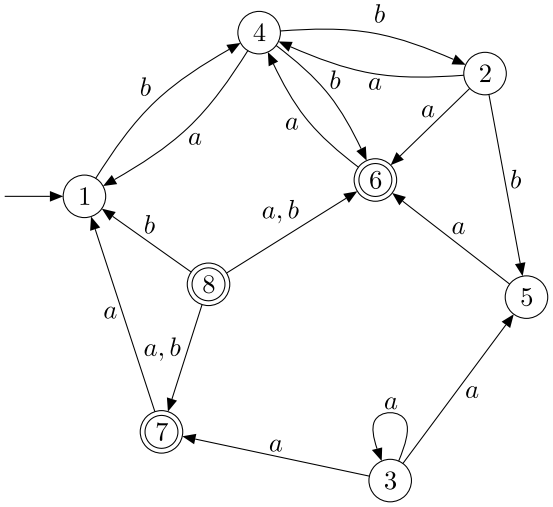
Formale Systeme, Automaten, Prozesse, SS 2020
Prof. Dr. P. Rossmanith
J. Dreier, T. Hartmann, H. Lotze, D. Mock

Datum: 1.6.2020

Übungsblatt 08

In der Exkursionswoche gibt es keine verpflichtenden Hausaufgaben. Sie können aber eine Auswahl der folgenden Aufgaben zur weiteren Übung bearbeiten und zur Korrektur abgeben. Es gibt aber keine Punkte für die Klausurzulassung.

Aufgabe H22 (0 Punkte)
Konstruieren Sie den minimalen deterministischen Automaten zu folgendem NFA:

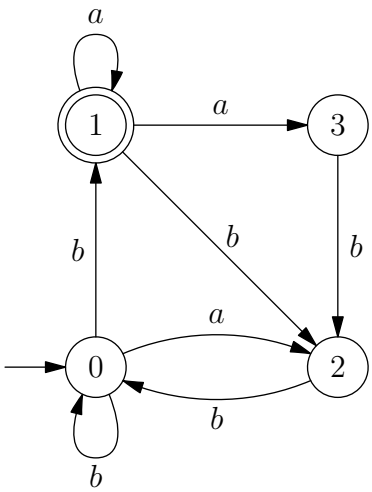


Aufgabe H23 (0 Punkte)
Seien L und R zwei reguläre Sprachen über einem Alphabet Σ. Ist die Sprache

$$L \star R := \{vxw \mid vw \in L, x \in R\}$$

regulär? Beweisen Sie Ihre Aussage! Beispiel: Sei $L = L((ab)^+), R = L(c^+)$. Dann sind *abccc* und *abacb* in $L \star R$, aber nicht *abab*, *abacab* oder *abcccabcab*.
Hinweis: Wenn Sie die Aussage konstruktiv beweisen wollen, müssen Ihre Konstruktion formal angeben und zeigen, dass diese richtig ist (siehe Lösungsvorschlag H5). Wenn Sie die Aussage widerlegen wollen, müssen Sie zwei reguläre, geeignete Sprachen für L und R whälen und beweisen (!), dass $L \star R$ nicht regulär ist.

Aufgabe H24 (0 Punkte)
Gegeben sei ein Netzwerkprotokoll über dem Alphabet $\{a, b\}$. Korrekt kodierte Nachrichtenpakete sind durch nebenstehenden NFA spezifiziert.



a) Ist *bbbabbaaa* eine korrekt kodierte Nachricht?

b) Konstruieren Sie sowohl einen DFA, als auch einen regulären Ausdruck für alle falsch kodierten Nachrichten. Erklären Sie, welche Konstruktionen Sie dabei verwenden.

Aufgabe H25 (0 Punkte)

Entwerfen Sie für jede Zahl $k \in \mathbf{N}$ eine kontextfreie Sprache, die nicht von einer Grammatik mit k Nichtterminalen erzeugt werden kann und beweisen Sie, dass es auch wirklich so ist.

Aufgabe H26 (0 Punkte)

Gegeben seien reguläre Ausdrücke $R = \{r_1, \ldots, r_n\}$. Entwickeln Sie ein Verfahren, dass für ein gegebenes Wort w entscheidet, ob es reguläre Ausdrücke $q_1, \ldots, q_k \in R$ und Wörter $u_1, \ldots, u_k$ gibt, die die folgenden Bedingungen erfüllen: Für $1 \leq i \leq k$ gilt

1. $w_1 = w$ und $w_{k+1} = \epsilon$,
2. $u_i \neq \epsilon$ ist der längste Präfix von w_i, für den $u_i \in L(r_j)$ für ein $j \in \{1, \ldots, n\}$ gilt,
3. $u_i \in L(q_i)$ mit $q_i = r_j$ und $u_i \notin L(r_k)$ für $k < j$,
4. w_{i+1} entsteht durch Entfernen des Präfixes u_i von w_i, also $w_i = u_i w_{i+1}$.

Das Verfahren soll weiterhin die Ausdrücke $q_1, \ldots, q_k$ und Wörter $u_1, \ldots, u_k$ ausgeben, falls sie existieren.

Konzipieren Sie Ihr Verfahren so, dass es möglichst schnell ist, wenn w sehr lang ist. Dafür soll das Verfahren zuerst die regulären Ausdrücke R einlesen und dann eine (möglicherweise langsame) Vorverarbeitung durchführen. Danach soll das Wort w sehr schnell bearbeitet werden.

Beispiel: Es sei

$$R = \{\texttt{for}, \texttt{while}, \texttt{if}, \texttt{;}, \texttt{<}, \texttt{+}, \texttt{++}, \texttt{(}, \texttt{)}, \texttt{\{}, \texttt{\}}, \texttt{=}, r_{num}, r_{id}\},$$

mit $r_{num} = (\texttt{0} + \cdots + \texttt{9})^+$ und $r_{id} = (\texttt{a} + \cdots + \texttt{z})^+(\texttt{a} + \cdots + \texttt{z} + \texttt{0} + \cdots + \texttt{9})^*$.

Auf einem Wort $w = \texttt{for(i=0;i<10;i++)\{print(i);\}}$ erhalten wir Prefixe $u_1, \ldots, u_{20}$:

$$\texttt{for}, \texttt{(}, \texttt{i}, \texttt{=}, \texttt{0}, \texttt{;}, \texttt{i}, \texttt{<}, \texttt{10}, \texttt{;}, \texttt{i}, \texttt{++}, \texttt{)}, \texttt{\{}, \texttt{print}, \texttt{(}, \texttt{i}, \texttt{)}, \texttt{;}, \texttt{\}}$$

Aufgabe H27 (0 Punkte)

In Aufgabe H26 wurde ein Vorschlag erarbeitet, wie man *first longest match* implementieren könnte. Dies ist sehr aufwendig, umfaßt es ja Konstruktionen wie das Entfernen von ϵ-Kanten, die Potenzmengenkonstruktion und das Minimieren von DFAs. Sie können dennoch das ganze Verfahren implementieren, wenn Sie wollen, sollten das aber in einer Gruppe machen und die Arbeit aufteilen.

Stattdessen können Sie auch eine einfachere Aufgabe lösen: Überlegen Sie sich ein einfacheres Verfahren, um einen Spezialfall von H26 zu lösen: Die regulären Ausdrücke umfassen neben r_{num} und r_{id}, welche am Ende der Liste stehen müssen, nur reguläre Ausdrücke, welche ein einzelnes Wort beschreiben (wie es auch im Beispiel in H26 der Fall ist). Es ist nicht schwer, in diesem Fall direkt einen geeigneten DFA zu konstruieren und während dieser simuliert wird, die entsprechenden u_i zu bestimmen.

Beschreiben Sie, wie Ihr Verfahren funktioniert und implementieren Sie es. Testen Sie es an aussagekräftigen Beispielen.

Ihr Programm sollte als Ausgabe eine Liste von *Token* liefern, welche wenigstens das jeweilige u_i und einen symbolischen Namen, der den regulären Ausdruck identifiziert, beinhaltet.

Geben Sie auch den Quelltext des Programms sowie Protokolle von Testläufen ab. Diese Protokolle mögen auch die jeweiligen Listen von *Token* in übersichtlicher Form enthalten.

Aufgabe H28 (0 Punkte)

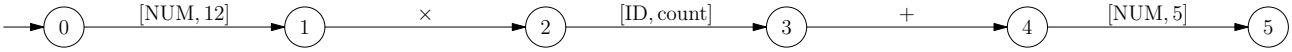
Schreiben Sie ein Programm, welches einen NFA M und eine Menge von Produktionen als Eingabe erhält. Es soll dann M mit den Produktionen sättigen.

Dabei müssen aber noch Zusatzinformationen gespeichert werden: Falls eine Transition $q_0 \overset{A}{\to} q_k$ zu M aufgrund einer Regel $A \to \beta = \beta_1 \dots \beta_k$ und den bereits existierenden Transitionen $q_0 \overset{\beta_1}{\to} q_1$, $q_1 \overset{\beta_2}{\to} q_2, \dots$, $q_{k-1} \overset{\beta_k}{\to} q_k$ hinzugefügt wird, soll dies gespeichert werden. Dies könnte man zum Beispiel in einer Datenstruktur B speichern, die um folgendes ergänzt wird:

$$B(q_0 \overset{A}{\to} q_k) \mapsto [q_0 \overset{\beta_1}{\to} q_1, q_1 \overset{\beta_2}{\to} q_2, \dots, q_{k-1} \overset{\beta_k}{\to} q_k]$$

Als Ausgabe möge Ihr Programm dann das modifizierte M und vor allem auch die Abbildung B zurückliefern.

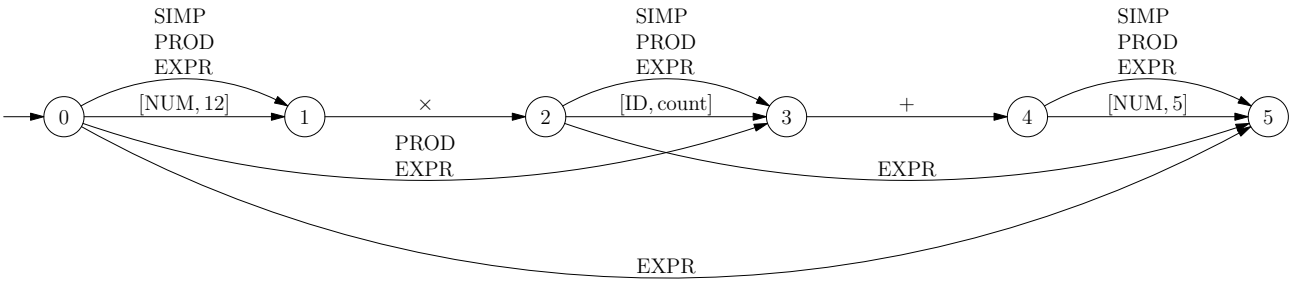
Hier ist ein Beispiel für einen Automaten, der *Token* als Alphabet besitzt. Ein Token hat eine *id* und möglicherweise einen zusätzlichen Quelltext. Das Token [NUM, 12] ist also ein NUM-Token mit dem zugehörigen Quelltext 12.



Wir verwenden folgende Produktionen:

$$\begin{aligned} \text{EXPR} &\to \text{EXPR} + \text{PROD} \mid \text{EXPR} - \text{PROD} \mid \text{PROD} \\ \text{PROD} &\to \text{PROD} \times \text{SIMP} \mid \text{PROD}/\text{SIMP} \mid \text{SIMP} \\ \text{SIMP} &\to \text{NUM} \mid \text{ID} \end{aligned}$$

Der gesättigte Automat sieht dann folgendermaßen aus.



Woher kam die Transition $0 \overset{\text{EXPR}}{\to} 5$?

Wir sehen nach: $B(0 \overset{\text{EXPR}}{\to} 5) = \big(0 \overset{\text{EXPR}}{\to} 3, 3 \overset{+}{\to} 4, 4 \overset{\text{PROD}}{\to} 5\big)$.

Demonstrieren Sie die Fähigkeiten Ihres Programms an aussagekräftigen Beispielen, insbesondere am obigen. Kombinieren Sie Ihr Programm mit jenem aus H13, um ein Programm zu erhalten, das solche einfachen arithmetischen Ausdrücke syntaktisch überprüfen kann.

Wie könnten Sie Ihr Programm nutzen, um auch einen Ableitungsbaum zu konstruieren?

Aufgabe H29 (0 Punkte)

Die Fibonaccizahlen sind so definiert: $F_0 = 0$, $F_1 = 1$, $F_n = F_{n-1} + F_{n-2}$ für $n \geq 2$.

Es sei L die Sprache aller Wörter, die eine Fibonaccizahl in Binärdarstellung über dem Alphabet $\{0, 1\}$ kodieren. Also ist $L = \{0, 1, 10, 11, 101, 1000, 1101, \dots\}$.

Beweisen oder widerlegen Sie: L ist kontextfrei.

Bonusbonusaufgabe:

Die Sprache L' entstehe aus L indem wir die Wörter aus L nehmen und jedes Zeichen außer der ersten zwei durch das Symbol \$ ersetzen. Also:

$$L' = \{0, 1, 10, 11, 10\$, 10\$\$, 11\$\$, \dots\}$$

Beweisen oder widerlegen Sie: L' ist kontextfrei.