Theoretische Informatik

Fragestunde

Matthias Heizmann, Christian Schilling

Software Engineering
Albert-Ludwigs-Universität Freiburg

22. Dezember 2017

Müssen Senkzustände in der (Probe-)Klausur angegeben werden?

Müssen Senkzustände in der (Probe-)Klausur angegeben werden?

Erklärung Senkzustand: Manche Automatenmodelle (bei uns nur DEA) oder Konstruktionen (Komplementierung von DPDA) erfordern, dass für jede Kombination aus Zustand und Zeichen (und ggf. Kellersymbol) ein Nachfolgezustand definiert ist. Gibt es in einem Automaten eigentlich keinen Nachfolger für eine bestimmte Kombination, können wir uns behelfen, indem wir einen „künstlichen Zustand“ einfügen, der in all solchen Fällen zum Nachfolger wird und ausschließlich Transitionen zu sich selbst hat. Wir nennen solch einen Zustand *Senkzustand*.

Antwort: Auch in der Klausur muss die Definition der Automatenmodelle eingehalten werden. Es könnte Aufgaben geben, in denen das Einführen eines Senkzustandes hilfreich ist.

Können Sie nochmals die Nerode-Relation erklären? Zum Beispiel anhand der Sprache $L = \{w \in \{a, b\}^* \mid \#_a(w) > \#_b(w)\}$.

Können Sie nochmals die Nerode-Relation erklären? Zum Beispiel anhand der Sprache $L = \{w \in \{a, b\}^* \mid \#_a(w) > \#_b(w)\}$.

Kurze Besprechung von Def. 2.9 im Skript: Für eine Sprache $L \subseteq \Sigma^*$ ist die *Nerode-Relation* wie folgt definiert.

$$R_L = \{(u, v) \mid \forall w \in \Sigma^* : uw \in L \Leftrightarrow vw \in L\}$$

Eine zusätzliche in der Fragestunde gegebene Erklärung und das Beispiel wurden nachträglich in das Vorlesungsskript eingefügt (Motivation nach Def. 2.8 und Beispiel nach Bsp 2.11).

Bei Aufgaben, in denen man reguläre Ausdrücke schreiben muss, ist teilweise nicht klar (bspw. @), was escaped werden muss. Ist es möglich, für die Klausuren dort Vorgaben zu geben (z.B. „alle Sonderzeichen müssen escaped werden“)?

Bei Aufgaben, in denen man reguläre Ausdrücke schreiben muss, ist teilweise nicht klar (bspw. @), was escaped werden muss. Ist es möglich, für die Klausuren dort Vorgaben zu geben (z.B. „alle Sonderzeichen müssen escaped werden“)?

Dies ist nur ein Problem in der Praxis, z.B. mit dem Programm `grep`. In der Klausur wird dieses Programm nicht benötigt. Daher wird dieses Problem nicht auftreten.

Ist es möglich, dass Ardens Lemma in der Zwischenklausur auftaucht? Es ist leider oft sehr schnell sehr rechenaufwändig und damit fehleranfällig.

Ist es möglich, dass Ardens Lemma in der Zwischenklausur auftaucht? Es ist leider oft sehr schnell sehr rechenaufwändig und damit fehleranfällig.

Ja, das Lemma (bzw. seine Anwendung) ist potentieller Klausurstoff. Generell gilt natürlich, dass Aufgaben in einer Klausur aus Zeitgründen kurz gehalten sind.

Was sind π und Y bei Ableitungsbäumen?

Was sind π und Y bei Ableitungsbäumen?

Wir verwenden π typischerweise um Regeln (also Elemente der dritten Komponente einer Grammatik (Σ, N, P, S)) zu benennen.

Das *abgeleitete Wort* $Y(\mathcal{T})$ ist die in Def. 3.4 eingeführte Funktion, die einen Ableitungsbaum $\mathcal{T}$ auf das durch ihn repräsentierte/abgeleitete Wort abbildet.

Kontextfreies Pumping Lemma: Tipps für (1) die Wortfindung und (2) wie man am besten die Fälle aufteilt

Kontextfreies Pumping Lemma: Tipps für (1) die Wortfindung und (2) wie man am besten die Fälle aufteilt

Die Wahl des Worts $z = uvwxy$ ist zentral. Sie entscheidet, ob der Beweis überhaupt möglich ist und, wenn ja, wie viele Fälle man unterscheiden muss. Grundsätzlich muss man versuchen, die wenigen Zusicherungen aus dem PL auszunutzen. Hauptsächlich relevant ist hierbei die Regel $|vwx| \leqslant n$. Man sollte also z so wählen, dass man ein beliebiges Teilwort der Länge n durch Pumpen aus der Sprache fallen lassen kann. Im Beispiel $L_1 = \{a^k b^k c^k \mid k \in \mathbb{N}\}$ haben wir $z = a^n b^n c^n$ gewählt, damit wir keinen Fall betrachten müssen, in dem sowohl a als auch c in vwx vorkommen.

Manchmal können wir für verschiedene Sprachen das gleiche Wort z wählen. Beispiel: Für die Sprache $L_2 = \{w \in \{a, b, c\}^* \mid \#_a(w) = \#_b(w) = \#_c(w)\}$ können wir wieder $z = a^n b^n c^n$ wählen. Der Beweis ist danach 1:1 der gleiche wie für L_1.

(Diese Aussagen gelten analog auch für das PL für reguläre Sprachen.)

Sind DPDA, die mit leerem Keller akzeptieren, äquivalent zu DPDA?

Sind DPDA, die mit leerem Keller akzeptieren, äquivalent zu DPDA?

Nein.

Ein DPDA, der mit leerem Keller akzeptiert, und die von ihm akzeptierte Sprache werden definiert, wie man es erwarten würde. Dieses Automatenmodell hat die folgende Eigenschaft: Wird ein Wort w akzeptiert, so kann keine Präfix von w akzeptiert werden. Es gibt also z.B. keinen DPDA, der mit leerem Keller die durch den regulären Ausdruck a^* beschriebene Sprache akzeptiert.

Interessante Tatsache: Die Menge der Sprachen, die von diesem Automatenmodell (DPDA, die mit leerem Keller akzeptieren) akzeptiert werden, sind weder eine Obermenge noch eine Teilmenge der regulären Sprachen.

Bemerkung: DPDA, die mit leerem Keller akzeptieren, wurden in der Vorlesung nicht eingeführt. Diese Frage geht daher über den Vorlesungsstoff hinaus.

Sind NPDA mit mehreren Kellerspeichern gleichmächtig, egal wie viele Kellerspeicher sie haben?

Sind NPDA mit mehreren Kellerspeichern gleichmächtig, egal wie viele Kellerspeicher sie haben?

Die Antwort hängt davon ab, was wir unter „Mächtigkeit“ verstehen.
Bezieht sich „Mächtigkeit“ auf die Menge von Sprachen, die von Instanzen dieses Automatenmodells akzeptiert werden können, dann lautet die Antwort:

„Nein, schon ein zweiter Kellerspeicher genügt, um eine Turingmaschine zu simulieren.“

Bemerkung: NPDA mit mehreren Kellerspeichern wurden in der Vorlesung bisher nicht besprochen. Diese Frage geht daher über den Vorlesungsstoff hinaus.

Wieso definiert man Turingmaschinen nicht mit unendlich vielen Zuständen, obwohl man praktisch diesen Effekt hat, da es beliebig viele sein können?

Wieso definiert man Turingmaschinen nicht mit unendlich vielen Zuständen, obwohl man praktisch diesen Effekt hat, da es beliebig viele sein können?

Es gibt hier möglicherweise ein Missverständnis:

Mit *beliebig viele* bezeichnen wir eine endliche Anzahl, die nicht durch eine natürliche Zahl beschränkt ist. So kann z.B. auch ein NEA oder ein NPDA beliebig viele Zustände haben.

Begründungen zur eigentlichen Frage:

- Turingmaschinen werden auch in der Literatur mit endlich vielen Zuständen definiert.
- Wir möchten eine Definition, die auf den vorherigen Automatenmodellen aufbaut.
- Automatenmodelle mit endlich vielen Zuständen (und endlich vielen Transitionen) können leicht repräsentiert werden. Gäbe es unendlich viele Zustände, müssten wir uns erst einen Formalismus überlegen um diese zu repräsentieren.

Sind alle Turingmaschinen mit beliebig vielen Bändern aus dem gleichen Grund äquivalent, aus dem es für jeden NPDA einen NPDA mit einem Zustand gibt (Skript S. 59 Ende)? (Wird dieselbe Information einmal durch Zustände im NPDA und einmal durch Symbole auf Bändern bei einer TM gespeichert)?

Sind alle Turingmaschinen mit beliebig vielen Bändern aus dem gleichen Grund äquivalent, aus dem es für jeden NPDA einen NPDA mit einem Zustand gibt (Skript S. 59 Ende)? (Wird dieselbe Information einmal durch Zustände im NPDA und einmal durch Symbole auf Bändern bei einer TM gespeichert)?

Wir haben in der Vorlesung noch gar nicht über Mehrbandturingmaschinen gesprochen. In der Tat sind diese nicht mächtiger. Die Grundidee zur Simulation ist, dass man zu einem beliebigen Zeitpunkt immer nur einen endlichen Teil eines Bands besucht hat. Daher kann man mehrere Bänder auf einem einzigen Band abbilden, indem man (neue) Trennsymbole verwendet. Sollte der benötigte Platz für eines der Bänder größer werden, unterbricht man die normale Berechnung und kopiert alle Symbole um ein Feld nach rechts.

Wieso ist eine Konfiguration einer Turingmaschine als Tripel (v, q, w) definiert statt als 4-Tupel (q, v, a, w') mit q = Zustand, v = linke Bandhälfte, a = aktuelles Symbol, w' = rechte Bandhälfte?

Wieso ist eine Konfiguration einer Turingmaschine als Tripel (v, q, w) definiert statt als 4-Tupel (q, v, a, w') mit q = Zustand, v = linke Bandhälfte, a = aktuelles Symbol, w' = rechte Bandhälfte?

Das ist eine willkürliche Festlegung, die Vor- und Nachteile hat.

In einer Konfiguration (v, q, w) wird $w \in \Gamma^+$ verlangt, d.h., w hat die Form aw' mit $a \in \Gamma$ und $w' \in \Gamma^*$. Wir können also immer das aktuelle Zeichen explizit machen, indem wir (v, q, aw') schreiben. Manchmal interessieren wir uns aber nicht für das aktuelle Zeichen. In diesem Fall ist die Darstellung als 4-Tupel etwas umständlicher.