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

Datum: 25.05.2020

Übungsblatt mit Lösungen 07

Aufgabe T21

Sei $G = (N, T, P, S)$ folgende kontextfreie Grammatik mit $N = \{S, A, B, C, D\}$, $T = \{a, b, c, d\}$ und P :

$$\begin{array}{lcl} S & \to & BDA \mid DAC \mid DC \\ A & \to & SA \mid aA \\ B & \to & \epsilon \mid Sc \mid cD \mid AD \\ C & \to & aD \mid dA \mid bC \mid a \mid d \mid b \\ D & \to & BS \mid cC \mid aA. \end{array}$$

Dies ist die gleiche Grammatik wie die von letzter Woche, nur dass es für A weniger Produktionen gibt.

a) Ist $L(G)$ endlich?

b) Ist $L(G)$ universell, d.h. $L(G) = \{a, b, c, d\}^*$?

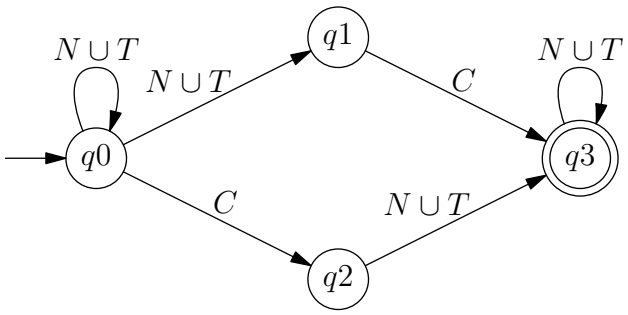
Lösungsvorschlag

Um die Fragen zu beantworten, benötigen wir eine Grammatik G' mit $L(G') = L(G)$, die keine unproduktiven, unerreichbaren oder nullierbaren Symbole enthält. Hier ist A unproduktiv und B nullierbar. Für die Grammatik G ergibt sich also folgende Grammatik G':

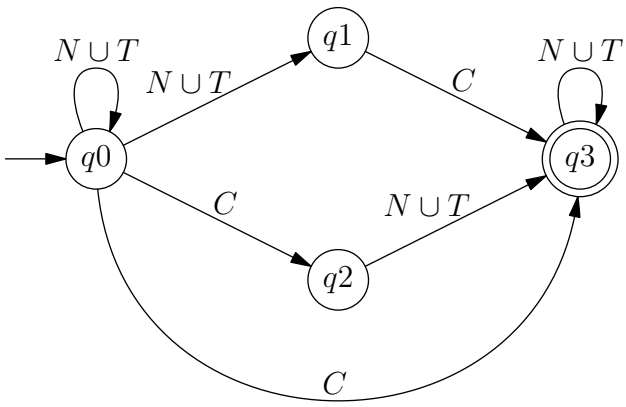
$$\begin{array}{lcl} S & \to & DC \\ B & \to & Sc \mid cD \\ D & \to & BS \mid S \mid cC \\ C & \to & aD \mid bC \mid a \mid d \mid b \end{array}$$

a) Aus der Vorlesung wissen wir: $L(G)$ ist genau dann endlich, falls es kein $A \in N$ gibt, so dass $A \in pre^*_{G'}(L)$, mit $L = (N \cup T)^+ A (N \cup T)^* \cup (N \cup T)^* A (N \cup T)^+$.

Wähle nun $A = C$. Wir konstruieren uns einen Automaten für L:



Nun können wir einen Saturierungsschritt für die Regel $C \to bC$ anwenden, (da $b \in N \cup T$) für die Transitionen $\delta(q_0, N \cup T) = q_1$ und $\delta(q_1, C) = q_3$, welches die Transition $\delta(q_0, C) = q_3$ liefert.



Demnach ist $C \in pre^*\left((N \cup T)^+ C (N \cup T)^* \cup (N \cup T)^* C (N \cup T)^+\right)$ und die Sprache damit unendlich.

b) Nein, denn es gilt $\epsilon \notin L(G')$, da G' keine nullierbaren Symbole enthält.

Aufgabe T22

Bringen Sie die folgende Grammatik $G = (N, T, P, S)$ erst in Chomsky- und dann in Greibach-Normalform. Hier ist $N = \{S, A, B, C\}$, $T = \{a, b, c\}$ und P:

$$S \to ABC \mid a, \quad A \to Sa, \quad B \to Sb, \quad C \to cS.$$

Lösungsvorschlag

Zuerst bringen wir die Grammatik in Chomsky-Normalform und erhalten

$$S \to AX \mid a, \quad X \to BC, \quad A \to SR_a, \quad B \to SR_b, \quad C \to R_c S$$

$$R_a \to a, \quad R_b \to b, \quad R_c \to c$$

Nun kümmern wir uns um S und erhalten zunächst

$$S \to SR_a X \mid a, \quad X \to BC, \quad A \to SR_a, \quad B \to SR_b, \quad C \to cS$$

und dann

$$S \to aZ \mid a, \quad Z \to aX \mid aXZ, \quad X \to BC, \quad A \to SR_a, \quad B \to SR_b, \quad C \to cS.$$

Abschließend betrachten wir A, B und X. Für A und B ist die Auflösung einfach, da die rechten Seiten mit S beginnen, S aber schon konvertiert ist. Danach kann auch X umgewandelt werden, und insgesamt ergibt sich

$$S \to aZ \mid a, \quad Z \to aX \mid aXZ, \quad C \to cS, \quad A \to aZR_a \mid aR_a$$

$$B \to aZR_b \mid aR_b, \quad X \to aZR_bC \mid aR_bC$$

$$R_a \to a, \quad R_b \to b, \quad R_c \to c$$

Aufgabe T23

Es sei $L = \{\, p\$p\$p \mid p \in \{a, b\}^* \,\}$. Benutzen Sie das Pumping-Lemma um zu zeigen, dass L nicht durch eine kontextfreie Grammatik erzeugt werden kann.

Lösungsvorschlag
Angenommen L sei kontextfrei. Dann existiert ein n, so dass jedes Wort $z \in L$ mit $|z| \geq n$ in $z = uvwxy$ zerlegt werden kann mit

1. $|vwx| \leq n$,
2. $|vx| \geq 1$ und
3. $uv^iwx^iy \in L$ für alle $i \in \mathbf{N}_0$.

Sei $z = a^n\$a^n\$a^n$. Offensichtlich gilt $z \in L$. Falls vx mindestens ein \$ enthält, so ist $uv^0wx^0y = uwy$ offensichtlich nicht in L enthalten, da dieses Wort noch höchstens ein \$ enthält.
Wir können also davon ausgehen, dass vx kein \$ enthält. Für jede solche Zerlegung die (1) und (2) erfüllt, gilt aber dass vwx höchstens zwei der getrennten a-Blöcke überlappt, da $|vwx| \leq n$. Damit ist aber $uv^2wx^2y \notin L$, da mindestens einer der drei a-Blöcke genau die Länge n hat, während mindestens einer der beiden anderen Blöcke mindestens die Länge $n + 1$ hat.

Aufgabe H19 (10 Punkte)
Schreiben Sie ein Programm, welches boolsche Ausdrücke evaluieren kann, wie sie in Aufgabe H18 definiert wurden. Es bietet sich an einen *recursive descent parser* zu entwickeln, ähnlich zu jenem der in der Vorlesung für arithmetische Ausdrücke vorgestellt wurde. Zu einem booleschen Ausdruck sollte die Ausgabe `True` oder `False` erzeugt werden, wenn der Ausdruck syntaktisch korrekt ist und andernfalls ein Fehler gemeldet werden.
Führen Sie Ihr Programm für alle Ausdrücke aus, die in den Zeilen der Datei `boolsche-ausdruecke` enthalten sind, welche sich im Moodleraum befindet. Dort finden sich Zeilen wie zum Beispiel `~1|(1|0&(0))&1`, welche dem boolschen Ausdruck $\neg 1 \vee (1 \vee 0 \wedge (0)) \wedge 1$ entspricht. Geben Sie die Ausgabe des Programms mit ab.

Hinweis: Verwenden Sie eine Grammatik, die nicht links-rekursiv ist.

Bonuspunkte: Überlegen Sie sich, wie möglichst nützliche Fehlermeldungen erzeugt werden können. Diese sollten die Stelle und die Art des Fehlers anzeigen. Erläutern Sie Ihre Entscheidungen und deren Implementierung. Zeigen Sie viele aussagekräftige Beispiele. Die auf der Webseite vorgegebenen Ausdrücke sind alle syntaktisch korrekt.

Lösungsvorschlag
Wir haben den Parser in Python implementiert, dieser ist in Abbildung 1 zu finden. Die richtige Ausgabe ist:

1) `False`	9) `True`	17) `True`	25) `True`
2) `True`	10) `True`	18) `True`	26) `True`
3) `False`	11) `True`	19) `True`	27) `False`
4) `True`	12) `False`	20) `False`	28) `True`
5) `False`	13) `True`	21) `True`	29) `True`
6) `False`	14) `True`	22) `True`	30) `True`
7) `True`	15) `True`	23) `True`	31) `False`
8) `True`	16) `False`	24) `True`	32) `False`

33) `True`
34) `True`
35) `True`
36) `True`
37) `True`
38) `True`
39) `True`
40) `False`
41) `True`
42) `False`
43) `True`
44) `True`
45) `True`
46) `True`
47) `True`
48) `True`
49) `True`
50) `True`
51) `False`
52) `True`
53) `True`
54) `True`
55) `False`
56) `True`
57) `True`
58) `True`
59) `True`
60) `False`
61) `True`
62) `True`
63) `True`
64) `False`
65) `False`
66) `False`
67) `False`
68) `False`
69) `True`
70) `True`
71) `True`
72) `False`
73) `False`
74) `True`
75) `True`
76) `True`
77) `True`
78) `False`
79) `True`
80) `True`
81) `True`
82) `True`
83) `False`
84) `True`
85) `True`
86) `True`
87) `True`
88) `True`
89) `True`
90) `False`
91) `True`
92) `True`
93) `True`
94) `True`
95) `False`
96) `True`
97) `False`
98) `True`
99) `False`
100) `True`

Auf der Website und im Moodle-Raum waren durch unser Ungeschick zwei verschiedene Versionen der Datei `boolsche-ausdruecke` vorhanden. Die korrekte Lösung für die (alte) Version im Moodle-Raum lautet:

1) `False`
2) `False`
3) `False`
4) `False`
5) `False`
6) `True`
7) `True`
8) `True`
9) `True`
10) `False`
11) `True`
12) `True`
13) `False`
14) `True`
15) `False`
16) `True`
17) `True`
18) `True`
19) `True`
20) `True`
21) `False`
22) `True`
23) `True`
24) `True`
25) `True`
26) `False`
27) `True`
28) `False`
29) `True`
30) `True`
31) `False`
32) `True`
33) `True`
34) `True`
35) `True`
36) `False`
37) `False`
38) `True`
39) `True`
40) `True`
41) `True`
42) `True`
43) `True`
44) `True`

45) `True`
46) `True`
47) `True`
48) `True`
49) `True`
50) `True`
51) `True`
52) `False`
53) `True`
54) `True`
55) `True`
56) `True`
57) `True`
58) `True`
59) `False`
60) `True`
61) `True`
62) `False`
63) `True`
64) `True`
65) `True`
66) `True`
67) `True`
68) `True`
69) `True`
70) `False`
71) `True`
72) `False`
73) `True`
74) `True`
75) `True`
76) `True`
77) `False`
78) `True`
79) `True`
80) `False`
81) `True`
82) `False`
83) `False`
84) `False`
85) `True`
86) `True`
87) `True`
88) `False`
89) `True`
90) `True`
91) `True`
92) `False`
93) `True`
94) `True`
95) `True`
96) `False`
97) `False`
98) `True`
99) `True`
100) `True`

Aufgabe H20 (10 Punkte)
In der Aufgabe H18 entwarfen wir kontextfreie Grammatiken für boolsche Ausdrücke. Erzeugen Sie nun kontextfreie Grammatiken in Chomsky- und Greibach-Normalform für jene Sprache.

Lösungsvorschlag
Entfernung ε-Regeln, Ersetzung der Non-Terminale ergibt:

$$S \to R_(S R_) \mid R_0 \mid R_1 \mid R_\neg S \mid S R_\wedge S \mid S R_\vee S$$

Zudem Regeln $R_t \to t$ für jedes Terminal $t \in T$. Diese lassen wir im Folgenden der Übersicht halber weg.
Ersetzung der "langen Regeln" ergibt:

$$\begin{aligned} S &\to R_(S_1 \mid R_0 \mid R_1 \mid R_\neg S \mid S S_2 \mid S S_3 \\ S_1 &\to S R_) \\ S_2 &\to R_\wedge S \\ S_3 &\to R_\vee S \end{aligned}$$

Ersetzung der Regel $S \to R_0$ und $S \to R_1$ ergibt folgende Regeln für die Grammatik in Chomsky-Normalform:

$$\begin{aligned} S &\to R_(S_1 \mid 0 \mid 1 \mid R_\neg S \mid R_\neg R_0 \mid R_\neg R_1 \mid S S_2 \mid R_0 S_2 \mid R_1 S_2 \mid S S_3 \mid R_0 S_3 \mid R_1 S_3 \\ S_1 &\to S R_) \mid 0 R_) \mid 1 R_) \\ S_2 &\to R_\wedge S \mid R_\wedge R_0 \mid R_\wedge R_1 \\ S_3 &\to R_\vee S \mid R_\vee R_0 \mid R_\vee R_1 \end{aligned}$$

```
import sys

class parser :
    def __init__(self, line) :
        self.line = line.strip() + '$'
        self.n = 0

    def parse(self) :
        value = self.D()
        if self.next() ≠ '$' :
          self.err('extra symbols at the end.')
        return value

    def next(self) :
        return self.line[self.n]

    def gobble(self) :
        self.n = self.n + 1

    def err(self, msg) :
        print(self.line)
        msg = '^ Syntax error: ' + msg
        for i in range(self.n) :
            msg = ' ' + msg
        print(msg)
        sys.exit()

    def check(self, a) :
        if line[self.n] ≠ a :
            self.err(a + " expected.")

    def D(self) :
        value = self.C()
        if self.next() == '|' :
            self.gobble()
            value = self.D() or value
        return value

    def C(self) :
        value = self.P()
        if self.next() == '&' :
            self.gobble()
            value = self.C() and value
        return value

    def P(self) :
        if self.next() == '0' :
            self.gobble()
            value = False
        elif self.next() == '1' :
            self.gobble()
            value = True
        elif self.next() == '~' :
            self.gobble()
            value = not self.P()
        elif self.next() == '(' :
            self.gobble()
            value = self.D()
            self.check(')')
            self.gobble()
        else :
            self.err('0, 1, ~, or ( expected.')
        return value

for line in sys.stdin :
    p = parser(line)
    print(p.parse())
```

Abb. 1: Implementierung des Recursive Descent Parsers in Python.

Kein Symbol ist nullierbar, und kein Symbol ist unerreichbar oder unproduktiv.
Wir bringen die Grammatik in Greibach-Normalform nach Algorithmus aus der Vorlesung (ausgehend von der Grammatik oben in Chomsky-Normalform, was nach Aufgabenstellung nicht unbedingt verlangt ist). Wir wählen die Reihenfolge der Non-Terminale $R_(, R_), R_\neg, R_\wedge, R_\vee, R_0, R_1, S, S_1, S_2, S_3$
Schritte $R_(, \ldots, R_1, S$:

$$
\begin{aligned}
&S \to (S_1 \mid 0 \mid 1 \mid \neg S \mid \neg R_0 \mid \neg R_1 \mid 0S_2 \mid 1S_2 \mid 0S_3 \mid 1S_3 \\
&\mid (S_1Z \mid 0Z \mid 1Z \mid \neg SZ \mid \neg R_0Z \mid \neg R_1Z \mid 0S_2Z \mid 1S_2Z \mid 0S_3Z \mid 1S_3Z \\
&S_1 \to SR_) \mid R_0R_) \mid R_1R_) \\
&S_2 \to R_\wedge S \mid R_\wedge R_0 \mid R_\wedge R_1 \\
&S_3 \to R_\vee S \mid R_\vee R_0 \mid R_\vee R_1 \\
&Z \to S_2Z \mid S_3Z \mid S_2 \mid S_3
\end{aligned}
$$

Schritt S_1, S_2, S_3:

$$
\begin{aligned}
&S \to (S_1 \mid 0 \mid 1 \mid \neg S \mid \neg R_0 \mid \neg R_1 \mid 0S_2 \mid 1S_2 \mid 0S_3 \mid 1S_3 \\
&\mid (S_1Z \mid 0Z \mid 1Z \mid \neg SZ \mid \neg R_0Z \mid \neg R_1Z \mid 0S_2Z \mid 1S_2Z \mid 0S_3Z \mid 1S_3Z \\
&S_1 \to (S_1R_) \mid 0R_) \mid 1R_) \mid \neg SR_) \mid \neg R_0R_) \mid \neg R_1R_) \mid 0S_2R_) \mid 1S_2R_) \mid 0S_3R_) \mid 1S_3R_) \\
&\mid (S_1ZR_) \mid 0ZR_) \mid 1ZR_) \mid \neg SZR_) \mid \neg R_0ZR_) \mid \neg R_1ZR_) \mid 0S_2ZR_) \mid 1S_2ZR_) \mid 0S_3ZR_) \mid 1S_3ZR_) \\
&\mid 0R_) \mid 1R_) \\
&S_2 \to \wedge S \mid \wedge R_0 \mid \wedge R_1 \\
&S_3 \to \vee S \mid \vee R_0 \mid \vee R_1 \\
&Z \to S_2Z \mid S_3Z \mid S_2 \mid S_3
\end{aligned}
$$

Schritt Z:

$$
\begin{aligned}
&S \to (S_1 \mid 0 \mid 1 \mid \neg S \mid \neg R_0 \mid \neg R_1 \mid 0S_2 \mid 1S_2 \mid 0S_3 \mid 1S_3 \\
&\mid (S_1Z \mid 0Z \mid 1Z \mid \neg SZ \mid \neg R_0Z \mid \neg R_1Z \mid 0S_2Z \mid 1S_2Z \mid 0S_3Z \mid 1S_3Z \\
&S_1 \to (S_1R_) \mid 0R_) \mid 1R_) \mid \neg SR_) \mid \neg R_0R_) \mid \neg R_1R_) \mid 0S_2R_) \mid 1S_2R_) \mid 0S_3R_) \mid 1S_3R_) \\
&\mid (S_1ZR_) \mid 0ZR_) \mid 1ZR_) \mid \neg SZR_) \mid \neg R_0ZR_) \mid \neg R_1ZR_) \mid 0S_2ZR_) \mid 1S_2ZR_) \mid 0S_3ZR_) \mid 1S_3ZR_) \\
&\mid 0R_) \mid 1R_) \\
&S_2 \to \wedge S \mid \wedge R_0 \mid \wedge R_1 \\
&S_3 \to \vee S \mid \vee R_0 \mid \vee R_1 \\
&Z \to \wedge SZ \mid \wedge R_0Z \mid \wedge R_1Z \mid \vee SZ \mid \vee R_0Z \mid \vee R_1Z \mid \wedge S \mid \wedge R_0 \mid \wedge R_1 \mid \vee S \mid \vee R_0 \mid \vee R_1
\end{aligned}
$$

Nun ist die Grammatik in Greibach-Normalform.

Aufgabe H21 (10 Punkte)
Betrachten Sie folgende Aufgabe.

Aufgabe I1

Beweisen oder widerlegen sie die folgenden Aussagen.

a) Die Sprache $L_1 = \{ a^i b^j \mid i + j \leq 200, i + j \geq 100 \}$ ist nicht regulär.

b) Die Sprache $L_2 = \{(abc)^n \mid n \in \mathbf{N}\}$ ist regulär.

c) Die Sprache $L_3 = \{a^n b^n a^n b^n \mid n \geq 0\}$, mit $\Sigma = \{a, b\}$ ist kontextfrei.

Die drei Studierenden Henri Lotse, Kim Hartfrau und David Rock geben zusammen ihre Hausaufgaben in FoSAP ab, und haben eine Lösung zu Aufgabe I1 eingereicht.
Sie sind nun ein FoSAP-Tutor und müssen die Abgabe der drei korrigieren. Wie viele Punkte würden Sie vergeben? Seien Sie dabei genauso streng wie Ihre eigene Tutorin.

a)

L_1 nicht regulär: Sei $15 \leq n \leq 100$ Sei $w = a^n b^n$. $w \in L$. Wir betrachten eine Zerlegung $w = xyz$ mit $|xy| \leq n$ und $|y| > 0$. Wegen Pumping–Lemma muss auch $xy^i z \in L$ für alle i. Das Wort $xy^{200}z$ hat Länge ungefähr 200 und somit zu viele a und b. So gilt gilt $xy^{200}z \notin L$. Also L nicht regulär und Aussage wahr.

b)

Sei n eine große und schöne Zahl und $w = (abc)^n$ ein Wort. Mit Leichtigkeit kann man herausfinden, dass dieses Wort in der Sprache enthalten ist und länger als n ist. Wir wählen die Zerlegung so dass x und z das leere Wort sind und y w ist. Damit gelten Bedingungen (1) und (2) des Pumpinglemmas. Betrachten wir das gepumpte Wort $xy^i z$ für jedes natürliche i. Dies gleicht $((abc)^n)^i = (abc)^{in}$. Damit ist es in der Sprache enthalten und das Pumpinglemma gilt für L. Also muss L regulär sein.

c)

$z = a^n\, b^n a^n b^n$, dann gibt's Zerlegung z mit $uvwxy$ mit $|vwx| \leq n$ und $|\, vx| > 0$. Sei jetzt $uvwxy$ so das der vwx-Part im ersten viertel liegt, also nur die ersten as enthält. Nach dem Pump-Lemma muss dann auch $uv^i wx^i y \in L$ für alle i kann aber trivialistischlischterweise nicht sein da die Anzahl der a am Anfang sich ändert und am Ende des Wortes aber gleich blieb. $\Rightarrow L_3$ nicht konteckstfrei und die Ausage widerlegt.

Lösungsvorschlag

a) Die Sprache ist regulär, da sie endlich ist: L_1 enthält Wörter der Länge höchstens 200.

Der Beweis ist falsch, da hier n konkret gewählt wird. Wenn wir mit dem Pumping Lemma Regularität widerlegen, so müssen wir zeigen, dass es für jedes n ein Wort mit Länge größer oder gleich n in der Sprache gibt, welches der Spezifikation aus dem Pumping Lemma widerspricht. Für $n > 200$ findet man hier aber kein Wort mehr, da wie schon erwähnt die Sprache nur Wörter der Länge höchstens 200 enthält. Somit ist für $n = 201$ das Pumping Lemma trivialerweise erfüllt.

b) Der erste grundlegende Fehler ist, dass man mit Hilfe des Pumpinglemmas nicht herausfinden kann, dass eine Sprache regulär ist. Damit kann mur nur zeigen, dass eine Sprache nicht regulär ist. Um zu zeigen, dass eine Sprache das Pumpinglemma erfüllt, muss man *für jedes Wort in L* zeigen, dass es eine Zerteilung gibt, sodass die Bedingungen aus dem Lemma erfüllt sind. Hier wird es sehr seltsam gemacht, es scheint ein Quantor zu fehlen.

c) Hier wurde eine feste Zerlegung von vom Wort z angenommen. Um aber mit dem Pumping-Lemma zu zeigen, dass L nicht kontextfrei ist, müssen alle Zerlegung $uvwxy$, welche $|vwx| \leq n$ und $|vx| > 0$ erfüllen, betrachtet werden. Außerdem sollte konkret für ein i gezeigt werden, dass schließlich das Wort $uv^i wx^i y$ nicht in L_3 liegt.