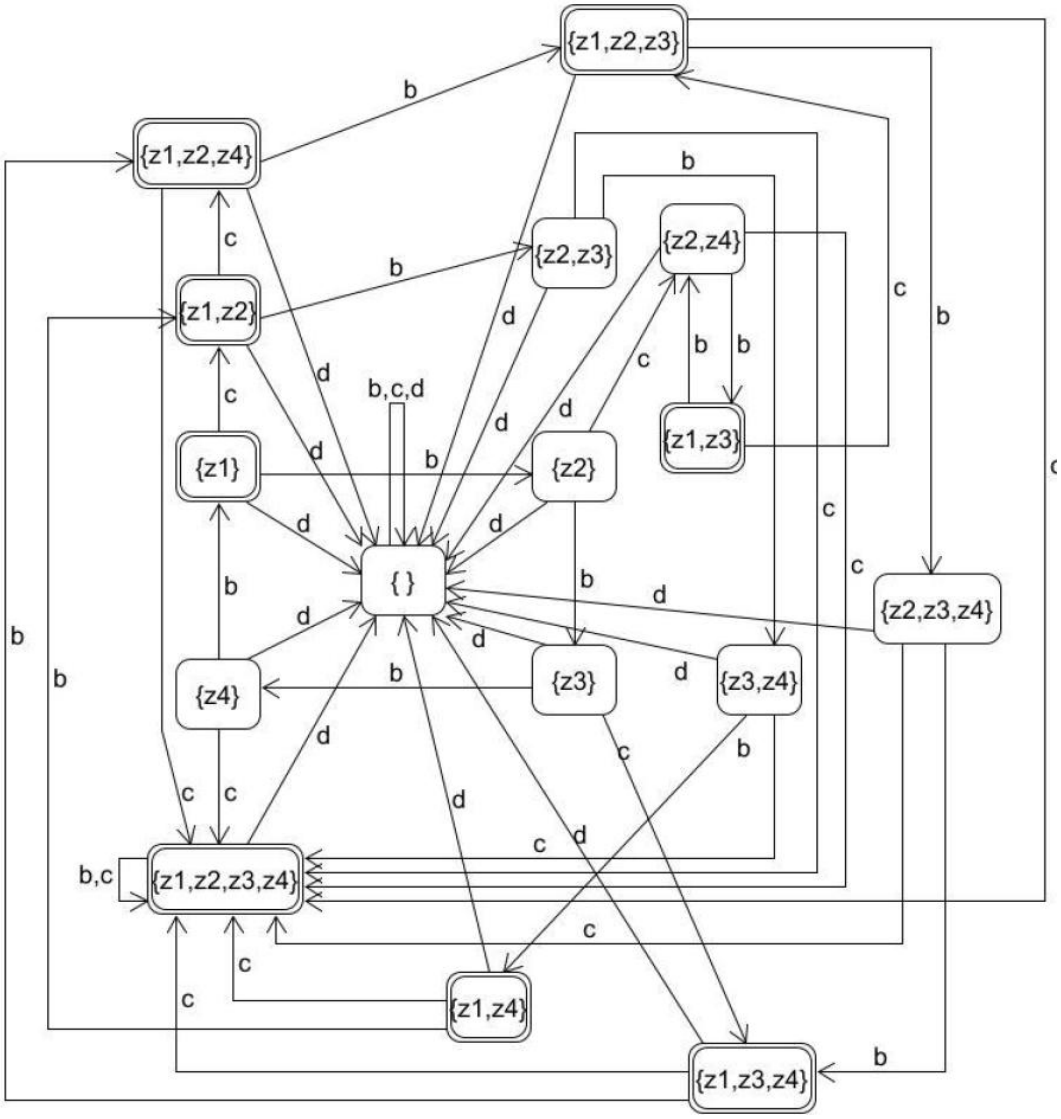
Nun werden alle dreielementigen Elemente der Potenzmenge betrachtet. Es wird noch kein neues Zeichen benötigt, da noch ein anderer Ausgangszustand nutzbar ist, es wird ergänzt:

z3 -c->{z1,z3,z4}, womit auch {z2,z4,z1}, {z3,z1,z2} und {z4,z2,z3} erreichbar sind.

Dann wird für das vierelementige Element eine Ergänzung vorgenommen, ein Zustand wurde noch nicht genutzt, so dass das gleiche Zeichen weiterhin nutzbar ist.

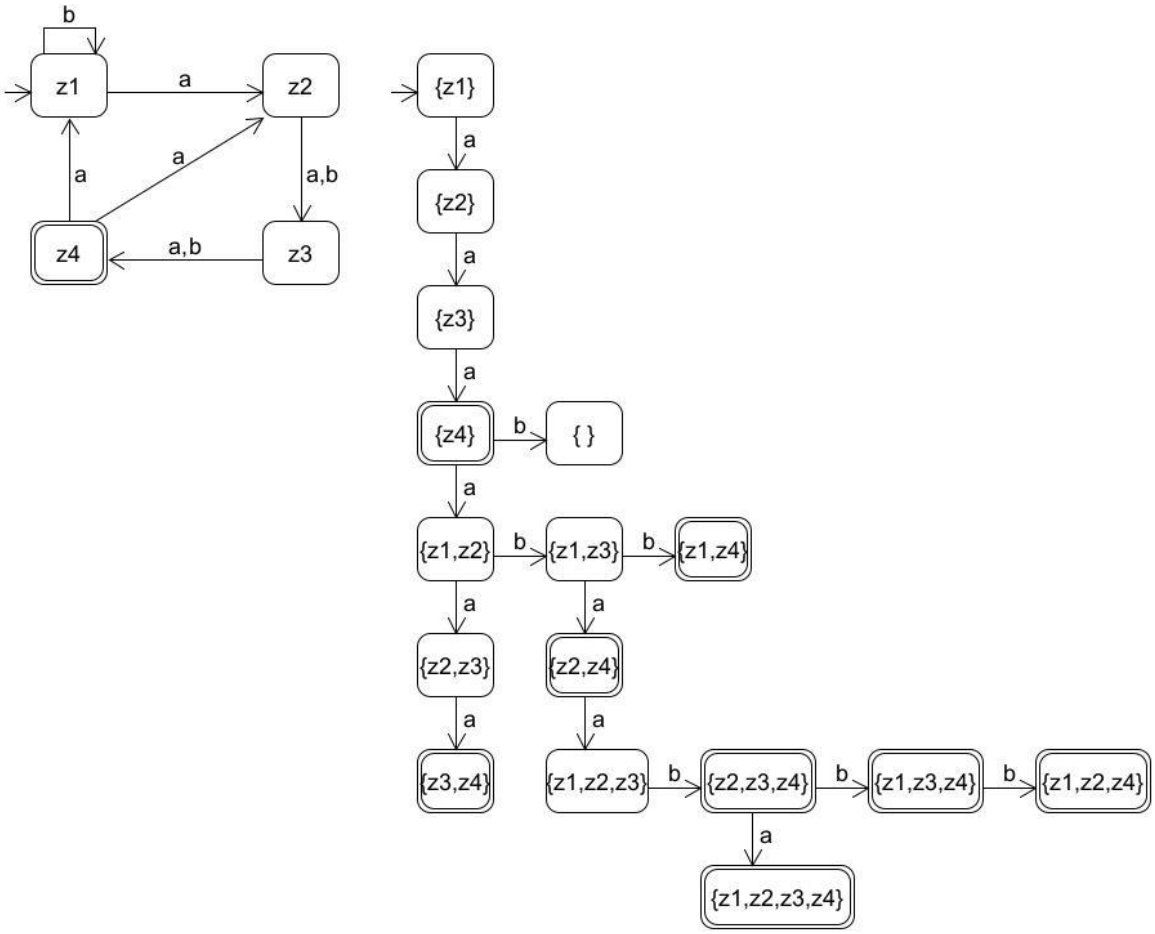
z4 -c->{z1,z2,z3,z4}

Es fehlt noch die leere Menge. Da alle Zustände für c schon ein eigenständiges Verhalten haben, wird ein neues Zeichen d benötigt. Dann muss es einen Zustand geben, der mit d in die leere Menge geht. Das wird hier vereinfachend für jeden Zustand gemacht, so dass d im Diagramm gar nicht vorkommt, aber zum Alphabet gehört. Der resultierende deterministische Automat sieht wie folgt aus.



Eine schönere Lösung findet sich bereits in dem folgenden Artikel, der innerhalb des Hochschulnetzes erreichbar ist.

F. R. Moore, On the Bounds for State-Set Size in the Proofs of Equivalence Between Deterministic, Nondeterministic, and Two-Way Finite Automata, IEEE Transactions on Computers, Volume C-20, Seiten 1211-1214, 1971, https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1671701



Die vorherige Abbildung zeigt links den Automaten für 4 Zustände, weitere Zustände würden so wie die Zustände z2 und z3 ergänzt, die einfach mit a und b in den nachfolgenden Zustand übergehen. Der Anfangs- und der Endzustand würde sich wie beim gezeigten Automaten verhalten.

Auf der rechten Seite steht nicht der vollständige deterministische Automat, sondern nur typische Pfade, wie die Zustände erreicht werden, was die Konstruktionsidee verdeutlichen soll. Zunächst schaltet a nur einen Zustand weiter. Im Endzustand wird zu zwei Zuständen verzweigt, wodurch die Anzahl der Elemente der erreichten Menge von Zuständen sich um eins erhöhen kann. Um alle Mengen von Zuständen mit der gleichen Elementanzahl zu erreichen, hat z1 eine Schleife mit b, während alle anderen Nicht-Endzustände einen Zustand weiterschalten.

Frage: Wie ist der logische Ansatz von dem Pumping-Lemma?

Antwort: (Zu) stark vereinfacht ist die Aussage die folgende: Wenn eine Sprache L von irgendeinem Automaten A akzeptiert werden kann (Sprache(A) = L), dann gibt es eine Wortlänge n, so dass alle Worte w, die in der Sprache L sind und die länger als n sind in Teile w=xyz zerlegt werden können, so dass für alle i xy^iz auch in der Sprache L ist.

Interessant ist, dass wir meistens die Negation nutzen. Dazu zunächst allgemein zur Negation. Ist die Aussage „wenn E dann F“ korrekt, dann ist auch die Aussage „wenn nicht F, dann nicht E“ korrekt. formaler:

E	F	wenn E dann F	wenn nicht F dann nicht E
wahr	wahr	wahr	wahr
wahr	falsch	falsch	falsch
falsch	wahr	wahr	wahr
falsch	falsch	wahr	wahr

Mit der Umkehrung kann „nicht E“, also hier „eine Sprache kann nicht von irgendeinem Autotaten akzeptiert werden“ gezeigt werden, was typischerweise unser Ziel ist.

Wir müssen dann „nicht F“ zeigen, was dann die Standardaufgabenstellung ist. Die lautet genauer: Zeige für alle Längen n, dass es ein Wort länger als n gibt, dass nicht aufgepumpt werden kann, es also mindestens ein i gibt, so dass xy^iz *nicht* in L liegt.

Zum Vorlesungsbeispiel $L = \{a^nb^m \mid n<m\}$: Wir wählen für eine beliebig große Länge n das Wort a^nb^{n+1} aus der Sprache, dies ist genauer eine „Struktur“, da ja n im Wort vorkommt und so „für alle Längen n“ beachtet wird. Dann wird, genau wie in den Folien, jede Zerlegungsmöglichkeit betrachtet und analysiert: Der aufpumbbare Teil y

- besteht nur aus a, dann wird ab i=2 (also einfaches aufpumpen) die Forderung n<m verletzt, das Wort ist nicht in der Sprache L
- besteht aus a’s gefolgt von b‘s, dann wird ab i=2 die Forderung erst nur a dann nur b verletzt, das Wort ist nicht in der Sprache L
- besteht nur aus b, dann wird für i=0 (also schrumpfen) die Forderung n<m verletzt, das Wort ist nicht in der Sprache L

Folglich haben wir die Nicht-Aufpumpbarkeit gezeigt und daraus folgt, dass es für diese Sprache keinen endlichen Automaten geben kann.

Frage: Wieso kann man generell mit dem Pumping-Lemma zeigen, dass eine Sprache nicht von einem Automaten akzeptierbar ist?

Antwort: (formaler) Dazu muss man zunächst das Pumping-Lemma genauer ansehen: Eine Sprache L kann nur kontextfrei sein, wenn

$$\exists\, n\in \text{NatuerlicheZahlen} \quad \forall\, w\in\{v \mid v\in L \wedge \text{länge}(v)>n\} \quad \exists\, x,y,z \quad w= xyz$$
$$\text{und } y\neq\varepsilon \text{ und } \text{länge}(xy) \leq n$$
$$\text{und } \forall\, i\in \text{NatuerlicheZahlen } xy^iz\in L$$

(∃ „es gibt“, ∀ „für alle“)

Um zu zeigen, dass eine Sprache nicht von einem Automaten akzeptierbar ist, muss für diese Sprache die Negation (¬) der obigen (prädikatenlogischen) Aussage gelten. Diese kann dann umgeformt werden mit ¬(∀x P) ist äquivalent zu (∃x ¬P) und ¬(∃x P) ist äquivalent zu (∀x ¬P)

$$\neg\exists\, n\in \text{NatuerlicheZahlen} \quad \forall\, w\in\{v \mid v\in L \wedge \text{länge}(v)>n\} \quad \exists\, x,y,z\ w=xyz$$
$$\text{und } y\neq\varepsilon \text{ und } \text{länge}(xy) \leq n$$
$$\text{und } \forall\, i\in \text{NatuerlicheZahlen } xy^iz\in L$$

$$\forall\, n\in \text{NatuerlicheZahlen} \quad \neg\forall\, w\in\{v \mid v\in L \wedge \text{länge}(v)>n\} \quad \exists\, x,y,z\ w=xyz$$

$$\text{und } y \neq \varepsilon \text{ und länge}(xy) \leq n$$

$$\text{und } \forall\, i \in \text{NatuerlicheZahlen } xy^i z \in L$$

$$\forall\, n \in \text{NatuerlicheZahlen} \quad \exists\, w \in \{v \mid v \in L \wedge \text{länge}(v) > n\} \quad \neg \exists\, x,y,z\ w = xyz$$

$$\text{und } y \neq \varepsilon \text{ und länge}(xy) \leq n$$

$$\text{und } \forall\, i \in \text{NatuerlicheZahlen } xy^i z \in L$$

$$\forall\, n \in \text{NatuerlicheZahlen} \quad \exists\, w \in \{v \mid v \in L \wedge \text{länge}(v) > n\} \quad \forall\, x,y,z\ \neg(w = xyz$$

$$\text{und } y \neq \varepsilon \text{ und länge}(xy) \leq n$$

$$\text{und } \forall\, i \in \text{NatuerlicheZahlen } xy^i z \in L)$$

$$\forall\, n \in \text{NatuerlicheZahlen} \quad \exists\, w \in \{v \mid v \in L \wedge \text{länge}(v) > n\} \quad \forall\, x,y,z\ w \neq xyz$$

$$\text{oder } y = \varepsilon \text{ oder länge}(xy) > n$$

$$\text{oder } \neg \forall\, i \in \text{NatuerlicheZahlen } xy^i z \in L)$$

$$\forall\, n \in \text{NatuerlicheZahlen} \quad \boxed{\exists\, w \in \{v \mid v \in L \wedge \text{länge}(v) > n\}} \quad \boxed{\forall\, x,y,z}\ w \neq xyz$$

$$\text{oder } y = \varepsilon \text{ oder länge}(xy) > n$$

$$\text{oder } \boxed{\exists\, i \in \text{NatuerlicheZahlen } xy^i z \notin L})$$

Der wichtigste Teil ist markiert und fordert, dass es ein Wort (abhängig von n) gibt, dass für alle Zerlegungen ein i durch aufpumpen (i>1) oder Schrumpfen (i=0) nicht in der Sprache liegt. Es muss also ein Wort für jedes n angegeben werden, dass egal wie nicht aufgepumpt werden kann. Ganz genau wird also nicht ein Wort sondern eine Wortstruktur angegeben, die nicht aufpumbbar ist, z. B. $a^n b^n$ mit den drei nicht aufpumpbaren Zerlegungsmöglichkeiten.

Frage: Bedeutet das Pumping-Lemma, dass wenn Worte aufpumpbar sind, die Sprache von einem Automaten akzeptierbar ist?

Antwort: Nein, die Formulierung fordert, dass dies für alle Worte ab einer bestimmten Länge gelten muss. Das Pumping-Lemma wird typischerweise nur eingesetzt, um zu zeigen, dass eine Sprache nicht von einem endlichen Automaten akzeptiert wird. Dazu muss dann nur ein einziges Wort (in Abhängigkeit von n) gefunden werden, dass nicht aufpumpbar ist (da die Zustandsanzahl nicht bekannt ist, muss das Wort flexibel in der Länge angegeben werden). In VL 11 wird dies mit einem Beispiel mit der ersten Folie deutlich gemacht.

Im Beispiel geht es um die Sprache $\{a^n b^m \mid n < m\}$. Das Wort, das den Widerspruch zum Pumping-Lemma liefert ist $a^k b^{k+1}$. Es werden alle Zerlegungsmöglichkeiten w=xyz geprüft, genauer Formen von y auf Aufpumpbarkeit geprüft:

y kann aus mehreren a und mehreren b bestehen, durch einmal Aufpumpen wäre das Wort nicht mehr in der Sprache

y kann aus einem oder mehreren a bestehen, durch mehrfaches Aufpumpen wird n<m verletzt und das Wort liegt nicht in der Sprache

y kann aus einem oder mehreren b bestehen, da y auch weggelassen werden kann („schrumpfen“), wird n<m verletzt und das Wort liegt nicht in der Sprache.

Damit ist die Form eines nicht aufpumpbaren Wortes gefunden, damit erfüllt die Sprache nicht die Forderung des Pumping-Lemmas und damit kann die Sprache nicht von einem endlichen Automaten akzeptiert werden.