Lösungsheft

Lukas Karras, Benedict Lelanz

19. November 2022

Inhaltsverzeichnis

1 Einleitung

Computerübung 1.1

Nachdem Sie FLACI geöffnet und auf das Modul *Compiler und Interpreter* geklickt haben, können Sie beginnen. Klicken Sie auf das + unten rechts und geben Sie dem zu erstellenden Diagramm einen Namen. Per Klick auf *Speichern* öffnet sich die Arbeitsoberfläche. Ziehen Sie die beiden Programmbausteine aus der Palette links in das leere Feld und definieren Sie für jeden dieser Bausteine auf der linken Seite Name und Quellsprache. Weiterhin wird ein Eingabebaustein benötigt, der entsprechend platziert wird, und ein Ausgabebaustein, der ebenso an den Programmbaustein `QZ2` gezogen wird.

Möchten Sie einen Schritt weiter gehen und die Ausgabe 144 erzeugen, müssen Sie das Quadratzahlenprogramm in der Quellsprache JavaScript implementieren. Drücken Sie dazu auf den Programmbaustein zwischen Ein- und Ausgabe und anschließend auf *Bearbeiten*. Verwenden Sie folgenden Code. (`input` ist dabei eine vom System bereitgestellte Variable)

```
return input*input;
```

Füllen Sie den Eingabebaustein mit 12, markieren Sie den Programmbaustein erneut und drücken Sie auf *Ausführen*.

Computerübung 1.2

Haben Sie den QR-Code gescannt, sollte Ihnen Ihr Handy folgendes in der Sprache ML geschriebenes Lied anzeigen:

*G*0-4 *E*0-4 *E*0-2 *F*0-4 *D*0-4 *D*0-2
*C*0-4 *D*0-4 *E*0-4 *F*0-4 *G*0-4 *G*0-4 *G*0-2
*G*0-4 *E*0-4 *E*0-2 *F*0-4 *D*0-4 *D*0-2
*C*0-4 *E*0-4 *G*0-4 *G*0-4 *C*0-2

Computerübung 1.3

Als Lösung ergibt sich folgendes Modell:

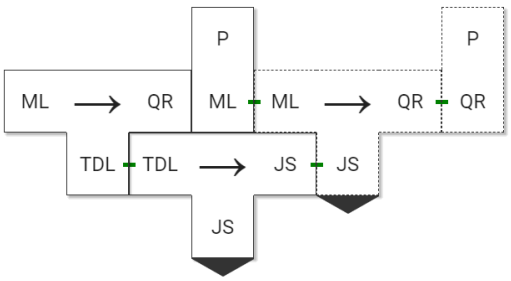


Abb. 1: ML->QR-Compiler

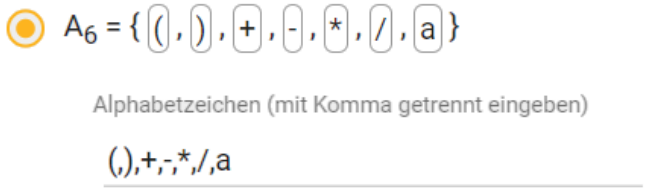
Der Song in der Sprache ML

*G*0-4 *E*0-4 *E*0-2 *F*0-4 *D*0-4 *D*0-2
*C*0-4 *D*0-4 *E*0-4 *F*0-4 *G*0-4 *G*0-4
*G*0-2 *G*0-4 *E*0-4 *E*0-2 *F*0-4 *D*0-4
*D*0-2 *C*0-4 *E*0-4 *G*0-4 *G*0-4 *C*0-2

wird als Programm-Quelltext hinterlegt, compiliert, und das Ergebnis (der QR-Code) mit dem Handy eingescannt. Jetzt werden Sie den eingegebenen Song wiedererkennen!

Computerübung 1.4

Zunächst wird A_6 gewählt, um ein eigenes Alphabet zu definieren. Geben Sie anschließend die einzelnen Zeichen des Alphabetes, getrennt durch ein Komma, ein.



Computerübung 1.5

Bei dieser Aufgabe sollten Sie sich selbst ein Alphabet ausdenken. Für eine Beispiellösung ist folgendes Alphabet gegeben: A={a,b,c}. Nun sollten Sie sich Beispielwörter über dem Alphabet ausdenken und deren Wortlänge bestimmen.

Beispiel:

Wort	Wortlänge
abaabcbccb	10
aaab	4
cbbcbc	6

Sie können Ihre Wörter über Ihrem Alphabet auch in FLACI eingeben und Ihre Ergebnisse überprüfen.

Übung 1.1

a) Beschreiben Sie das Wort u mit der Zeichenfolge $x_1x_2...x_i$ und v mit $y_1y_2...y_k$. Es ist offensichtlich, dass $|u| = i$ und $|v| = k$. Die Verkettung erzeugt ein Wort, das *alle* Zeichen des Wortes u und darauf folgend *alle* Zeichen des Wortes v in der gegebenen Reihenfolge enthält. Daraus ergibt sich, dass $|u \circ v| = i + k = |u| + |v|$.
Aufgrund der Abgeschlossenheit ($u \circ v \in A^*$) lässt sich obiger Schritt beliebig oft wiederholen. Folglich gilt $|u \circ u \circ ... \circ u| = n \cdot |u|$. $|u^n|$ ist lediglich eine andere Schreibweise für den linken Teil letzterer Gleichung.

b)
- Abgeschlossenheit: Das Verknüpfungsgebilde $(A^*, \circ)$ ist ein Monoid. Ein Monoid ist eine Halbgruppe mit neutralem Element. Eine Halbgruppe ist eine Menge (A^*) mit assoziativer Verknüpfung $\circ : A^* \times A^* \to A^*$. Aufgrund dieser Eigenschaft ist A^* abgeschlossen bezüglich $\circ$.
- Assoziativität: $\forall x, y, z \in A^* : x \circ (y \circ z) = (x \circ y) \circ z$. D.h. Klammern können beliebig gesetzt oder entfernt werden. Sei $x = x_1x_2...x_u$, $y = y_1y_2...y_v$ und $z = z_1z_2...z_w$, dann gilt: $x_1x_2...x_u \circ (y_1y_2...y_v \circ z_1z_2...z_w) = x_1x_2...x_u \circ y_1y_2...y_vz_1z_2...z_w = x_1x_2...x_uy_1y_2...y_vz_1z_2...z_w = x_1x_2...x_uy_1y_2...y_v \circ z_1z_2...z_w = (x_1x_2...x_u \circ y_1y_2...y_v) \circ z_1z_2...z_w$
- Kommutativität: Sei $x = a_1a_2...a_i$ und $y = b_1b_2...b_k$, mit $i, k \geq 1$, dann ist $x \circ y = a_1a_2...a_ib_1b_2...b_k \neq b_1b_2...b_ka_1a_2...a_i = y \circ x$. Die Kommutativität gilt also nicht.
- Neutralelement: Die nicht vorhandene Kommutativität haben wir an dem Fall $i, k \geq 1$ festgemacht. Anderenfalls gilt $\varepsilon \circ x = x \circ \varepsilon = x$, wobei ε das Neutralelement ist.

c) $A = \{(,), a, +, -, *, /\}$

$\in A^*$	$\notin A^*$
))((	abc
-(a+a/(a+a))	123
/*/*	
(a*(a+a))	

d) Es sei $A = \{a_1, a_2...a_n\}$. Die Menge A^i enthält n^i Wörter mit genau i Zeichen.

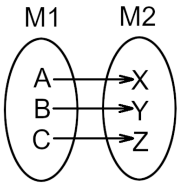
	Wörter	Länge	Anzahl
$A^* = \{$	ε,	0	1
	$a_1, a_2...a_n$,	1	n
	$a_1a_1, a_1a_2...a_na_{n-1}, a_na_n$,	2	n^2
	$a_1a_1a_1, a_1a_1a_2...a_na_na_{n-1}, a_na_na_n$,	3	n^3
	...$\}$		

Die Lösung besteht also darin, die Summe $|A^0| + |A^1| + ... + |A^k|$ zu berechnen (siehe Bsp. 1.6 (S. 15)). Etwas kompakter kann man mithilfe der endlichen geometrischen Reihe schreiben:

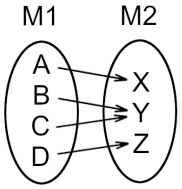
$$\sum_{i=0}^{k} n^i = \frac{n^{k+1} - 1}{n - 1}$$

Übung 1.2

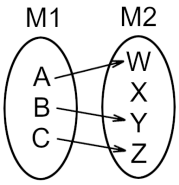
$f : M1 \rightarrow M2$



f ist bijektiv
(und surjektiv, injektiv)



f ist surjektiv,
aber nicht injektiv



f ist injektiv,
aber nicht surjektiv

Computerübung 1.6

Zuerst wählen Sie A_6 aus, um ein eigenes Alphabet zu bestimmen.
Für diese Aufgabe ist A_6 = {a,b,c,d}.
Anschließend klicken Sie, unter der Überschrift "Formale Sprachen", auf die Schaltfläche *Sprache*. Hier werden Ihnen nun alle Wörter über dem Alphabet A_6 angezeigt (A^*). Um nun eine Sprache L (mit $L \subset A^*$) zu erstellen, wählen Sie entweder Wörter aus A^* durch Klicken aus, oder Sie können eines der Auswahlkriterien (L_1 - L_4) auswählen.

Übung 1.3

L_1 beschreibt die Sprache der natürlichen Zahlen ohne Vornullen.

Beispiel:

Wort	1	3054	0976	0
gehört zu L_1?	ja	ja	nein	ja

Übung 1.4

Eine mögliche Notation lautet:

$$L = \{w | w \in \{a\}^* \wedge |w| \in \mathbb{P}\}$$

$\mathbb{P}$ ist dabei die Menge der Primzahlen.

Übung 1.5

Nutzen Sie für das Nimm-Spiel folgenden Link: https://flaci.com/nimspiel.html

Computerübung 1.7

Wenn Sie im Abschnitt *Kontextfreie Grammatiken* angekommen sind, finden Sie in der oberen rechten Ecke, links von Ihrem Benutzernamen, die Schaltfläche *Beispielsammlungen*. Klicken Sie diese an und navigieren Sie zu *Grammatiken* → *MiniJavaScript*. Sowohl der zu prüfende Ausdruck `if (1>B) {while (7<C) {B=C}}; write(-5)` als auch der Programmteil aus https://flaci.com/nimspiel.html werden akzeptiert!

Computerübung 1.8

Die Sprache der arithmetischen Ausdrücke sieht in FLACI folgendermaßen aus:

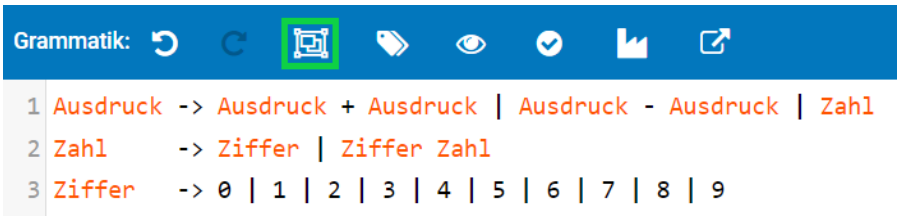


Abb. 2: Sprache der arithmetischen Ausdrücke

Klicken Sie auf die grün markierte Schaltfläche, um zwischen den Modi Syntax-Diagramm und Textmodus zu wechseln.

2 Formale Grammatiken

Computerübung 2.1

Siehe Lehrbuch, S. 34, Abb. 2.3.

Computerübung 2.2

Öffnen Sie in FLACI unter *Meine Grammatiken* die Lösung von Computerübung 2.1 und verwenden Sie das Eingabefeld hinter *Ableiten*, um für das Wort `371` einen Ableitungsbaum zu erzeugen.

Computerübung 2.3

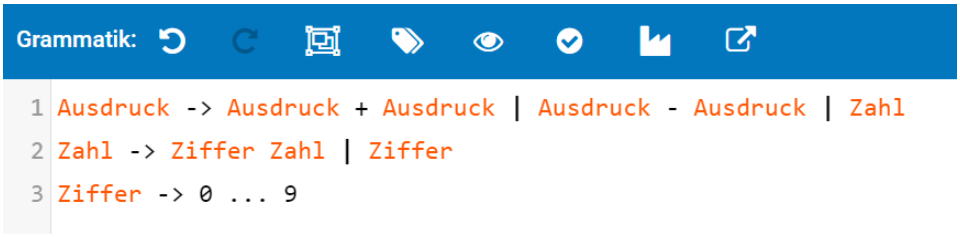


Abb. 3: Grammatik für einfache arithmetische Ausdrücke

Geben Sie das Wort `3-4+5` ein und erzeugen Sie die Ableitungsbäume. Wählen Sie zwischen ihnen, indem Sie das Dropdown-Menü oberhalb des Baumes nutzen.

Computerübung 2.4

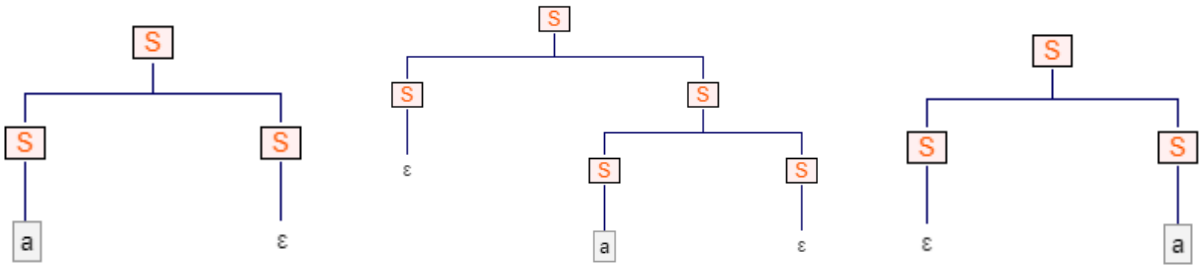
Wenn Sie den Einzelschrittmodus benutzen und auf Rechtsableitung umgeschaltet haben, sollten Sie feststellen, dass das am weitesten rechts stehende Nichtterminal zuerst durch eine rechte Regelseite ersetzt wird. Das Wort `371` entsteht also von rechts her.

Computerübung 2.5

Drei weitere Ableitungen lauten:

- $S \Rightarrow SS \Rightarrow aS \Rightarrow a$ (Linksableitung)
- $S \Rightarrow SS \Rightarrow SSS \Rightarrow SaS \Rightarrow aS \Rightarrow a$ (Unbestimmte Ableitung)
- $S \Rightarrow SS \Rightarrow Sa \Rightarrow a$ (Rechtsableitung)

Da für das Wort `a` unendlich viele Bäume existieren, bietet FLACI nur einen Baum. Die folgenden Bäume sind somit über andere Wege entstanden. Sie repräsentieren die Ableitungen in der gegebenen Reihenfolge.



Computerübung 2.6

Machen Sie FLACI mit der Grammatik bekannt und leiten Sie das Wort `12345` ab. Sie sollten feststellen, dass Ihnen FLACI genau 14 Bäume bietet.

Computerübung 2.7

Zu jedem Wort w, mit $|w| \geq 3$, gibt es mehrere Ableitungsbäume. Zum Beispiel hat ein Wort der Länge 4 stets 5 verschiedene Bäume. Dabei ist es irrelevant, wie viele `a` oder `b` vorkommen. Die Grammatik ist also mehrdeutig.

Computerübung 2.8

Die Frage, zu welchem `if` das `else` gehört, lässt sich aufgrund der beiden möglichen *Entstehungsgeschichten* nicht beantworten.

Für den Nutzer eines `if-then-else`-Blocks ist es beispielsweise übersichtlich, wenn das Zusammengehören von Code-Fragmenten durch Klammerung gekennzeichnet ist. Ergänzen Sie dazu in der Grammatik die beiden Terminale `(` und `)` und nutzen Sie diese, um das *statement* hinter jedem `then` klar von dem Rest einer rechten Regelseite abzugrenzen.

Grammatik: Prüfen Transformieren

```
1 statement -> if expression then ( statement ) |
2 if expression then ( statement ) else statement |
3 S1 | S2
4 expression -> A1 | A2
```

Abb. 4: Eindeutige Grammatik eines `if-then-else`-Blocks

Computerübung 2.9

Das Wort `a+a*(a+a)` gehört der durch G, mit $G = (\{E,T,F\},\{(,),a,+,*\},P,E)$ und $P = \{E \to T \mid E+T, T \to F \mid T*F, F \to a \mid (E)\}$, definierten Sprache an.

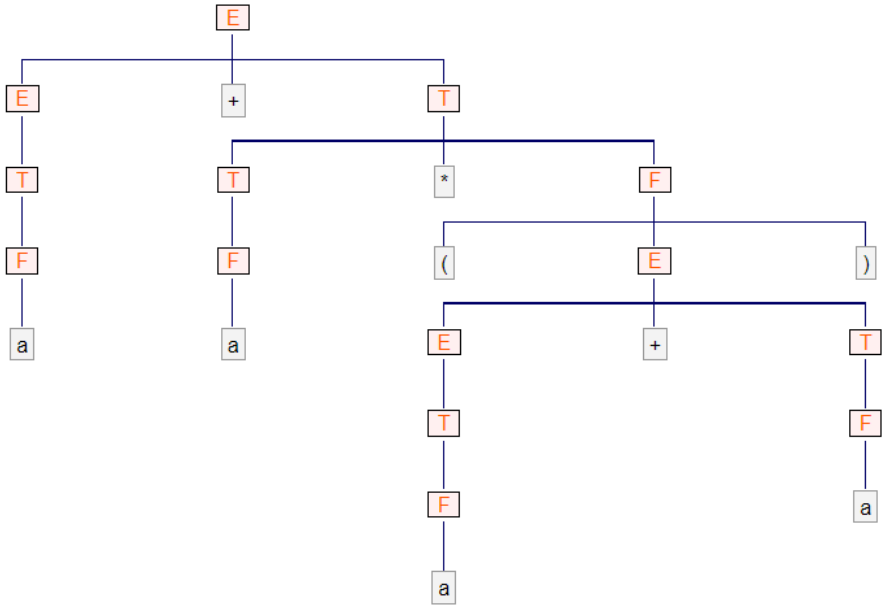


Abb. 5: Ableitungsbaum von `a+a*(a+a)`

FLACI besitzt keine Möglichkeit, auf Mehrdeutigkeit zu prüfen. Dennoch lässt sich anhand der Beschreibung des Ableitungsprozesses leicht feststellen, dass es sich um eine eindeutige Grammatik handelt:

1. Erzeuge die Satzform `T+T+...+T` **ausschließlich** per *Linksrekursion*
2. Ersetze jedes `T` durch `F` oder auf die gleiche Art und Weise durch die Satzform `F*F*...*F`
3. Weitere Substitution von `F` ist trivial

Durch den FLACI-Zauberstab generierte Wörter könnten `(a)+(a)`, `a*(a)+a` oder `a*(((a))*a*a)+a` sein.

Übung 2.1

Eine endliche Sprache ist stets durch eine reguläre Grammatik beschreibbar. Eine reguläre Grammatik kann hingegen wesentlich mehr als nur endliche Sprachen beschreiben: nämlich unendliche. Daher liegt die Menge der endlichen Sprachen innerhalb der Menge der CHOMSKY-Typ-3-Sprachen.

Computerübung 2.10

Um zu einer Sprache L das leere Wort ε hinzuzufügen, ist es meist nicht zielführend, lediglich die Regel $s \to \varepsilon$ dem Spitzensymbol hinzuzufügen, da andere Regeln wieder auf s ableiten

könnten. Die akzeptierte Wortmenge würde sich somit von L und L_ε unterscheiden.
Definieren Sie daher ein neues Nichtterminal s', welches zum Spitzensymbol wird. Anschließend fügen Sie folgende Regel hinzu: $s' \rightarrow s \mid \varepsilon$

Computerübung 2.11

Geben Sie G_1 in FLACI (kontextfreie Grammatiken) ein. Anschließend klicken Sie auf *Transformieren* und in dem Dropdown-Menü wählen Sie *ε-Regeln entfernen*. FLACI formt G_1 nun in eine ε-freie Grammatik um.

Übung 2.2

$G' = (N', T', P', s')$, mit

$$
\begin{aligned}
N' &= \{S, E, T, Zahl, Ziffer\},\\
T' &= \{+,(,),0,1,2,3,4,5,6,7,8,9\},\\
P' &= \{S \rightarrow (\ E\ ) \mid Zahl,\\
&\quad\ E \rightarrow S\ T \mid S,\\
&\quad\ T \rightarrow +\ S,\\
&\quad\ Zahl \rightarrow Ziffer\ WeitereZiffern \mid Ziffer,\\
&\quad\ WeitereZiffern \rightarrow Ziffer\ WeitereZiffern \mid Ziffer,\\
&\quad\ Ziffer \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9\ \},\\
s' &= S
\end{aligned}
$$

Computerübung 2.12

$G'' = (N'', T'', P'', s'')$, mit

$$
\begin{aligned}
N'' &= \{S_0, S, E, T, Zahl, Ziffer\},\\
T'' &= \{+,(,),0,1,2,3,4,5,6,7,8,9\},\\
P'' &= \{S_0 \rightarrow S \mid \varepsilon,\\
&\quad\ S \rightarrow (\ E\ ) \mid Zahl,\\
&\quad\ E \rightarrow S\ T \mid S,\\
&\quad\ T \rightarrow +\ S,\\
&\quad\ Zahl \rightarrow Ziffer\ WeitereZiffern \mid Ziffer,\\
&\quad\ WeitereZiffern \rightarrow Ziffer\ WeitereZiffern \mid Ziffer,\\
&\quad\ Ziffern \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9\ \},\\
s'' &= S_0
\end{aligned}
$$

Übung 2.3

$G' = (N', T', P', s')$, mit

$N' = \{variable, buchstabe, name, zeichen\}$,

$T' = \{$a,b,c,d,e,A,B,C,D,E,0,1,2,3,4,5,6,7,8,9,!,−,+$\}$,

$P' = \{variable \rightarrow buchstabe\ name \mid buchstabe,$
$name \rightarrow buchstabename \mid zeichenname \mid buchstabe \mid zeichen,$
$buchstabe \rightarrow$ a | b | c | d | e | A | B | C | D | E ,
$zeichen \rightarrow$ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ! | - | + $\}$,

$s' = variable$

Übung 2.4

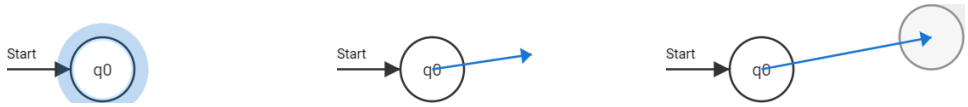
Da die gegebene Sprache Chomsky-Typ-1 ist, nutzen wir folgenden Lösungsansatz:
$S_0 = \{A\}$
$S_1 = \{A, \sout{aABc}, aBc\}$, da $|aABc| = 4 > |w|$
$S_2 = \{A, aBc, \sout{abc}\}$, da $abc \neq w$, und stopp, da $S_1 = S_2$

Das Wort `acb` gehört nicht zu der gegebenen Sprache, da $S_1 = S_2$. Somit können ab S_2 keine neuen Wörter der Länge 3 mehr erzeugt werden und das Wort `acb` kommt nicht in S_2 vor.

3 Endliche Automaten und reguläre Sprachen

Computerübung 3.1

Wählen Sie die FLACI-Komponente *Abstrakte Automaten* und erstellen Sie einen neuen DEA. Beginnen Sie mit der Definition des Alphabetes, indem Sie unter *Eingabealphabet* Σ_5 auswählen und die beiden benötigten Zeichen eintragen.
Das Zustandsmodell bauen Sie nun auf, indem Sie den Cursor auf den Rand eines Zustandes bewegen und mit gedrückter Maustaste ihn von diesem weg ziehen. Drücken Sie auf den so entstandenen Pfeil und wählen Sie das gewünschte Zeichen.



Benutzen Sie den Simulationsmodus, um diverse Wörter zu testen. Sie werden feststellen, dass der DEA den Trap-Zustand nicht mehr verlässt, haben Sie ein Wort eingegeben, das zwar zu Σ^*, aber nicht zu L gehört.
Eine mögliche Modifikation des Automaten bestünde darin, das Akzeptanzverhalten um das leere Wort ε zu erweitern. Markieren Sie q_0 dazu als Endzustand.

Übung 3.1

δ	0	1	2	3	4	5	6	7	8	9
q0	q2	q1	q1	q1	q1	q1	q1	q1	q1	q1
q1	q1	q1	q1	q1	q1	q1	q1	q1	q1	q1
q2	q2	q2	q2	q2	q2	q2	q2	q2	q2	q2

Abb. 6: Überführungsfunktion als Tabelle

Abb. 7: Reguläre Grammatik G'

Computerübung 3.2

Ein Denker sieht seine eigenen Handlungen als Experimente und Fragen - als Versuche, etwas herauszufinden. Erfolg und Misserfolg sind für ihn vor allem Antworten.
Friedrich Nietzsche

Computerübung 3.3

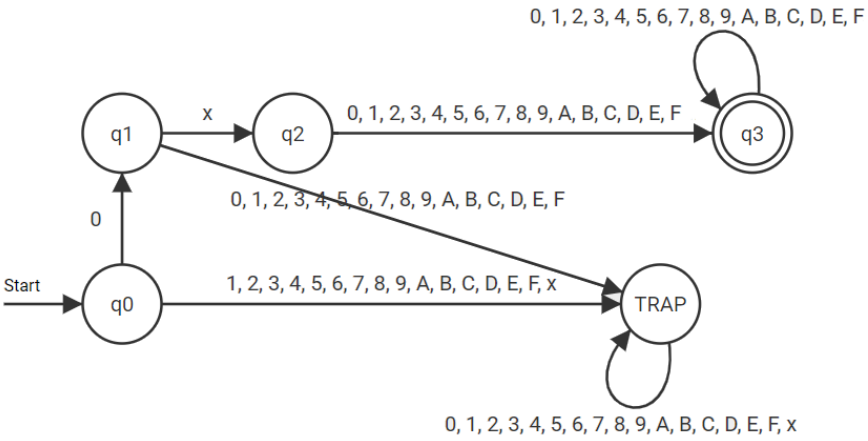


Abb. 8: DEA für die Sprache der Hex-Zahlen

Computerübung 3.4

Wählen Sie dazu *Konvertieren → Automat zu kfG*. Eine reguläre Grammatik entsteht, die unter *Meine Grammatiken* abrufbar ist. Das Resultat sehen Sie in der Lösung zu Übung 3.1.

Computerübung 3.5

Es liegt kein Unterschied vor, da diese Grammatik ohne ε-Regeln nicht weiter vereinfacht werden kann.

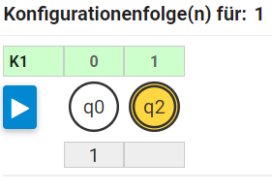
Computerübung 3.6

Das gegebene Verfahren liefert Ihnen zuerst 6 Regeln der Form $q_i \rightarrow xq_j$. Weiterhin benötigen Sie die Regeln der Form $q_i \rightarrow x$, falls q_j ein Endzustand ist. Folglich gilt:

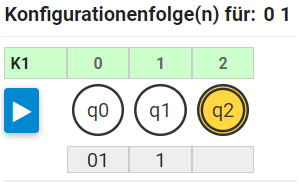
$$P = \{q_0 \rightarrow 0q_1|1q_2, q_1 \rightarrow 0q_1|1q_2, q_2 \rightarrow 0q_1|1q_2\} \cup \{q_0 \rightarrow 1, q_1 \rightarrow 1, q_2 \rightarrow 1\}$$

Haben Sie den Automaten korrekt definiert und die Konvertierung durchgeführt, sollte sich das Ergebnis FLACI's nicht von obiger Regelmenge unterscheiden.

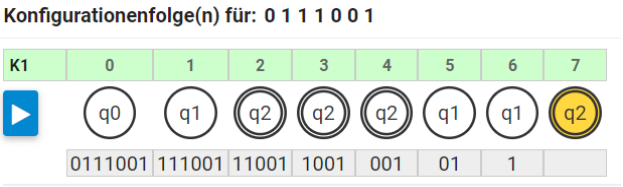
Computerübung 3.7



Satzform	Angewandte Regel
X	X → 1
1	



Satzform	Angewandte Regel
X	X → 0 X
0 X	X → 1
0 1	



Satzform	Angewandte Regel
X	X → 0 X
0 X	X → 1 X
0 1 X	X → 1 X
0 1 1 X	X → 1 X
0 1 1 1 X	X → 0 X
0 1 1 1 0 X	X → 0 X
0 1 1 1 0 0 X	X → 1
0 1 1 1 0 0 1	

Übung 3.2

Nutzen Sie den Hinweis und entwerfen Sie die Übergänge, die durch bbb abgelaufen werden und in den Trap-Zustand führen. Wörter w, mit $w = b^n, n > 3$ werden folglich auch abgelehnt, so wie die Definition wünscht. Nach keinem, einem oder zwei b kommt mindestens ein a. Es ist demnach naheliegend, q_0 als "Loop-Zustand" für a zu verwenden und von den restlichen beiden Nicht-Trap-Zuständen einen a-Übergang zu q_0 zu führen.

Computerübung 3.8

Keine Lösung.

Übung 3.3

Wir betrachten $\hat{\delta}(Q', aw)$. Stellen Sie sich vor, auf jedem einzelnen Zustand der Menge Q' sitzt ein NEA-Clone, der das Wort aw abzuarbeiten hat. Jeder Clone hat sich an die Vorgehensweise $\hat{\delta}(\delta(q,a),w)$ zu halten, wobei q den aktuellen Zustand des jeweiligen Clones beschreibt.
Ist $\delta(q,a)$ definiert, bekommt ein solcher Clone allerdings eine Menge von Zuständen, die er alleine nicht bestreiten kann. Das Cloning wird ausgeführt. Es entsteht der Ausdruck $\hat{\delta}(Q'', w)$, der selbigem Ablauf unterliegt.

Ist $\delta(q,a)$ nicht definiert, crasht der Clone.
Die Rekursion erreicht ihren Elementarfall, wenn das Wort vollständig abgetastet ist.
Aufgrund der Vereinigung, die alle Zustandsmengen, die im Elementarfall erst vollständig aufgebaut sind, zusammenfasst, lässt sich für die Akzeptanz des Wortes w sagen, dass $\hat{\delta}(\{q_0\},w) \cap E \neq \emptyset$ gelten muss. Das bedeutet, dass mindestens ein Clone in einem Endzustand ohne Crash gestoppt hat.

Computerübung 3.9

Das Wort `abb` wird weiterhin ordnungsgemäß abgearbeitet.

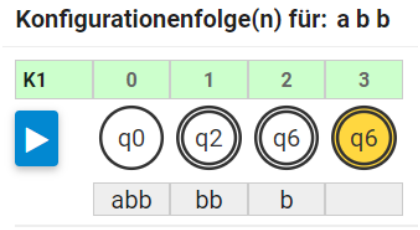
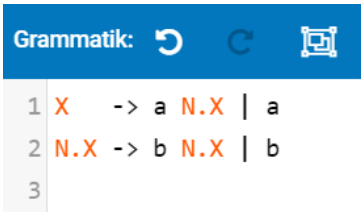


Abb. 9: Konfigurationsreihenfolge für `abb`

Computerübung 3.10

Nach Abarbeitung der ersten zwei Schritte über das Menü *Transformieren* erhalten sie die gewünschte Grammatik. Anschließend können Sie $N.X$ gemäß der Aufgabe in Y umbenennen.

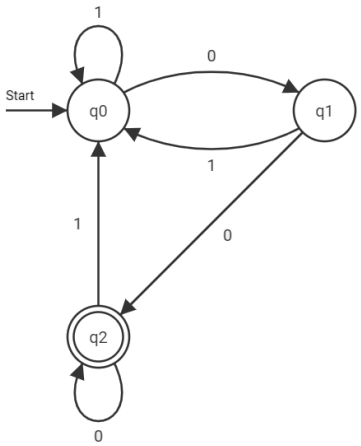


Computerübung 3.11

FLACI liefert Ihnen einen DEA M, der zu dem konstruierten DEA (Lehrbuch, S. 68) isomorph ist. D.h. lediglich die Zustände sind anders positioniert.
Als vereinfachte Grammatik G, mit $L(G) = L(M)$, erhalten Sie durch FLACI:

$$G = (\{q_0,q_1\},\{0,1\},\{q_0 \to 0q_1|1q_0, q_1 \to 0q_1|0|1q_0\},q_0)$$

Auf den ersten Blick mag man meinen, der aus der Grammatik hervorgegangene NEA (Abb. 10) definiere eine andere Sprache. Auf den zweiten Blick sollte zu erkennen sein, dass er sehr wohl die gegebene Sprache beschreibt. Er ist dabei nur einer von *unendlich* vielen verschiedenen für diese Sprache möglichen NEAs.



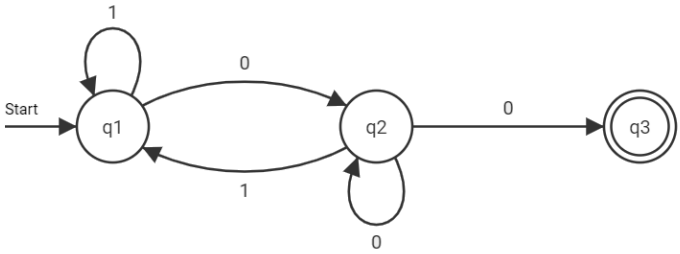


Abb. 10: Äquivalenter NEA

Übung 3.4

a) Für das Wort ab gibt es zwei mögliche Ableitungen: (S, aA, abA, ab) und (S, aB, ab).

b) Nach der Transformation gilt $P' = \{S \to aA|a|aB, A \to bA|b, B \to aB|b\}$.

c) NEA $M = (N \cup \{F\}, T, \delta, S, \{F\})$

δ	a	b
S	{A, F, B}	{}
A	{}	{A, F}
B	{B}	{F}
F	{}	{}

d) Ein Zustandsname der Art ABC im folgenden DEA soll die Zustandsmenge {A, B, C} des NEAs andeuten und Trap die leere Menge.
DEA $M' = (\{S, A, B, F, AFB, AF, Trap\}, T, \delta', S, \{F, AFB, AF\})$

δ'	a	b
S	AFB	Trap
A	Trap	AF
B	B	F
F	Trap	Trap
AFB	B	AF
AF	Trap	AF
Trap	Trap	Trap

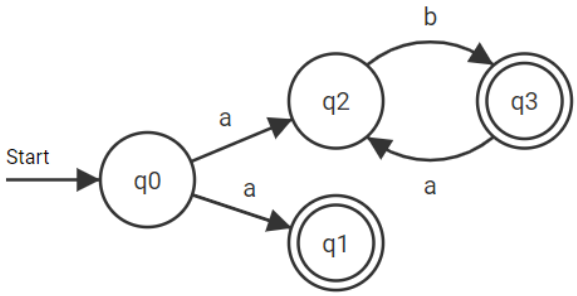
e) Für einen äquivalenten NEA genügt es, die Bestandteile des DEAs zu übernehmen. Die Überführungsfunktion δ'' modifizieren Sie, indem Sie für jedes $\delta'(q_i, x) = q_j$ $\delta''(q_i, x) = \{q_j\}$ hinzufügen.

f) Eine solche Grammatik kann nicht existieren.

Betrachten Sie das "Problemwort" ab - die Regel $S \to aK$ stellt sich als notwendig heraus. Ergänzen Sie nun die Regel $K \to b$ ohne eine weitere rechte Regelseite, steuern Sie aufgrund weiterer geforderter Regeln für Wörter der Form **ab*** geradewegs in die Mehrdeutigkeit. Die Regel $K \to bK|\varepsilon$ wird also G' hinzugefügt. Für Wörter der Form **a+b** benötigen Sie die Regel $S \to aL$, da es keine Option ist, diesen Teil der K-Regel zu überlassen. Für L muss folglich gelten: $L \to aL|\cdots$. Egal, wie Sie den letzten Teil drehen und wenden, wie viel weitere Nichtterminale Sie einbauen - die Mehrdeutigkeit ist unumgänglich.

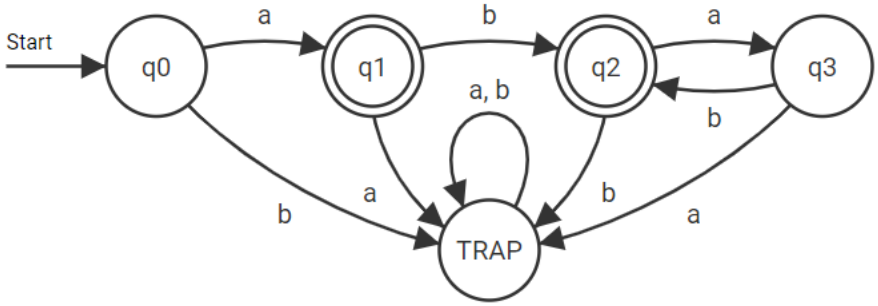
Computerübung 3.12

1. Der Graph sieht folgendermaßen aus:



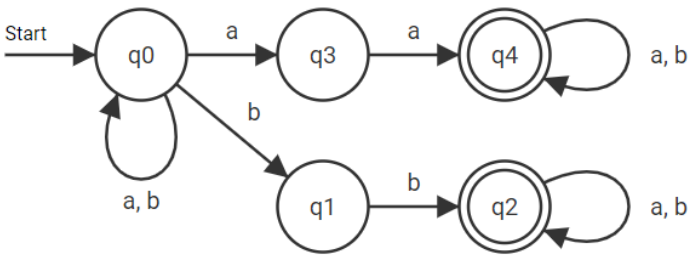
2. $L(M) = \{w \mid w = a \text{ oder } w = [ab]^+\}$

3. Ein äquivalenter DEA:



Computerübung 3.13

a) Der Graph sieht folgendermaßen aus:



b) Das Wort abaab wird vom Automaten akzeptiert, wie die dazugehörige Konfigurationsreihenfolge bestätigt.

0	1	2	3	4	5
q0	q0	q0	q3	q4	q4
abaab	baab	aab	ab	b	

c) $L(M) = \{w | w = \text{X}aa\text{Y}$ oder $w = \text{X}bb\text{Y}$, mit $\text{X} = \{a,b\}^*, \text{Y} = \{a,b\}^*\}$
oder in Umgangssprache:
$L(M)$ beschreibt Wörter, welche mindestens das Teilwort aa oder bb enthalten.

Computerübung 3.14

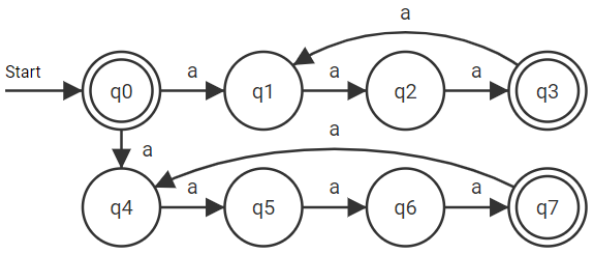


Abb. 11: Akzeptor für Wörter über {a}, deren Länge durch 3 oder 4 teilbar ist

Computerübung 3.15

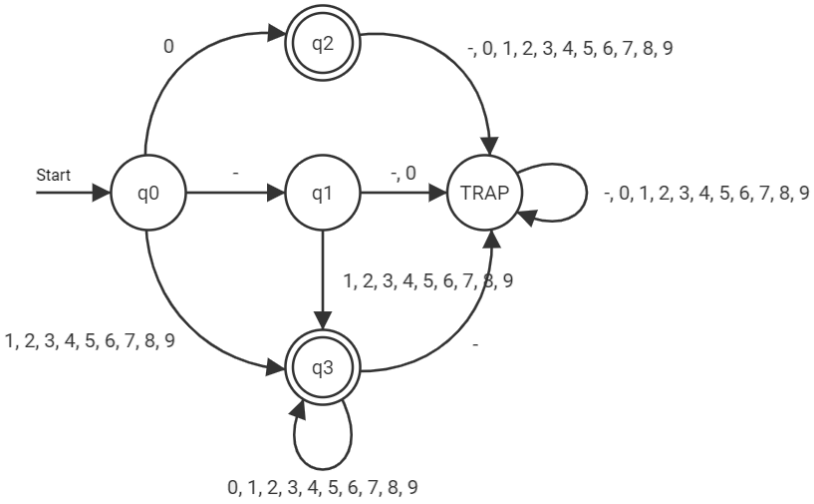


Abb. 12: DEA für die Sprache der ganzen Zahlen

Nach erfolgreicher Transformation sollte FLACI Ihnen einen zu diesem DEA isomorphen DEA liefern.

Computerübung 3.16

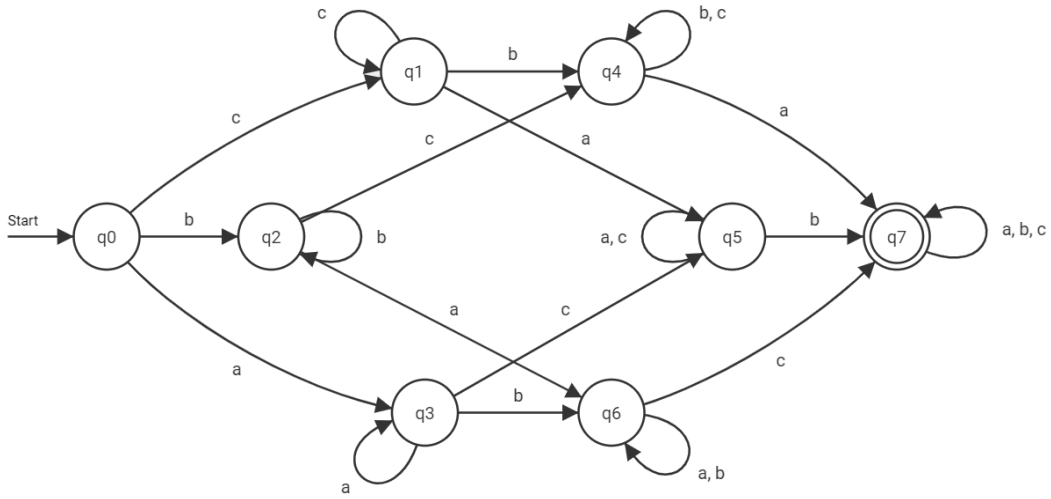


Abb. 13: DEA, der Wörter über {a,b,c} akzeptiert, die jedes Zeichen min. einmal enthalten.

Diese Lösung ist weniger offensichtlich, wurde vor der Definition des Automaten keine formale Beschreibung für die Sprache aufgestellt. Darum soll diese hier folgen, wobei auf eine eher intuitive als kompakte Art gesetzt wird.

$$L = \{w \mid w = n\,\mathbf{a}\,n\,\mathbf{b}\,n\,\mathbf{c}\,n,\ n\,\mathbf{a}\,n\,\mathbf{c}\,n\,\mathbf{b}\,n,\ n\,\mathbf{b}\,n\,\mathbf{a}\,n\,\mathbf{c}\,n,\ n\,\mathbf{b}\,n\,\mathbf{c}\,n\,\mathbf{a}\,n,\ n\,\mathbf{c}\,n\,\mathbf{a}\,n\,\mathbf{b}\,n$$

$$\text{oder } n\,\mathbf{c}\,n\,\mathbf{b}\,n\,\mathbf{a}\,n \mid n \in \{a,b,c\}^*\}$$

Es ergeben sich also 3! mögliche Reihenfolgen von `a,b,c`, die berücksichtigt werden müssen. Der jeweils dazwischenliegende Teil *n* kann beliebig gefüllt werden. Wörter nach dieser Definition gebildet sind jene, die auf das in der Aufgabe vorgegebene Muster passen.
Im obigen DEA sind die Loops *n* reduziert, sodass nur noch absolut notwendige Zeichen beliebig oft wiederholt werden dürfen. Des Weiteren handelt es sich um einen *Minimal-DEA*. D.h. die Zustände wurden auf ein Minimum reduziert.
Bei dem Erstellen des Automaten können Sie ebenso die Möglichkeiten der theoretischen Informatik ausschöpfen und obige Definition auf einen NEA übertragen - durchaus mit ε-Übergängen -, den Sie anschließend in einen DEA konvertieren.

Computerübung 3.17

a) $M = (\{S,R,F\},\{\text{a},?,1\},\delta,S,\{R\})$ mit δ =

δ	a	?	1
S	R	F	F
R	R	F	R
F	F	F	F

Da für jeden Zustand, mit jedem Eingabezeichen, genau eine Definition vorhanden ist, handelt es sich um einen DEA.

b) Die Zustandsfolge der Eingabewörter lautet:

- `aa11a` := $S \to R \to R \to R \to R \to R$
- `aaa1a` := $S \to R \to R \to R \to R \to R$
- `1a1a1a1a` := $S \to F \to F \to F \to F \to F \to F \to F \to F$

Nur wenn der letzte Zustand ein R ist, wird das Wort akzeptiert. Das bedeutet, dass die Wörter `aa11a` und `aaa1a` akzeptiert werden, während `1a1a1a1a` nicht akzeptiert wird.

c) Eine dazugehörige reguläre Grammatik lautet:
$G = (N, T, P, s)$ mit

$$\begin{aligned} N &= \{S, R\}, \\ T &= \{\text{a}, 1\}, \\ P &= \{S \to \text{a}\ R \mid \text{a}, \\ &\quad\ R \to 1\ R \mid 1 \mid \text{a}\ R \mid \text{a}\ \}, \\ s &= S \end{aligned}$$

Da F der TRAP-Zustand ist, entfällt dieser und es wird keine Regel, deren Folgezustand F ist, in die reguläre Grammatik übernommen.

Computerübung 3.18

Probieren Sie selbst.

Computerübung 3.19

Für eine bessere Übersicht ist in Abb. 14 kein TRAP-Zustand vorhanden. Folgender Automat akzeptiert die Fährmannsprache mit einer seekranken Ziege.
Wie sehr leicht zu erkennen ist, gibt es kein Wort, das von dem Automaten akzeptiert wird, da die Ziege bis $q7$ bereits zwei Mal den Fluss überquert hat und somit nicht noch einmal an einer Überfahrt teilnehmen kann.

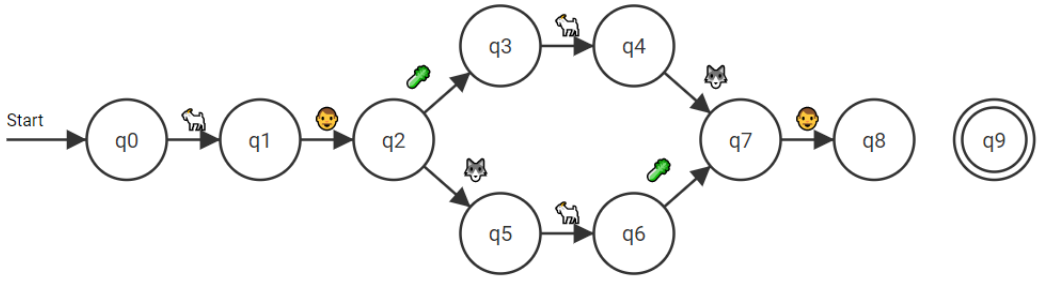


Abb. 14: Fährmannsprache mit seekranker Ziege

Übung 3.5

Zu zeigen: R_L ist rechtsinvariant, d.h. $\forall z \in \Sigma^* : xR_Ly \Rightarrow xzR_Lyz$.

Beweis
Zwei Wörter **x** und **y** stehen in Relation, mit xR_Ly, wenn $\forall z \in \Sigma^* : xz \in L \Leftrightarrow yz \in L$. Zwei Wörter **xk** und **yk** stehen für $\forall k \in \Sigma^*$ ebenso in Relation, mit xkR_Lyk, da sie auf die gleiche Zeichenfolge enden und $kz \in \Sigma^*$.

$$\{(x,y) \mid \forall k \in \Sigma^* : \ \forall z \in \Sigma^* : xkz \in L \Leftrightarrow ykz \in L\} = \{(x,y) \mid \forall z \in \Sigma^* : xz \in L \Leftrightarrow yz \in L\}$$

Übung 3.6

$R_M = \{(x,y) \mid \hat{\delta}(q_0,x) = \hat{\delta}(q_0,y) = q_i, q_i \in Q\}$

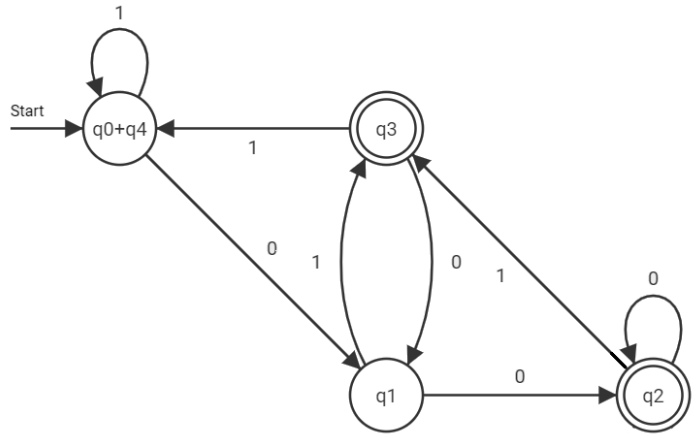
- *Reflexiv* - $\forall x \in \Sigma^* : (x,x) \in R_M$, da $\hat{\delta}(q_0,x) = \hat{\delta}(q_0,x)$.
- *Symmetrisch* - $(x,y) \in R_M \Rightarrow (y,x) \in R_M$, da die Seiten einer Gleichung vertauschbar sind.
- *Transitiv* - $(x,y) \in R_M \wedge (y,z) \in R_M \Rightarrow (x,z) \in R_M$, da x und z durch Wahrheit der Prämisse unweigerlich $\in L$ sind.

Übung 3.7

akzeptiertes Wort	00	01	000	001	100	101	0000	0001
erreichter Zustand	q2	q4	q2	q4	q2	q4	q2	q4

0100	0101	1000	1001	1100	1101	00000
q2	q4	q2	q4	q2	q4	q2

Computerübung 3.20



Es ergeben sich 4 Äquivalenzklassen:

- $[q_0 + q_4] = [11]$
- $[q_1] = [10]$
- $[q_2] = [00]$
- $[q_3] = [01]$

Übung 3.8

Es ergeben sich 3 Äquivalenzklassen. Der Index von R_L ist damit endlich und die Sprache ist regulär.

- $[\varepsilon] = \{\varepsilon, 1, 01, 11, 001...\}$
- $[0] = \{0, 10, 010, 110, 0010...\}$
- $[00] = \{00, 000, 100, 0000, 0100...\}$

Computerübung 3.21

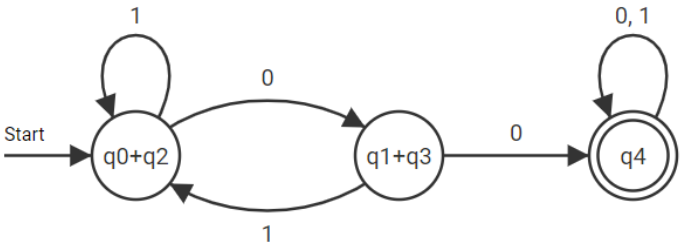
FLACI liefert Ihnen nach Minimierung den gleichen DEA.

Übung 3.9

Um einen DEA minimieren zu können, muss man alle Zustände finden, die zusammengefügt werden können. Hierzu stellt man die rechts stehende Tabelle auf. Wenn es in der Tabelle Felder gibt, die kein x enthalten, so kann man diese beiden Zustände zusammenfügen. In diesem Beispiel können folgende Zustände zu einem zusammengefügt werden:
q0, q2 → q0+q2
q1, q3 → q1+q3.

	q0	q1	q2	q3	q4
q0		x		x	x
q1	!		x		x
q2	!	!		x	x
q3	!	!	!		x
q4	!	!	!	!	

Es ergibt sich folgender Minimal-DEA:



Computerübung 3.22

Keine Lösung.

Übung 3.10

A	B	$\neg A$	$\neg B$	$A \Rightarrow B$	$\neg B \Rightarrow \neg A$
0	0	1	1	1	1
0	1	1	0	1	1
1	0	0	1	0	0
1	1	0	0	1	1

Übung 3.11

Die Sprache $L = \{0^{i^2} | i > 0\}$ beschreibt die Menge der Wörter über dem Alphabet $\{0\}$, deren Länge eine Quadratzahl ist.

Beweis für Nichtregularität
Angenommen, es gibt diese Konstante n, sodass sich jedes Wort w, mit $|w| \geq n, w \in L$, in die Form uvw zerlegen lässt, mit $|uv| \leq n$ und $v \geq 1$, und folglich auch $\forall i \in \mathbb{N}_0 : uv^i w \in L$ gilt.
Wir betrachten das Wort $0^{n^2} = uvw$, mit $u = 0^a$, $v = 0^b$ und $w = 0^c$, dessen Länge offensichtlich $\geq n$ ist, und setzen i auf 2. Es gilt $a + 2b + c \leq n^2 + n$, da $|uv| \leq n$ und $|uvw| = n^2$. Damit kann $|0^a 0^{2b} 0^c|$ keine Quadratzahl sein, da $n^2 + n < (n+1)^2$.

Übung 3.12

- L_1: nicht regulär, da es keine "aufpumpbare" Zerteilung eines Wortes gibt.

  Die Satzform uvw kann wie folgt belegt werden:
  - $u = a^i b^j$, $v = b^k$, $w = b^{2i-k-j}$: aus uv^0w folgt, dass $k = 0$, um ein korrektes Wort zu erhalten. $v = b^0$ verletzt die Forderung $|v| \geq 1$.
  - $u = a^{i-j}$, $v = a^j b^k$, $w = b^{2i-k}$: um fehlerhaften Satzbau für uv^2w zu vermeiden, folgt
    $\begin{cases} j = 0 \Rightarrow k = 0, \text{ wegen } a^i b^{2i+k}. \\ k = 0 \Rightarrow j = 0, \text{ wegen } a^{i+j} b^{2i}. \end{cases}$

    Die Forderung $|v| \geq 1$ kann demnach nicht erfüllt werden.
  - $u = a^{i-j-k}$, $v = a^j$, $w = a^k b^{2i}$: analog zu Fall 1.

- L_2: ist regulär und kann durch folgende reguläre Grammatik beschrieben werden:

  $G = (N, T, P, s)$, mit
  $N = \{S, A, B\}$,
  $T = \{\text{a,b}\}$,
  $P = \{S \rightarrow \text{a } B,$
  $A \rightarrow \text{a } B,$
  $B \rightarrow \text{b } A \mid \text{b}\}$,
  $s = S$

- L_3: ist regulär, da folgende reguläre Grammatik die Sprache beschreibt:
  $G = (N, T, P, s)$, mit

  $$\begin{aligned} N &= \{S, X, Y\}, \\ T &= \{\text{a}\}, \\ P &= \{S \rightarrow \text{a } X, \\ &\quad X \rightarrow \text{a } Y \mid \text{a}, \\ &\quad Y \rightarrow \text{a } X\}, \\ s &= S \end{aligned}$$

- L_4: nicht regulär. Sei $uvw = 0^{2^n}$, mit $u = 0^j$, $v = 0^k$ und $w = 0^l$. Die Forderung $|0^{2^n}| \geq n$ ist erfüllt. Die Satzform uv^2w impliziert $|uv^2w| \leq 2^n + n < 2^{n+1}$, mit $n \geq 1$. (Aufgrund der Forderung $|uv| \leq n$)

- L_5: nicht regulär, da die Satzform uvw wie folgt belegt werden kann:
  - $u = a^n b^m a^j$, $v = a^k$, $w = a^{m+n-k-j}$: analog zu Fall 1 von L_1.
  - $u = a^n b^{m-j}$, $v = b^j a^k$, $w = a^{n+m-k}$: analog zu Fall 2 von L_1.
  - $u = a^n b^{m-j-k}$, $v = b^j$, $w = b^k a^{n+m}$: analog zu Fall 1.
  - $u = a^j$, $v = a^{n-j} b^m a^k$, $w = a^{m+n-k}$: $uv^0w \notin L_5$.
  - $u = a^{n-j}$, $v = a^j b^k$, $w = b^{m-k} a^{n+m}$: analog zu Fall 2.
  - $u = a^j$, $v = a^k$, $w = a^{n-j-k} b^m a^{m+n}$: analog zu Fall 1.

Computerübung 3.23

Vergleich mit Buch.

Computerübung 3.24

Der NEA ohne Epsilon-Übergängen sieht wie folgt aus:

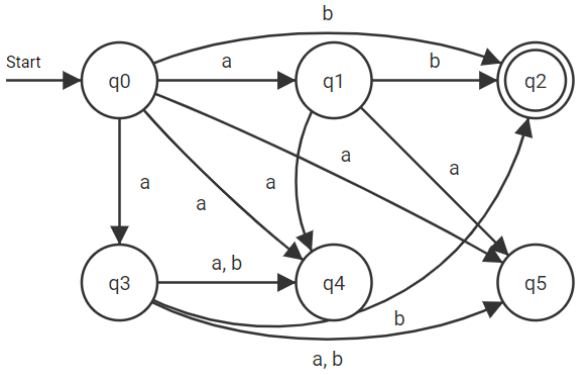


Abb. 15: NEA ohne ε-Übergänge

Dieser kann vereinfacht werden (siehe Abb. 16)

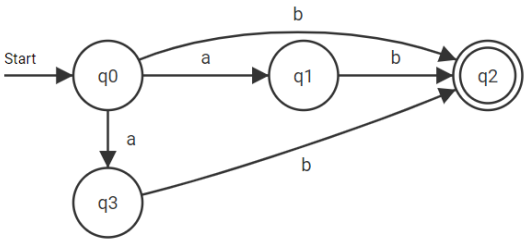


Abb. 16: vereinfachter NEA ohne ε

Anschließend können q1 und q3 zu einem Zustand verschmelzen.

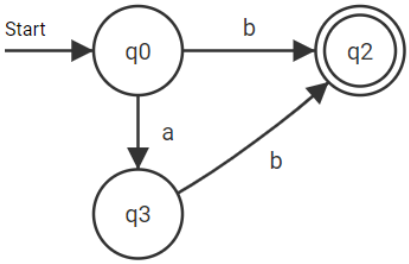


Abb. 17: weiter vereinfachter NEA onhe ε

Um den in Abb. 17 abgebildeten NEA in einen DEA zu konvertieren, muss lediglich der *TRAP*-Zustand hinzugefügt werden. Alle unerwünschten Übergänge führen in den *TRAP*-Zustand (siehe Abb. 18)

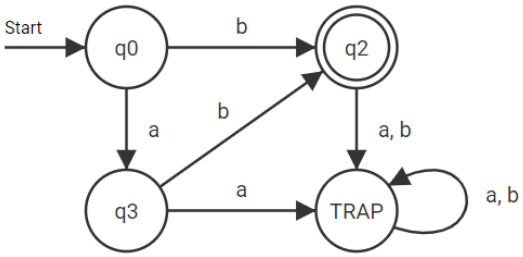


Abb. 18: DEA aus NEA konvertiert

Übung 3.13

Zunächst bildet man die ε-Hülle jedes Zustandes mit jedem Zeichen des Eingabealphabetes. Um dies übersichtlich zu halten, machen wir dies in Tabellenform. Die entstandene Tabelle sieht folgendermaßen aus:

q	δ'	ε-Hülle(q)	d(Spalte 3, x)	ε-Hülle(Spalte 4)
A	(A,a)	{A}	$\emptyset$	$\emptyset$
A	(A,b)	{A}	{S}	{S,H,B}
B	(B,a)	{B}	{S}	{S,H,B}
B	(B,b)	{B}	$\emptyset$	$\emptyset$
H	(H,a)	{H}	$\emptyset$	$\emptyset$
H	(H,b)	{H}	$\emptyset$	$\emptyset$
S	(S,a)	{S,B,H}	{S1,S}	{S1,S,H,B}
S	(S,b)	{S,B,H}	{S2}	{S2}
S1	(S1,a)	{S1}	$\emptyset$	$\emptyset$
S1	(S1,b)	{S1}	{A}	{A}
S2	(S2,a)	{S2}	{B}	{B}
S2	(S2,b)	{S2}	$\emptyset$	$\emptyset$

Aus dieser Tabelle kann man nun einen NEA ohne ε-Übergänge erstellen, indem man aus der 2.Spalte den Zustand mit dem Eingabezeichen zu jedem Zustand aus der Menge in der 5.Spalte verbindet.

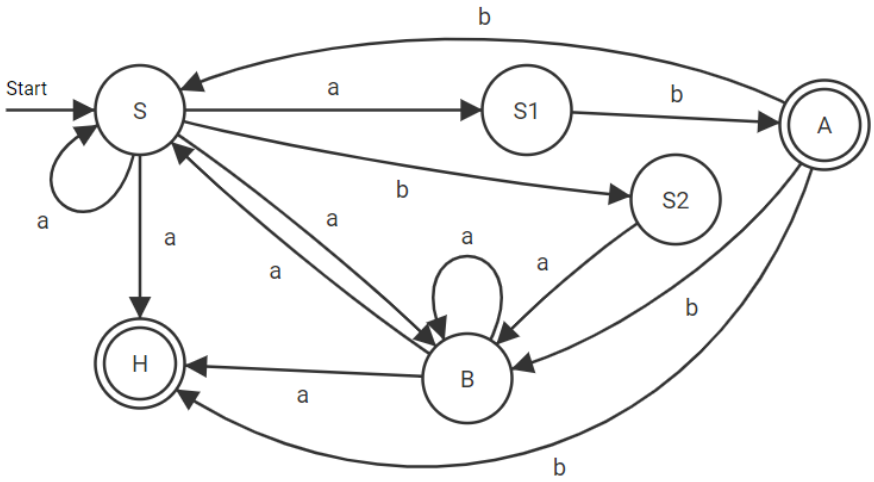


Abb. 19: NEA ohne ε

Computerübung 3.25

Beide Wörter ab und abb werden von dem Automaten akzeptiert.

Computerübung 3.26

δ	0	1
q0	q1	q2
q1	q3	q2
q2	q1	q4
q3	q3	q2
q4	q1	q4

λ	Δ
q0	no
q1	no
q2	no
q3	yes
q4	yes

Computerübung 3.27

Beispiele:

- 50

Zustand	q0	q1
Ausgabe		noch zu zahlen: 1€

- 100

Zustand	q0	q4
Ausgabe		noch zu zahlen: 50 cent

- 100 50

Zustand	q0	q4	q2
Ausgabe		noch zu zahlen: 50 cent	Bitte Kaffee entnehmen.

- 100 100

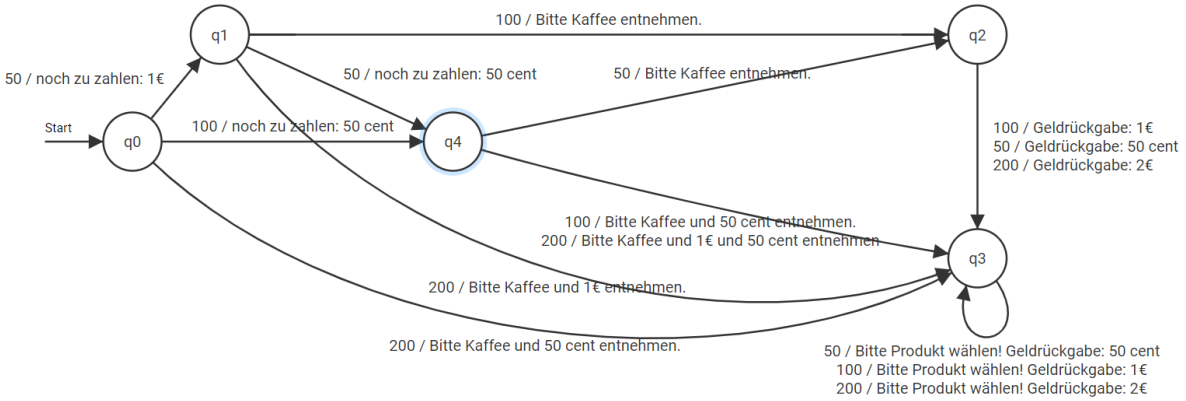
Zustand	q0	q4	q3
Ausgabe		noch zu zahlen: 50 cent	Bitte Kaffe und 50 cent entnehmen.

Übung 3.14

δ	a	b
q0	q1	q2
q1	q2	q0
q2	q2	q1

λ	a	b
q0	1	0
q1	1	0
q2	0	0

Computerübung 3.28



Computerübung 3.29

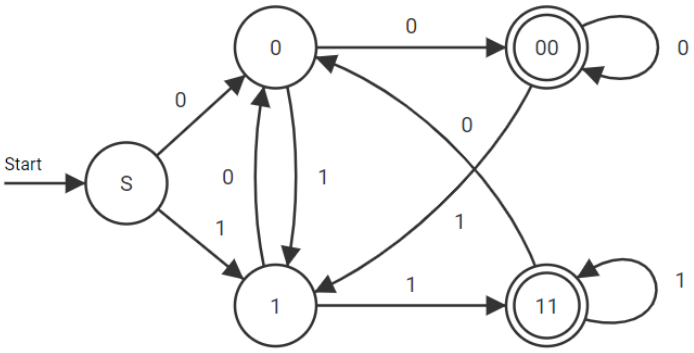


Abb. 20: $L = \{w \mid w = v00 \text{ oder } w = v11, v \in \{0,1\}^*\}$

4 Reguläre Ausdrücke (RA)

Übung 4.1

- $(\mathbf{u}^*)^* = \mathbf{u}^*$

  $\mathbf{u}^*$ ist als Monoid abgeschlossen.

- $\mathbf{u}(\mathbf{v}+\mathbf{w}) = \mathbf{uv}+\mathbf{uw}$

  $\mathbf{u}$ wird entweder mit $\mathbf{v}$ oder $\mathbf{w}$ konkateniert.

- $(\mathbf{uv})^*\mathbf{u} = \mathbf{u}(\mathbf{vu})^*$

  Der Ausdruck $(\mathbf{uv})^*\mathbf{u}$ beginnt stets mit genau einem $\mathbf{u}$ und endet auf beliebig viele $\mathbf{vu}$.

- $(\mathbf{u}+\mathbf{v})^* = (\mathbf{u}^*+\mathbf{v}^*)^*$

  Ein Wort in $(\mathbf{u}+\mathbf{v})^*$ kann hintereinander beliebig viel $\mathbf{u}$ bzw $\mathbf{v}$ enthalten.

Übung 4.2

$L_1L_2 = \{10011, 1011, 111\}$

$\{10, 11\}^* = \{\varepsilon, 10, 11, 1010, 1011, 1110, 1111, 101010, ...\}$

Übung 4.3

$\mathbf{R} = \mathtt{a}(\mathtt{a}+\mathtt{b})^* + (\mathtt{a}+\mathtt{b})^*\mathtt{a} + \mathtt{a}(\mathtt{a}+\mathtt{b})^*\mathtt{a}$

Übung 4.4

Zur Lösung beider Aufgabenteile ist ein Syntax-Diagramm von Vorteil. In diesem kann man die Verbindungen den Wörtern `acabaccc` und `bbaaacc` entsprechend abtasten und feststellen, dass für beide $\in L$ gilt. Ebenso lässt sich die im Lehrbuch gegebene Lösung nachvollziehen, dass keine Zeichenkette der Form `bc` in den Wörtern der Sprache enthalten sein kann.

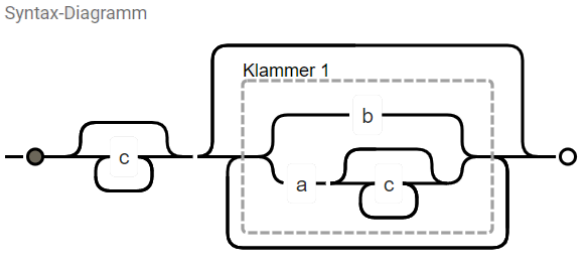


Abb. 21: Syntax-Diagramm für $\mathbf{c}^*(\mathbf{b}+\mathbf{ac}^*)^*$

Übung 4.5

Behauptung
Der reguläre Ausdruck $(\mathbf{1}+\mathbf{10})^*$ enthält keine Wörter mit zwei aufeinanderfolgenden Nullen. D.h. $\forall i \in \mathbb{N}_0 : (\mathbf{1}+\mathbf{10})^i \not\ni w, w$ enthält die Teilzeichenkette 00.

Induktionsanfang
Die Menge der durch den reguläre Ausdruck $(\mathbf{1}+\mathbf{10})^i$ beschriebenen Wörter enthält für $i = 0$ nur ε. Also existiert mindestens ein i, für das die Behauptung gilt.

Induktionsschritt
Gilt die Behauptung für i, dann auch für $i+1$, denn $(\mathbf{1}+\mathbf{10})^{i+1} = (\mathbf{1}+\mathbf{10})^i(\mathbf{1}+\mathbf{10})$. Die Menge $L((\mathbf{1}+\mathbf{10})^i)$ wird also mit $\{1, 10\}$ konkateniert, wodurch keine Wörter mit Teilzeichenkette 00 entstehen können. Damit gilt die Behauptung.

Übung 4.6

Zunächst erfolgt eine Umbenennung der Zustände, sodass aus den Zuständen q_0 und q_1, die Zustände q_1 und q_2 werden.
Anschließend ist $R = R_2(1,2)$ gesucht.

$$
\begin{aligned}
R_2(1,2) &= R_1(1,2) \cup R_1(1,2) \circ R_1(2,2)^* \circ R_1(2,2) \\
R_1(1,2) &= R_0(1,2) \cup R_0(1,1) \circ R_0(1,1)^* \circ R_0(1,2) \\
R_0(1,2) &= \{1\} \\
R_0(1,1) &= \{\varepsilon, 0\} \\
R_1(1,2) &= \{1\} \cup \{\varepsilon, 0\} \circ \{\varepsilon, 0\}^* \circ \{1\} = \{0^*1\} \\
R_1(2,2) &= R_0(2,2) \cup R_0(2,1) \circ R_0(1,1)^* \circ R_0(1,2) \\
R_0(2,2) &= \{\varepsilon, 0\} \\
R_0(2,1) &= \{1\} \\
R_1(2,2) &= \{\varepsilon, 0\} \cup \{1\} \circ \{\varepsilon, 0\}^* \circ \{1\} = \{\varepsilon, 0, 10^*1\} \\
R_2(1,2) &= \{0^*1\} \cup \{0^*1\} \circ \{\varepsilon, 0, 10^*1\}^* \circ \{\varepsilon, 0, 10^*1\} \\
R &= \mathbf{0^*1} + (\mathbf{0^*1}) \cdot (\boldsymbol{\varepsilon} + \mathbf{0} + \mathbf{10^*1})^* \cdot (\boldsymbol{\varepsilon} + \mathbf{0} + \mathbf{10^*1}) \\
&= \mathbf{0^*1} + (\mathbf{0^*1}) \cdot (\boldsymbol{\varepsilon} + \mathbf{0} + \mathbf{10^*1})^* \\
&= (\mathbf{0^*1}) \cdot (\boldsymbol{\varepsilon} + \mathbf{0} + \mathbf{10^*1})^* \\
R &= (\mathbf{0^*1})(\mathbf{0} + \mathbf{10^*1})^*
\end{aligned}
$$

Computerübung 4.1

Keine Lösung.

Computerübung 4.2

Zu zeigen ist, dass $R_1 = (\mathbf{0}+\mathbf{1})(\mathbf{1}+\mathbf{00}+\mathbf{01})^*$ und $R_2 = \mathbf{0}^*\mathbf{1}(\mathbf{0}+\mathbf{10}^*\mathbf{1})^*$ nicht äquivalent sind.

Es gilt, dass $\forall i \in \mathbb{N}_0 : 10^i \in L(R_2)$. Allerdings ist für alle ungeraden i, aufgrund des regulären Teilausdrucks **00** von R_1, das Wort $10^i \notin L(R_1)$.

Computerübung 4.3

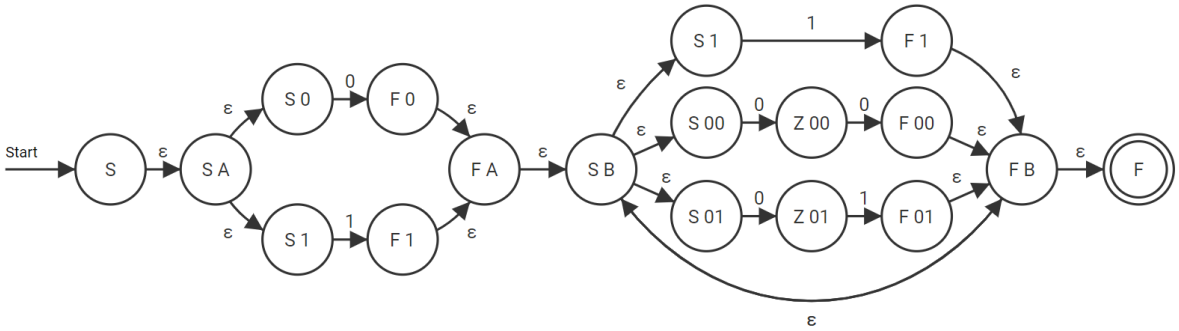


Abb. 22: NEA des RA $(0+1)(1+00+01)^*$

Versucht man nun mit diesem NEA_ε das Wort 10011100 abzuleiten, stellt man fest, dass FLACI die Simulation abbricht, da zu viele Maschinen erzeugt werden. Das bedeutet nicht, dass das Wort nicht von dem Automaten akzeptiert wird, das ist lediglich ein "Schutzmechanismus" von FLACI, um eine mögliche unendliche Rechenoperation abzubrechen.
Um ein eindeutiges Ergebnis zu erhalten, ob das Wort 10011100 zur Sprache des RA gehört, muss man den NEA in einen DEA umwandeln und minimieren. Dieser sieht folgendermaßen aus:

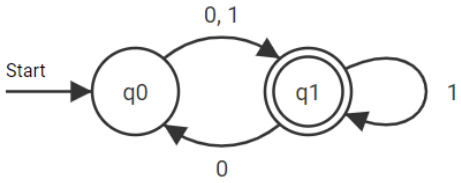


Abb. 23: DEA des RA $(0+1)(1+00+01)^*$

Dieser liefert bei der Simulation des Wortes 10011100 folgende Konfigurationsreihenfolge:
q0 → q1 → q0 → q1 → q1 → q1 → q1 → q0 → q1
Somit wird das Wort 10011100 von dem RA akzeptiert.

Computerübung 4.4

Ein möglicher RA in Praxisnotation für die Sprache der Daten von 2000 bis 2008 ist folgender: (0[1-9]|[1-2][0-9]|3[0-1])\.([0][1-9]|10|11|12)\.(200[0-8]) .

Die Umwandlungen erfolgen durch die eingebauten Funktionen in FLACI.

Übung 4.7

Keine Lösung.

Computerübung 4.5

Zunächst wandelt man den gegebenen RA in die Praxisnotation um: $(0|1)^*(000|111)(0|1)^*$.
Anschließend kann man aus diesem einen NEA erstellen und diesen minimieren.
Der minimale NEA sieht folgendermaßen aus:

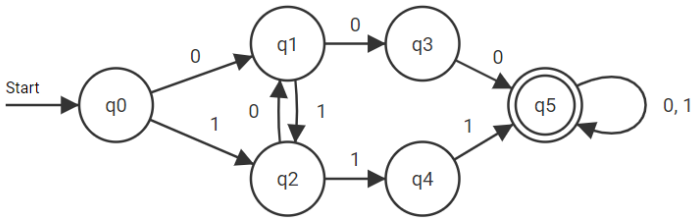


Abb. 24: NEA des RA (0+1)*(000+111)(0+1)*

Computerübung 4.6

Zu zeigen ist die Äquivalenz von $R_1 = (\mathbf{a}(\mathbf{b}+(\mathbf{ab}+\mathbf{ac}))^*)^*\mathbf{a}$ und $R_2 = \mathbf{a}((\mathbf{b}^*+(\mathbf{a}(\mathbf{b}+\mathbf{c}))^*)^*\mathbf{a})^*$.

Sei $\mathbf{u} = (\mathbf{b}+(\mathbf{ab}+\mathbf{ac}))^*$. $\mathbf{u}$ lässt sich umformen zu $(\mathbf{b}^*+(\mathbf{ab}+\mathbf{ac})^*)^*$ und weiter vereinfachen zu $(\mathbf{b}^*+(\mathbf{a}(\mathbf{b}+\mathbf{c}))^*)^*$.

Der Ausdruck $R_1 = (\mathbf{au})^*\mathbf{a}$ ist mit dem Ausdruck $\mathbf{a}(\mathbf{ua})^*$ äquivalent. Nach dem Einsetzen von $\mathbf{u}$ ergibt sich R_2.

Computerübung 4.7

Ein möglicher NEA ist in Abb. 25 zu sehen.

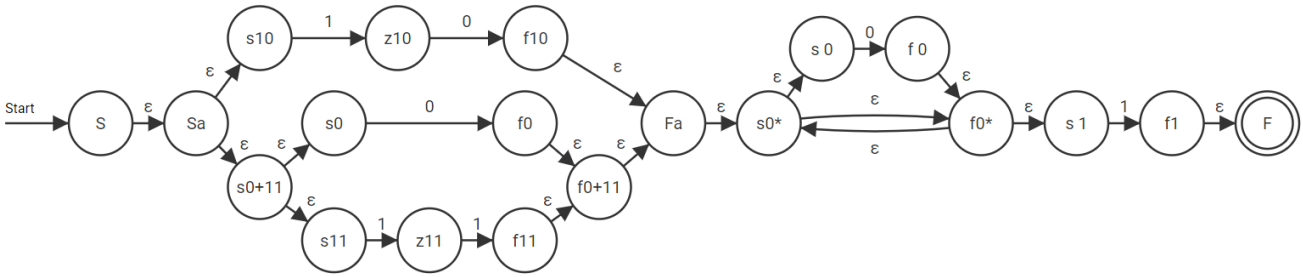
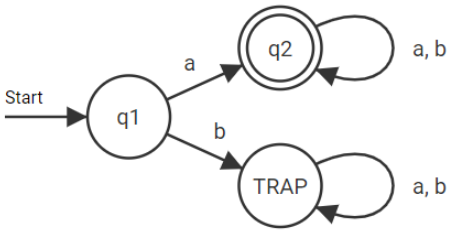


Abb. 25: Möglicher NEA für den RA 10+(0+11)0*1.

Übung 4.8

Zunächst erstellen wir den DEA aus der gegebenen Beschreibung der Sprache.



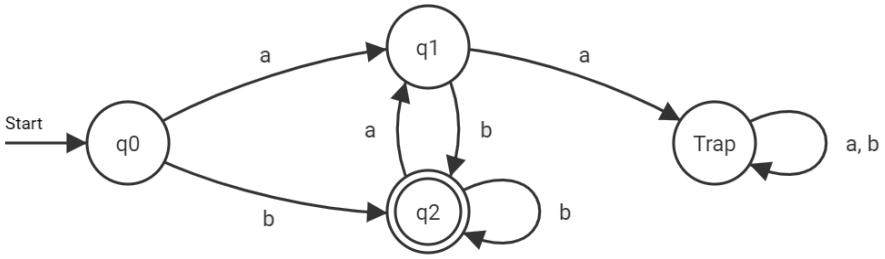
Die einzige Möglichkeit vom dem Startzustand *q1* zu dem Endzustand *q2* zu gelangen ist der direkte Weg von q1 zu q2. Daher wählen wir unser **i = 1** und **j = 2**.
Daher ist $R = R_2(1,2)$ gesucht.

$$
\begin{aligned}
R_2(1,2) &= R_1(1,2) \cup R_1(1,2) \circ R_1(2,2)^* \circ R_1(2,2) \\
R_1(1,2) &= R_0(1,2) \cup R_0(1,1) \circ R_0(1,1)^* \circ R_0(1,2) \\
R_0(1,2) &= \{a\} \\
R_0(1,1) &= \{\varepsilon\} \\
R_1(1,2) &= \{1\} \cup \{\varepsilon\} \circ \{\varepsilon\}^* \circ \{a\} = \{a\} \\
R_1(2,2) &= R_0(2,2) \cup R_0(2,1) \circ R_0(1,1)^* \circ R_0(1,2) \\
R_0(2,2) &= \{a,b\} \\
R_0(2,1) &= \{\varepsilon\} \\
R_1(2,2) &= \{a,b\} \cup \{\varepsilon\} \circ \{\varepsilon\}^* \circ \{a\} = \{a,b\} \\
R_2(1,2) &= \{a\} \cup \{a\} \circ \{a,b\}^* \circ \{a,b\} \\
R &= \mathbf{a} + \mathbf{a} \cdot (\mathbf{a} + \mathbf{b})^* \cdot (\mathbf{a} + \mathbf{b}) \\
&= \mathbf{a} + \mathbf{a} \cdot (\mathbf{a} + \mathbf{b})^* \\
&= \mathbf{a} \cdot (\mathbf{a} + \mathbf{b})^* \\
R &= \mathbf{a}(\mathbf{a} + \mathbf{b})^*
\end{aligned}
$$

Übung 4.9

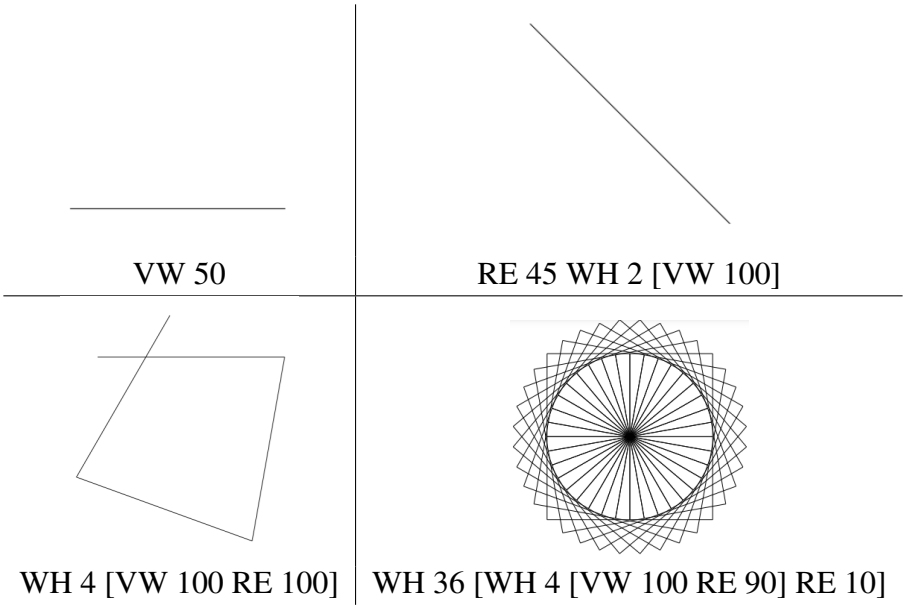
a) Ein RA in Praxis-Syntax lautet `a?(b+a?)*b`.
Beispielwörter sind: `abbabbab`, `bbbbbab` und `abababab`.

b) Der Minimal-DEA für die Sprache L sieht wie folgt aus:



c) $G = (\{A, B\}, \{a, b\}, \{A \rightarrow bA|b|aB, B \rightarrow bA|b\}, A)$

Übung 4.10



Das ZR-Programm RE 270 hinterlässt nichts Sichtbares.

Übung 4.11

Die folgenden ZR-Programme beschreiben die Grafiken in der gegebenen Reihenfolge.

- WH 4 [VW 100 RE 90]
- WH 360 [VW 1 RE 1]
- WH 2 [WH 4 [WH 4 [VW 50 RE 90] RE 270] RE 45]
- WH 8 [WH 360 [VW 1 RE 1] RE 45]

Computerübung 4.8

Deutsche Autokennzeichen bestehen aus einem Unterscheidungszeichen, das mit 1-3 Großbuchstaben (inkl. Ä, Ö, Ü) den Bezirk der Zulassung beschreibt, und einer Erkennungsnummer. Letztere beginnt mit 1-2 Großbuchstaben, gefolgt von 1-4 Ziffern ohne vorangehende 0.

Ein dafür geeigneter RA lautet:
[A-ZÄÖÜ][A-ZÄÖÜ]?[A-ZÄÖÜ]? [A-Z][A-Z]? [1-9][0-9]?[0-9]?[0-9]?

Computerübung 4.9

Zunächst definieren wir die vollständige kfG:

$G = (N, T, P, s)$ mit

N = $\{Eintraege, Eintrag, Zahl, Ziffern, Dezimalzahl, Kna, GBs, KBs, Num, Werte,$
$1-9, 0-9, a-z, A-Z\}$,

T = {:, ;, ., 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z,
A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z},

P = $\{Eintraege \rightarrow Eintrag\ Eintraege \mid Eintrag$
$Eintrag \rightarrow Zahl : Kna ; Werte ;$
$Zahl \rightarrow 1-9\ Ziffern \mid 0$
$Ziffern \rightarrow 0-9\ Ziffern \mid \varepsilon$
$Kna \rightarrow GBs\ KBs\ Num$
$GBs \rightarrow A-Z\ GBs \mid A-Z$
$KBs \rightarrow a-z\ KBs \mid \varepsilon$
$Num \rightarrow Zahl \mid A-Z\ Zahl$
$Werte \rightarrow Dezimalzahl ; Werte \mid Dezimalzahl$
$Dezimalzahl \rightarrow Zahl\ .\ 0-9\ Ziffern \mid$
$1-9 \rightarrow 1|2|3|4|5|6|7|8|9$
$0-9 \rightarrow 0|1|2|3|4|5|6|7|8|9$
$a-z \rightarrow$ a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
$A-Z \rightarrow$ A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z},

s = $Eintraege$

Um diese Grammatik zu vereinfachen, definieren wir uns drei Token.

Zahl	:	(0|[1-9][0-9]*)
Dezimalzahl	:	(0|[1-9][0-9]*).[0-9]+
Bezeichner	:	[A-Z]+[a-z]*[A-Z]?[1-9][0-9]*

Anschließend sieht die Grammatik folgendermaßen aus:

$G' = (N', T', P', s')$ mit
$N' = \{Eintraege, Eintrag, Werte\}$,
$T' = \{:, ;\}$,
$P' = \{Eintraege \rightarrow Eintrag\ Eintraege \mid Eintrag$
$Eintrag \rightarrow$ **Zahl** : **Bezeichner** ; $Werte$;
$Werte \rightarrow$ **Dezimalzahl** | **Dezimalzahl** ; $Werte$
$s' = Eintraege$

Computerübung 4.10

Verwenden Sie den RegExp **Bezeichner** aus der letzten Übungsaufgabe und bedienen Sie sich am Pattern matching. Eine mögliche Java-Prozedur, die dies erledigt, sei hier einmal gegeben.

```
void scan() throws FileNotFoundException {
    String bezeichner = "[A–Z]+[a–z]*[A–Z]?[1–9][0–9]*";
    Pattern pattern = Pattern.compile(bezeichner);
    Scanner sc = new Scanner(new FileInputStream("file.txt"));
    while(sc.hasNextLine()) {
        Matcher matcher = pattern.matcher(sc.nextLine());
        if (matcher.find()) System.out.println(matcher.group());
    }
    sc.close();
}
```

Computerübung 4.11

Eine mögliche rG könnte folgende sein:

$G = (N, T, P, s)$ mit

$N = \{S, vK, K, nK1, nK2\}$,

$T = \{\ ,,0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$,

$P = \{$S$\rightarrow$ $1\ vK \mid 2\ vK \mid 3\ vK \mid 4\ vK \mid 5\ vK \mid 6\ vK \mid 7\ vK \mid 8\ vK \mid 9\ vK \mid$
$0\ K \mid 1\ K \mid 2\ K \mid 3\ K \mid 4\ K \mid 5\ K \mid 6\ K \mid 7\ K \mid 8\ K \mid 9\ K$

vK $\rightarrow$ $0\ vK \mid 1\ vK \mid 2\ vK \mid 3\ vK \mid 4\ vK \mid 5\ vK \mid 6\ vK \mid 7\ vK \mid 8\ vK \mid 9\ vK \mid$
$0\ K \mid 1\ K \mid 2\ K \mid 3\ K \mid 4\ K \mid 5\ K \mid 6\ K \mid 7\ K \mid 8\ K \mid 9\ K \mid \varepsilon$

$K \rightarrow$ $,\ nK1$

$nK1 \rightarrow$ $0\ nK2 \mid 1\ nK2 \mid 2\ nK2 \mid 3\ nK2 \mid 4\ nK2 \mid 5\ nK2 \mid 6\ nK2 \mid 7\ nK2 \mid 8\ nK2 \mid 9\ nK2$

$nK2 \rightarrow$ $0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9\ \}$,

$s = S$

Der daraus resultierende NEA verwendet als Dezimaltrennzeichen einen Punkt, um den Unterschied zum Komma als Trennzeichen der Terminale deutlich zu machen.

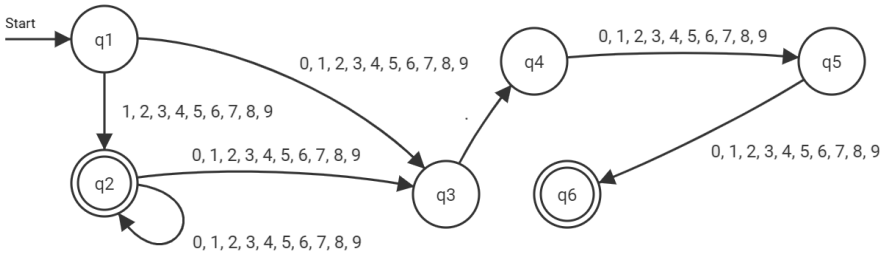


Abb. 26: Äquivalenter NEA

Zum Aufzeigen von Fehlern in Eingaben könnten folgende Token behilflich sein.

Token:	**Akzeptiert**
Expression:	(0|[1-9][0-9]*),[0-9][0-9]
Token:	**KommaUnstimmigkeit**
Expression:	(0|[1-9][0-9]*)(„+)?[0-9][0-9]
Token:	**VorkommastellenUnstimmigkeit**
Expression:	(0[0-9]+)?,[0-9][0-9]
Token:	**NachkommastellenUnstimmigkeit**
Expression:	(0|[1-9][0-9]*),([0-9]|[0-9][0-9][0-9][0-9]+)?

Abb. 27: Token zum Erkennen fehlerhafter Eingaben

5 Sprachübersetzer

Übung 5.1

Wird ein Programm mit einem Interpreter ausgeführt, so vergeht zwischen jeder Anweisung eine kurze Zeit, die der Interpreter benötigt, um die Anweisung zu analysieren und in Prozessor-Instruktionen zu übersetzen. Ganz besonders ineffizient wird diese Arbeitsweise bei der Abarbeitung einer Schleife. Die sich dort befindenden Befehle werden mit jeder Iteration neu analysiert.
Der Compiler hingegen übersetzt das gesamte Programm vor der eigentlichen Ausführung. Es liegt auf der Hand, dass mit dieser Arbeitsweise eine höhere Effizienz als bei Benutzung eines Interpreters erzielt wird.
Eine besondere Stärke von Interpretern ist, Code als Zeichenkette dynamisch zu generieren und anschließend ausführen zu können. Damit ist es möglich, einzelne Befehle im Voraus zu testen.
Klassische Compiler und Interpreter werden in Just-in-time Compilern zu einem mächtigen Werkzeug kombiniert, auf das z.B. das JRE (Java Runtime Enviroment) angewiesen ist. Diese Art von Compilern erzeugt den Code in Maschinensprache kurz bevor er gebraucht wird - schnelleren und für den Prozessor besser verständlicheren Code.

Computerübung 5.1

Zunächst öffnen Sie FLACI und navigieren zu *Compiler und Interpreter*. Dort angekommen, können Sie ein neues Diagramm erstellen.
Nun haben Sie auf der linken Seite die verschiedenen Bausteine, welche Sie per Drag-and-Drop auswählen und hinzufügen können. Bausteine sind miteinander verbunden, wenn sich eine kleine grüne Linie zwischen den Bausteinen befindet.

Computerübung 5.2

Ein Java Programm wird erst in Bytecode (*.class*) übersetzt, bevor eine ausführbare Datei (*.jar*) erstellt werden kann.

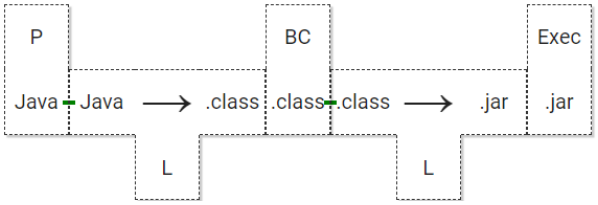


Abb. 28: Modellierung der Verarbeitung eines Java Programms

Computerübung 5.3

Die Übersetzung läuft ohne Zwischenübersetzung ab.

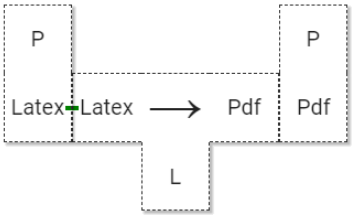


Abb. 29: Compilation Latex zu PDF

Computerübung 5.4

Keine Lösung.

Übung 5.2

Bootstrapping im Allgemeinen, aber in Bezug auf die Informatik, bedeutet, aus einem einfachen System ein komplexes zu erzeugen.
Ein einfaches System, das alle geraden Nicht-Primzahlen ablehnt, bestünde in einer Liste von natürlichen Zahlen z, $z \geq 2$, in der die Zahl 2 als Primzahl und alle weiteren Vielfache als Nicht-Primzahl markiert werden.
Eine Erweiterung dieses Systems ergibt sich, indem zur nächsten unmarkierten Zahl gesprungen und das Verfahren wiederholt wird.
Das komplexe System bedarf keiner neuen Erweiterung, wenn die zu prüfende Zahl erreicht wurde.
Dieser Algorithmus ist unter dem Namen *Sieb des Eratosthenes* bekannt.

Computerübung 5.5

ai f+4:
ai passt zu dem Token *ID*. Das **f** aus **f+4** gehört ebenfalls dem Token *ID* an und **+4** gehört zu dem Token *NUM* (siehe Abb. 30 links)

if
Die Eingabe **if** passt zu dem Token *IF*, da das Token *IF* vor dem Token *ID* definiert wurde und FLACI nach der *First-wins-Strategie* funktoniert. (siehe Abb. 30 mitte)

aifb+4
aifb passt zu dem Token *ID*, da die Eingabe zusammengeschrieben ist und das **a** nur in *ID* vorkommt. Die **+4** passt zu dem Token *NUM*. (siehe Abb. 30 rechts)

Eingabe:

```
1 ai f+4
```

Ausgabe:

```
1 ([ID,"ai"], [ID,"f"],
  [NUM,"+4"])
```

Eingabe:

```
1 if
```

Ausgabe:

```
1 ([IF,"if"])
```

Eingabe:

```
1 aifb+4
```

Ausgabe:

```
1 ([ID,"aifb"], [NUM,"+4"])
```

Abb. 30: Ein- und Ausgaben in FLACI

Computerübung 5.6

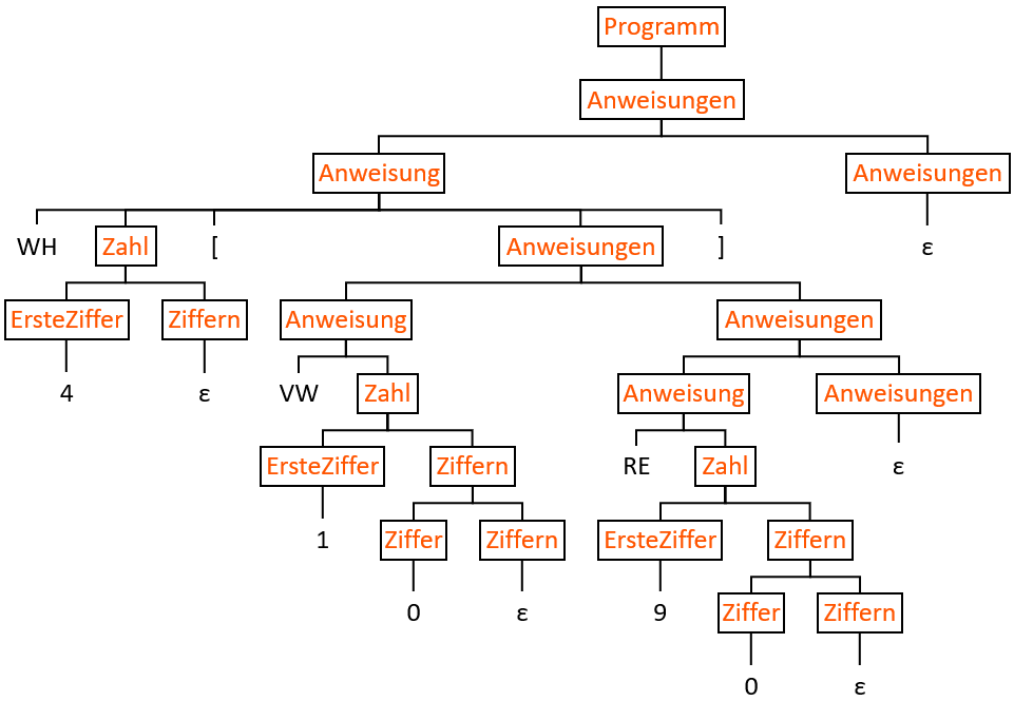


Abb. 31: Ableitungsbaum von WH 4 [VW 10 RE 90]

Übung 5.3

Tokenklasse	**Regulärer Ausdruck**
Farbwert	weiss | blau | gelb | gruen | rot | schwarz
Zahl	$[1-9][0-9]^*$

Durch das Reduzieren von G auf G' entfallen die Regeln für *Farbwert*, *Zahl* und weitere Regeln zum Beschreiben einer Zahl. Zudem entfallen die dafür notwendigen Terminale und Nichtterminale.

6 Kellerautomaten und kontextfreie Sprachen

Übung 6.1

Das System ist simpel - eine geöffnete Klammer wird auf den Stapel gelegt, mit einer geschlossenen Klammer wird eine geöffnete Klammer vom Stapel genommen. Das grau markierte bleibt dabei unberücksichtigt.

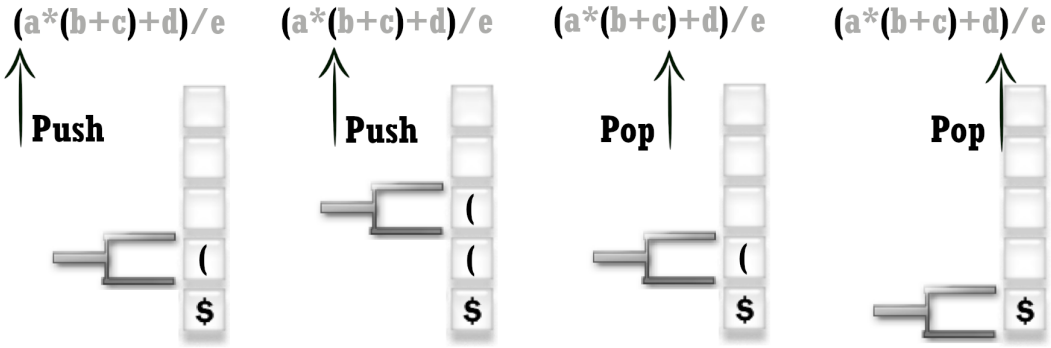


Abb. 32: Stapel zur Erkennung korrekt geklammerter arithmetischer Ausdrücke

Computerübung 6.1

Siehe Lehrbuch, S. 144.

Computerübung 6.2

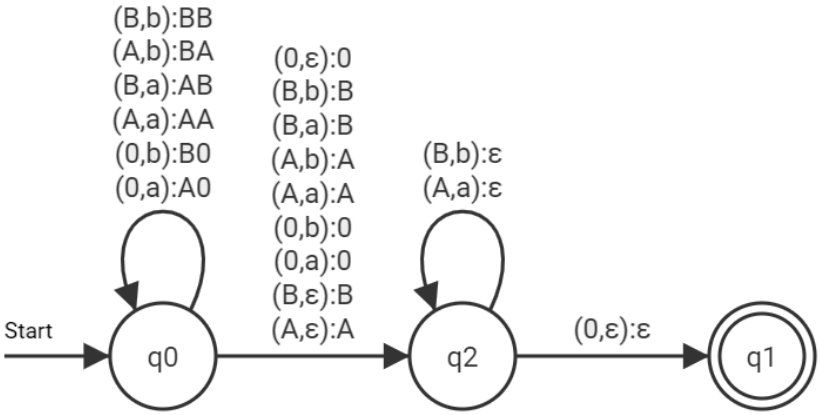


Abb. 33: NKA für die Sprache der Palindrome über {a, b}

Übung 6.2

Zu zeigen ist, dass zu jedem NKA $M = (Q, \Sigma, \Gamma, \delta, q_0, k_0)$ ein NKA $M' = (Q', \Sigma', \Gamma', \delta', S, k'_0)$, wobei S die Menge von Startzuständen beschreibt, derart existiert, dass $L(M) = L(M')$.

Die Konstruktion von M' aus M lässt sich durch folgende Erweiterungen beschreiben:

- $Q' = Q \cup S$
- $\Sigma' = \Sigma$
- $\Gamma' = \Gamma$
- $\delta' = \delta \cup \{(x, y) \mid x = (q, \varepsilon, k'_0),\ y = \{(q_0, k_0)\})$ und $q \in S\}$
- $k'_0 = k_0$

Jeder der neuen Startzustände ist also mit dem ursprünglichen Startzustand q_0 verbunden, wobei diese Übergänge den Stapel sowie das aktuelle Eingabezeichen unverändert lassen.

Um den Beweis der Äquivalenz zu vervollständigen, ist auch die Konstruktion von $M'' = (Q'', \Sigma'', \Gamma'', \delta'', q''_0, k''_0)$ aus M' Voraussetzung. Diese geschieht analog. Für M'' gilt:

- $Q'' = Q' \cup \{q''_0\}$
- $\Sigma'' = \Sigma'$
- $\Gamma'' = \Gamma'$
- $\delta'' = \delta' \cup \{(x, y) \mid x = (q''_0, \varepsilon, k''_0),\ y = \{(q, k'_0) \mid q \in S\})$
- $k''_0 = k'_0$

Computerübung 6.3

Zustand	Eingabe	Keller
q_0	`baab`	`$`
q_1	`baab`	`S$`
q_1	`baab`	`bSb$`
q_1	`aab`	`Sb$`
q_1	`aab`	`b$`

Tabelle 1: Erfolglose Konfigurationsreihenfolge für `baab`

Computerübung 6.4

Ein NKA, der nicht der von FLACI durchgeführten kfG→NKA - Transformation entstammt, ist der nachfolgende. Die Lösung zu Übung 6.1 beschreibt die Arbeitsweise; der Stapel dient lediglich zur Auskunft über die Anzahl der noch zu schließenden Klammern, um eine Grundlage für die Korrektheit eines arithmetischen Ausdrucks zu bekommen.

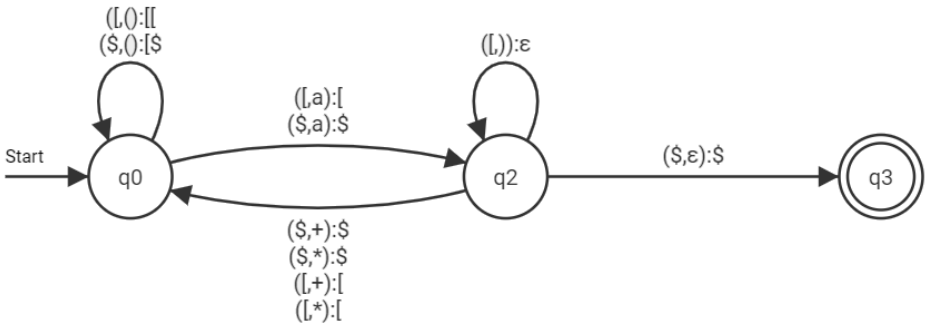


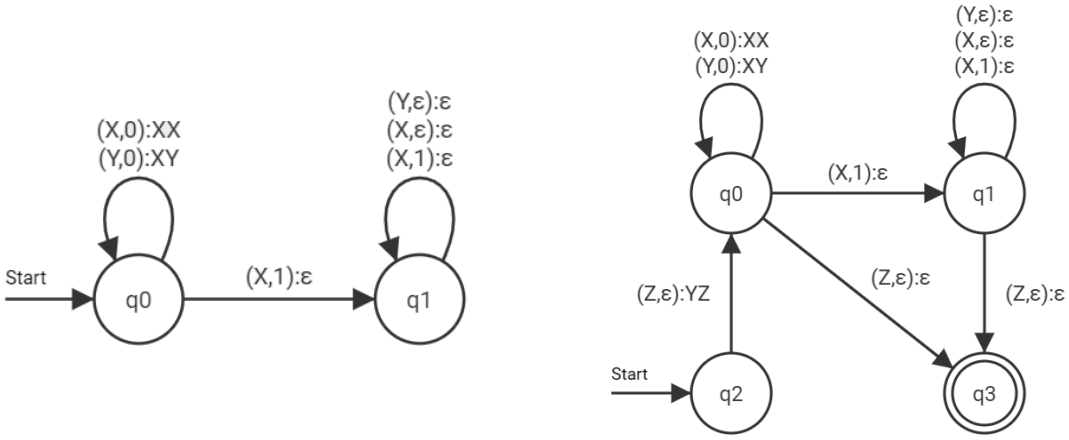
Abb. 34: NKA für arithmetische Ausdrücke

Übung 6.3

Keine Lösung.

Computerübung 6.5

Der (in FLACI nicht anwendbare) NKA6 (links) wird zu einem NKA7 (rechts) umkonstruiert.



Die nach Konvertierung erhaltene Grammatik (links) und die durch FLACI optimierte (rechts) sehen folgendermaßen aus:

```
START   -> q2_Z_q3 q1_Y_q1 | q0_Y_q1 q1_Y_q1
 | 0 q0_X_q1 q1_Y_q1
q1_X_q1 -> 1 | EPSILON
q1_Y_q1 -> EPSILON
q2_Z_q3 -> q0_Y_q1 q1_Y_q1 | 0 q0_X_q1
 q1_Y_q1
q0_Y_q1 -> 0 q0_X_q1 q1_Y_q1
q0_X_q1 -> 0 q0_X_q1 q1_X_q1 | 1

```

```
START -> D | 0 C | B
A     -> 1
B     -> 0 C
C     -> 0 C A | 1 | 0 C
D     -> 0 C | B

```

Computerübung 6.6

Siehe Lösung zu Computerübung 6.5.

Computerübung 6.7

Der Kellerinhalt wird mit jedem a um ein | erweitert. Folgt ein b beginnt die "Abbau-Phase". Zu dem Endzustand q_2 kann man nur gelangen, wenn das TOS ein $\$$ ist. Im Fall, dass mehr b abgebaut werden sollen, würde der Automat stoppen, da kein Übergang der Form $\delta(q_1, b, \$) = (q_n, k)$ existiert.

Computerübung 6.8

Ein NKA, der die Sprache L der Palindrome beschreibt, sieht folgendermaßen aus:

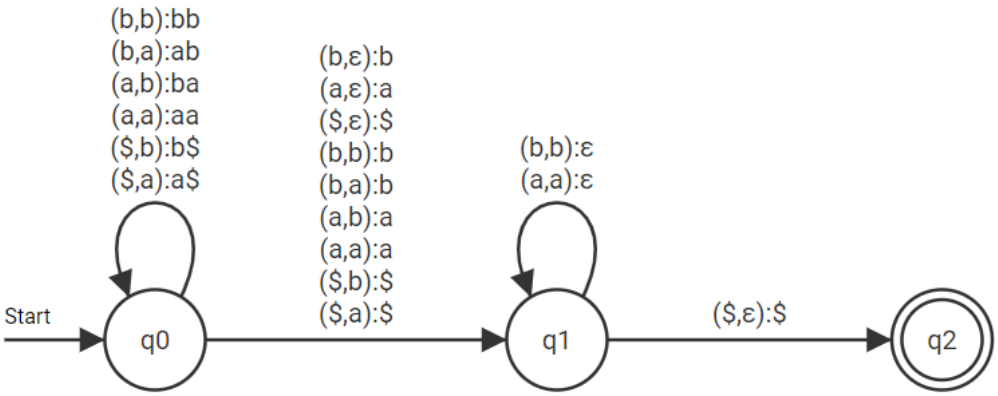


Abb. 35: NKA für die Sprache der Palindrome

Folgend seien zwei Beispielwörter gegeben und die Stelle gekennzeichnet, an der der Übergang zum "Keller-Abbau-Zustand" q_1 stattfindet.

Computerübung 6.9

Während die Wortmitte durch den NKA aus der letzten Übungsaufgabe mit jedem Clone erneut zufällig bestimmt wird, bis einer der Clone das Palindrom akzeptiert, geschieht dies bei dem gegebenen DKA (Lehrbuch, S. 160, Beispiel 6) mit der Kennzeichnung der Wortmitte durch ein c. Der Übergang zu q_1 erfolgt mit diesem Zeichen.

Computerübung 6.10

Eine Umwandlung der kfG in einen NKA erfolgt über die eingebaute Funktion in FLACI. Probiert man nun das Wort WH 4 [VW 100 RE 90] mit dem NKA abzuleiten, wird man mit dem Hinweis "Mehr als 50 Maschinen wurden simuliert. Suche wird abgebrochen." enttäuscht. Dieser Hinweis bedeutet nicht, dass das Wort nicht akzeptiert wurde. Es bedeutet nur, dass der Rechenaufwand, um die erfolgreiche Konfigurationsfolge zu erhalten, zu groß ist. Dieses Beispiel zeigt die Ineffizienz des Nichtdeterminismus.

Computerübung 6.11

Ausgangsgrammatik: $G = (N, T, P, s)$ mit

$$
\begin{aligned}
N &= \{A, B, C, D, E, F\},\\
T &= \{\text{a,b}\},\\
P &= \{A \to \text{a } B \text{ b} \mid \text{a } A \mid D\ ,\\
&\quad\ D \to E\ ,\\
&\quad\ E \to \text{ab}\ ,\\
&\quad\ B \to F \text{ a}\ ,\\
&\quad\ C \to \text{abba}\ ,\\
&\quad\ F \to F\ \},\\
s &= A
\end{aligned}
$$

Nach der Vereinfachung entsteht folgende Grammatik $G' = (N', T', P', s')$ mit

$$
\begin{aligned}
N' &= \{A, D, E\},\\
T' &= \{\text{a,b}\},\\
P' &= \{A \to \text{a } A \mid D\ ,\\
&\quad\ D \to E\ ,\\
&\quad\ E \to \text{ab}\ \},\\
s' &= A
\end{aligned}
$$

Computerübung 6.12

Nach der Entfernung der Kettenregeln mithilfe von FLACI entsteht folgende Grammatik $G'' = (N'', T'', P'', s'')$ mit

N'' = $\{A\}$,
T'' = {a,b},
P'' = $\{A \rightarrow \text{a } A \mid \text{ab }\}$,
s'' = A

Computerübung 6.13

Grammatik:

```
S -> a S a | a a | A
A -> b A | B
B -> b
```

Nach Entfernung von Kettenregeln:

```
S -> a S a | a a | b A | b
A -> b A | b
```

Nach Anlegen neuer Nichtterminale:

```
S -> Xa S Xa | Xa Xa | Xb A | Xb
A -> Xb A | Xb
Xa -> a
Xb -> b
```

Nach Reduzierung:

```
S -> Xa Z1 | Xa Xa | Xb A | Xb
A -> Xb A | Xb
Z1 -> S Xa
Xa -> a
Xb -> b
```

Übung 6.4

Siehe Lehrbuch, S. 168, Abb. 6.11.

Übung 6.5

Fall: $A \rightarrow BC$, $B \overset{*}{\Rightarrow} v$, $C \overset{*}{\Rightarrow} x$
Ergebnis: Aus $vwx = vx$ folgt $w = \varepsilon$ und $v, x \neq \varepsilon$, also gilt $vx \neq \varepsilon$.

7 LL(k)-Sprachen

Übung 7.1

$FIRST(a) = a$
$FIRST(Xa) = FIRST(X)$, da $\varepsilon \notin FIRST(X)$, und $FIRST(X) = FIRST(a) \cup FIRST(Xa)$ (siehe Lehrbuch, S.177, Definition 7.4). Also gilt $FIRST(Xa) \cap FIRST(a) \neq \emptyset$.

Ein LL(1)-Parser, der auf solche Gegebenheiten nicht vorbereitet ist, würde an dieser Stelle in eine Endlosschleife fallen und womöglich eine *Stack-Overflow*-Fehlermeldung hervorbringen. Der Grund liegt in der Linksrekursivität bei der Regel $X \longrightarrow Xa$, durch die $FIRST(Xa)$ nicht bestimmt werden kann.

Übung 7.2

Da in dieser Grammatik kein Nichtterminal zu ε abgeleitet werden kann, genügt es, auf Forderung 1 zu prüfen.

Für E: $FIRST(E+T) = FIRST(E) = FIRST(E+T) \cup FIRST(T)$

Die hier vorliegende Verarbeitung der rechten Regelseite $E+T$ ist äquivalent zu der aus Übung 7.1 - erneut zeigt sich, dass Linksrekursivität kein Bestandteil einer LL(1)-Grammatik sein kann. Das Prüfen der weiteren Regeln ist demnach nicht von Bedeutung.
FLACI ist auf Linksrekursivität vorbereitet, sodass der Algorithmus zum Prüfen, ob die gegebene Grammatik LL(1) Grammatik ist, terminiert. Als Ergebnis für die erste Regel liefert FLACI folgendes:

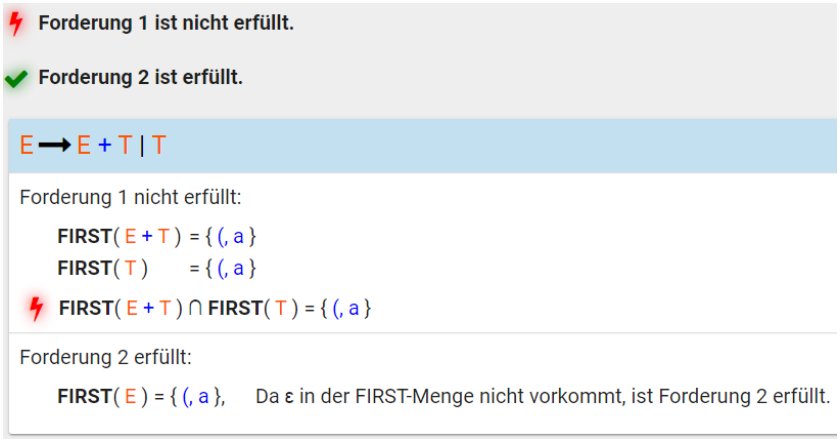


Abb. 36: Verletzung von Forderung 1

Die Berechnung der First-Menge wurde hier reduziert auf:

$$FIRST(E+T) = FIRST(T) = FIRST(F) = \{(, a\}$$

Computerübung 7.1

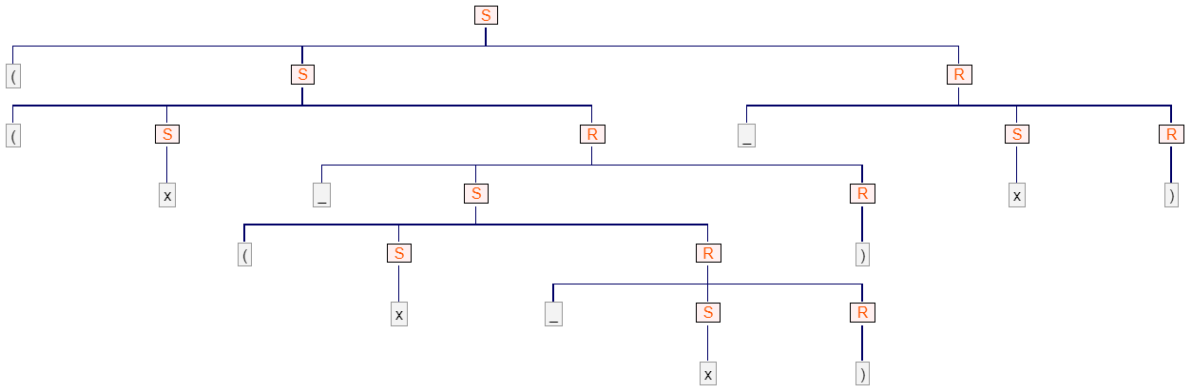


Abb. 37: Ableitungsbaum für `((x_(x_x))_x)`

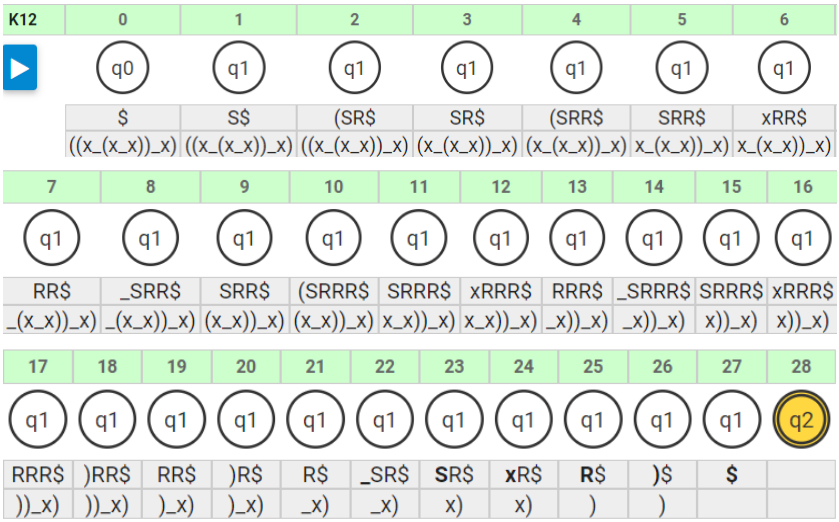


Tabelle 2: Konfigurationenfolge für `((x_(x_x))_x)`

Zustand	Eingabe	Vorausschau	Keller	Übergang
q_0	`((x_(x_x))_x)`	`(`	`$`	$\delta(q_0, \varepsilon, \$) = (q_1, \mathrm{S}\$)$
q_1	`((x_(x_x))_x)`	`(`	`S$`	$\delta(q_1, \varepsilon, \mathrm{S}) = (q_1, (\mathrm{SR})$
q_1	`((x_(x_x))_x)`	`(`	`(SR$`	$\delta(q_1, (, () = (q_1, \varepsilon)$
q_1	`(x_(x_x))_x)`	`(`	`SR$`	$\delta(q_1, \varepsilon, \mathrm{S}) = (q_1, (\mathrm{SR})$
q_1	`(x_(x_x))_x)`	`(`	`(SRR$`	$\delta(q_1, (, () = (q_1, \varepsilon)$
q_1	`x_(x_x))_x)`	`x`	`SRR$`	$\delta(q_1, \varepsilon, \mathrm{S}) = (q_1, \mathtt{x})$
q_1	`x_(x_x))_x)`	`x`	`xRR$`	$\delta(q_1, \mathtt{x}, \mathtt{x}) = (q_1, \varepsilon)$
q_1	`_(x_x))_x)`	`_`	`RR$`	$\delta(q_1, \varepsilon, \mathrm{R}) = (q_1, \_\mathrm{SR})$

Zustand	Eingabe	Vorausschau	Keller	Übergang
q_1	`_(x_x))_x)`	`_`	`_SRR$`	$\delta(q_1, \_, \_) = (q_1, \varepsilon)$
q_1	`(x_x))_x)`	`(`	`SRR$`	$\delta(q_1, \varepsilon, \mathtt{S}) = (q_1, \mathtt{(SR)}$
q_1	`(x_x))_x)`	`(`	`(SRRR$`	$\delta(q_1, \mathtt{(}, \mathtt{(}) = (q_1, \varepsilon)$
q_1	`x_x))_x)`	`x`	`SRRR$`	$\delta(q_1, \varepsilon, \mathtt{S}) = (q_1, \mathtt{x})$
q_1	`x_x))_x)`	`x`	`xRRR$`	$\delta(q_1, \mathtt{x}, \mathtt{x}) = (q_1, \varepsilon)$
q_1	`_x))_x)`	`_`	`RRR$`	$\delta(q_1, \varepsilon, \mathtt{R}) = (q_1, \mathtt{\_SR})$
q_1	`_x))_x)`	`_`	`_SRRR$`	$\delta(q_1, \_, \_) = (q_1, \varepsilon)$
q_1	`x))_x)`	`x`	`SRRR$`	$\delta(q_1, \varepsilon, \mathtt{S}) = (q_1, \mathtt{x})$
q_1	`x))_x)`	`x`	`xRRR$`	$\delta(q_1, \mathtt{x}, \mathtt{x}) = (q_1, \varepsilon)$
q_1	`))_x)`	`)`	`RRR$`	$\delta(q_1, \varepsilon, \mathtt{R}) = (q_1, \mathtt{)})$
q_1	`))_x)`	`)`	`)RR$`	$\delta(q_1, \mathtt{)}, \mathtt{)}) = (q_1, \varepsilon)$
q_1	`)_x)`	`)`	`RR$`	$\delta(q_1, \varepsilon, \mathtt{R}) = (q_1, \mathtt{)})$
q_1	`)_x)`	`)`	`)R$`	$\delta(q_1, \mathtt{)}, \mathtt{)}) = (q_1, \varepsilon)$
q_1	`_x)`	`_`	`R$`	$\delta(q_1, \varepsilon, \mathtt{R}) = (q_1, \mathtt{\_SR})$
q_1	`_x)`	`_`	`_SR$`	$\delta(q_1, \_, \_) = (q_1, \varepsilon)$
q_1	`x)`	`x`	`SR$`	$\delta(q_1, \varepsilon, \mathtt{S}) = (q_1, \mathtt{x})$
q_1	`x)`	`x`	`xR$`	$\delta(q_1, \mathtt{x}, \mathtt{x}) = (q_1, \varepsilon)$
q_1	`)`	`)`	`R$`	$\delta(q_1, \varepsilon, \mathtt{R}) = (q_1, \mathtt{)})$
q_1	`)`	`)`	`)$`	$\delta(q_1, \mathtt{)}, \mathtt{)}) = (q_1, \varepsilon)$
q_1	ε		`$`	$\delta(q_1, \varepsilon, \$) = (q_2, \varepsilon)$
q_2	ε		ε	Akzeptiert!

Tabelle 3: Konfigurationenfolge für `((x_(x_x))_x)` als Tabelle

Übung 7.3

Das vollständige Eingabewort in diesem Analyseprozess ist nicht etwa `a * a + a`, sondern `a * a + a $`. Demnach stünde hinter dem Resteingabewort in jeder Zeile noch ein $. Dieses $ dient als "Eingabe-Schluss-Zeichen" und befände sich auch noch in den letzten drei Zeilen in der Spalte *Eingabewort*. Üblich ist es bei der Notation solcher Tabellen, auf dieses Zeichen zu verzichten.

Keller	Eingabewort	Regel
$E\$$	`a * a + a`	E → T E'
$TE'\$$	`a * a + a`	T → F T'
$FT'E'\$$	`a * a + a`	F → a
$aT'E'\$$	`a * a + a`	
$T'E'\$$	`* a + a`	T' → *FT'
$*FT'E'\$$	`* a + a`	
$FT'E'\$$	`a + a`	F → a
$aT'E'\$$	`a + a`	
$T'E'\$$	`+ a`	T' → ε
$E'\$$	`+ a`	E' → +TE'
$+TE'\$$	`+ a`	
$TE'\$$	`a`	T → FT'
$FT'E'\$$	`a`	F → a
$aT'E'\$$	`a`	
$T'E'\$$		T' → ε
$E'\$$		E' → ε
$\$$		

Übung 7.4

Um die Parsetabelle richtig zu erzeugen, müssen FIRST- und ggf. FOLLOW-Mengen erzeugt werden. Diese lauten:

Für E: $FIRST(TE') = \{a, (\}$
Für E': $FIRST(+TE') = \{+\}, FIRST(\varepsilon) = \{\varepsilon\}, FIRST(+TE') \cap FIRST(\varepsilon) = \emptyset$
$FOLLOW(E') = \{), \$\}, FOLLOW(E') \cap FIRST(E') = \emptyset$
Für T: $FIRST(FT') = \{a, (\}$
Für T': $FIRST(*FT') = \{*\}, FIRST(\varepsilon) = \{\varepsilon\}, FIRST(*FT') \cap FIRST(\varepsilon) = \emptyset$
$FOLLOW(T') = \{+,), \$\}, FOLLOW(T') \cap FIRST(T') = \emptyset$
Für F: $FIRST(a) = \{a\}, FIRST((E)) = \{(\}, FIRST(a) \cap FIRST((E)) = \emptyset$

Für jede linke Regelseite E, E', T, T' und F wird in der Parsetabelle nun eine Zeile eingerichtet. Für jedes vorkommende Terminal inklusive dem $-Zeichen, als Eingabe-Schluss-Zeichen, eine Spalte. Das Feld [N, T] ergibt sich nun aus der rechten Regelseite R, mit N $\rightarrow$ R und $T \in FIRST(N) \cup FOLLOW(N)$, wenn $\varepsilon \in FIRST(N)$, sonst $T \in FIRST(N)$. Wir verzichten also auf die Angabe der vollständigen Regel in einem Feld, da dies nicht notwendig ist.

Übung 7.5

Die Parsetabelle ist in Lehrbuch, S. 182, Tabelle 7.1 zu sehen.

Keller	Eingabewort	Regel
S\$	`(x_(x_x))`	S $\rightarrow$ (S R
$(SR$\$	`(x_(x_x))`	
SR\$	`x_(x_x))`	S $\rightarrow$ x
xR\$	`x_(x_x))`	
R\$	`_(x_x))`	R $\rightarrow$ _ S R
$\_SR$\$	`_(x_x))`	
SR\$	`(x_x))`	S $\rightarrow$ (S R
$(SRR$\$	`(x_x))`	
SRR\$	`x_x))`	S $\rightarrow$ x
xRR\$	`x_x))`	
RR\$	`_x))`	R $\rightarrow$ _ S R
$\_SRR$\$	`_x))`	
SRR\$	`x))`	S $\rightarrow$ x
xRR\$	`x))`	
RR\$	`))`	R $\rightarrow$)
$)R$\$	`))`	
R\$	`)`	R $\rightarrow$)
)\$	`)`	
\$		

Tabelle 4: Analyseprozess des Wortes `(x_(x_x))`

Computerübung 7.2

Haben Sie die Grammatik korrekt in FLACI eingegeben (oder kopiert), wird FLACI Ihnen die LL(1)-Eigenschaft bestätigen.

Computerübung 7.3

Siehe zugehörige Erläuterung in Lehrbuch, S. 190 und 191.

Computerübung 7.4

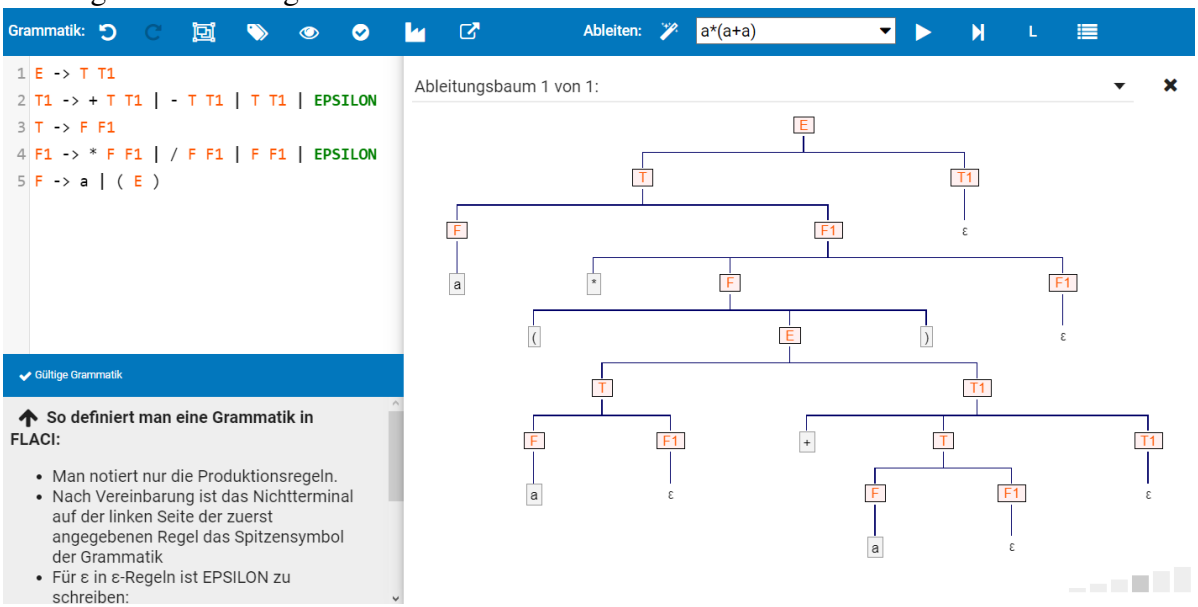
Gegeben ist die Grammatik $G = (N, T, P, s)$ mit

$$\begin{aligned} N &= \{E, T, F\}, \\ T &= \{a, (,), +, -, *, /\}, \\ P &= \{E \rightarrow T\{[+ \mid -]T\}, \\ &\quad\ T \rightarrow F\{[* \mid /]F\}, \\ &\quad\ F \rightarrow a \mid (E), \\ s &= E \end{aligned}$$

Die Transformationsregeln lauten:

1. $X \rightarrow \{Y\}$ zu $X \rightarrow YX \mid \varepsilon$
2. $X \rightarrow [Y]$ zu $X \rightarrow Y \mid \varepsilon$

Die Regel $E \rightarrow T\{[+ \mid -]T\}$ muss zunächst erweitert werden zu $E \rightarrow TT'$; $T' \rightarrow \{[+ \mid -]T\}$. Letztere Regel wird gemäß Transformationsregel 1 zu $T' \rightarrow [+ \mid -]TT' \mid \varepsilon$ und gemäß Transformationsregel 2 zu $T' \rightarrow +TT' \mid -TT' \mid TT' \mid \varepsilon$. Analog dazu wird die Regel $T \rightarrow F\{[* \mid /]F\}$ behandelt.
Das Ergebnis sieht folgendermaßen aus:



Computerübung 7.5

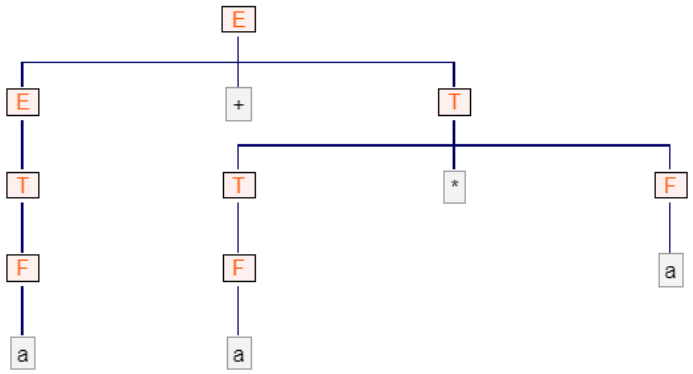


Abb. 38: Einziger Ableitungsbaum für a+a*a

Die gegebene Grammatik ist aufgrund der Linksrekursivität keine LL(1)-Grammatik. Eine äquivalente LL(1)-Grammatik ist in Lehrbuch, S. 185, Beispiel 7.7 zu sehen.

Computerübung 7.6

Keine Lösung.

Computerübung 7.7

Python benutzt zur Feststellung, zu welchem `if` ein `else` gehört, die Anzahl der Tabulatoren vor letzterem. Ein `else` gehört zu dem `if`, das gleich weit eingerückt ist.

```
if True:
    if False:
        print("Nicht moeglich!")
else:
    print("Unmoeglich!")
```

Da der `else`-Block zum ersten `if` gehört, wird keine Ausgabe getätigt.

8 LR(k)-Sprachen

Computerübung 8.1

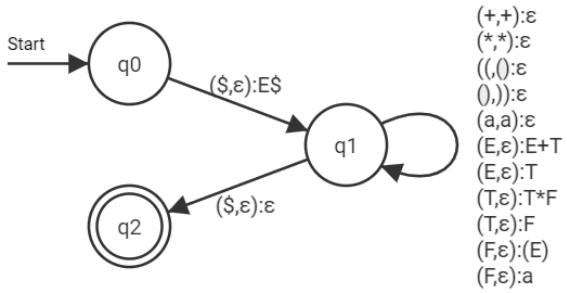


Abb. 39: Ineffizienter NKA für die Sprache der arithmetischen Ausdrücke

Übung 8.1

`a+a*(a+a)` $\Rightarrow$ `F+a*(a+a)` $\Rightarrow$ `T+a*(a+a)` $\Rightarrow$ `E+a*(a+a)` $\Rightarrow$ `E+F*(a+a)` $\Rightarrow$ `E+T*(a+a)` ...

Die Information, dass das Vorausschauzeichen ein $*$ ist, genügt nicht, um ohne Intelligenzeinsatz sicher bestimmen zu können, ob die Regel $E \rightarrow E+T$ oder $F \rightarrow a$ auf die Satzform angewendet werden soll.

Übung 8.2

Das Top-Down-Verfahren scheitert bereits an der Regelauswahl in $E \rightarrow E+T|T$, da in den FIRST-Mengen beider rechter Regelseiten das erste Vorausschauzeichen `a` enthalten ist. Offenbar lässt sich ein Wort der Sprache bottom-up "länger" analysieren, bevor es zu Unklarheiten der Bestimmung nächster anzuwendender Regeln kommt.

Übung 8.3

Keine Lösung.

Computerübung 8.2

Ausgabe:

```
1 ([a,"a"], [+,"+"], [a,"a"],
  [*,"*"], [(,"("], [a,"a"],
  [+,"+"], [a,"a"], [),")"])
```

Arbeitsergebnis des Scanners: Zuordnung der Lexeme zu den Tokenklassen

Ausgabe:

```
1 true
```

Arbeitsergebnis des Parsers: Angabe, ob syntaktische Korrektheit vorliegt

Übung 8.4

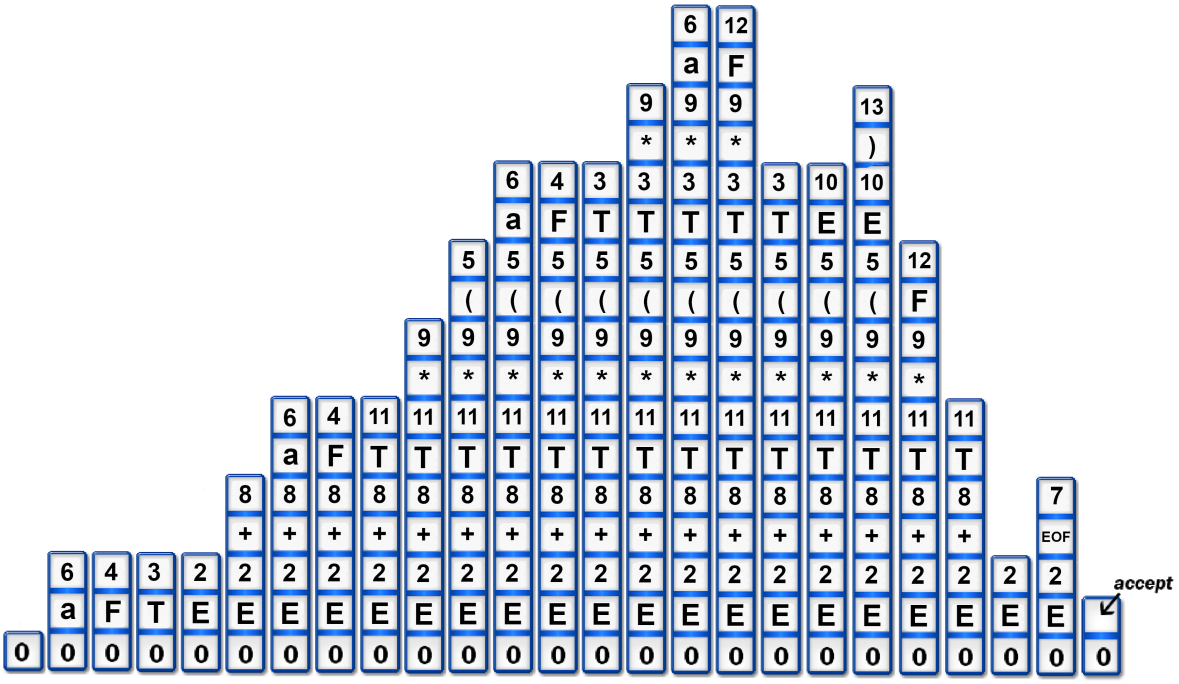


Abb. 40: Analyseprozess a+a*(a+a)

Computerübung 8.3

```
Bei der Erzeugung des entsprechenden LALR(1)-Parsers ist ein Konflikt
aufgetreten.

Conflict in grammar (state:2, token:EOF)
  reduce by F -> E
  shift T
Conflict in grammar (state:2, token:+)
  reduce by F -> E
  shift *
Conflict in grammar (state:3, token:*)
  reduce by E -> T
  shift F
Conflict in grammar (state:10, token:))
  reduce by F -> E
  shift undefined
Conflict in grammar (state:10, token:+)
```

Abb. 41: Kleiner Ausschnitt aus einer Reihe von Problemmeldungen

Computerübung 8.4

Definieren Sie zunächst die Grammatik in FLACI und erstellen Sie anschließend einen neuen Compiler-Baustein unter der Auswahl von *Neuer Compiler (mit TDL von Grammatik)*. In der Tokenliste wird das Token `num` durch den regulären Ausdruck **0|[1-9][0-9]*** untermauert. Compilieren Sie diesen in der Sprache TDL geschriebenen Compiler in die Sprache JavaScript. Der Compiler-Quelltext bekräftigt mit dem Kommentar

```
/* Jison generated parser */
```

in der ersten Zeile, dass Jison für die automatisierte Generierung verantwortlich war. Das fertige Compiler-Modell wird noch durch zwei Programmbausteine (Ein- und Ausgabe) ergänzt, mit denen die Analyse der gegebenen Wörter `2+4*5` und `2++4*5` vollzogen wird.

Computerübung 8.5

Bei dieser Aufgabe wird dieselbe Vorgehensweise wie in Computerübung 8.4 verwendet. Voraussetzung dafür ist das Vorhandensein der PL/0 Grammatik in Ihrer FLACI-Sammlung.

Ein Beispielwort, das der Parser akzeptiert, ist ein rekursiv arbeitendes Programm zur Bestimmung der n-ten Fibonacci-Zahl. Hier wurde $n := 10$ gesetzt. Das Resultat wird in z gespeichert.

```
var x, y, z, n;
procedure fib;
    begin
        if n >= 2 then
            begin
                n := n - 1;
                call fib;
                z := x + y;
                x := y;
                y := z
            end
    end;
begin
    x := 0;
    y := 1;
    n := 10;
    call fib
end.
```

9 Sprachübersetzerprojekte

Übung 9.1

$G = (N, T, P, s)$ mit

$$
\begin{aligned}
N &= \{Musikstueck, Song, Note, Ton, Oktave, Laenge\},\\
T &= \{\text{C,D,E,F,G,A,H,0,1,2,4,8,-}\},\\
P &= \{Musikstueck \rightarrow Song\ ,\\
&\quad\ Song \rightarrow \varepsilon \mid Note\ Song\ ,\\
&\quad\ Note \rightarrow Ton\ Oktave\ \text{-}\ Laenge\ ,\\
&\quad\ Ton \rightarrow \text{C} \mid \text{D} \mid \text{E} \mid \text{F} \mid \text{G} \mid \text{A} \mid \text{H}\ ,\\
&\quad\ Oktave \rightarrow 0 \mid 1\ ,\\
&\quad\ Laenge \rightarrow 1 \mid 2 \mid 4 \mid 8\ \},\\
s &= Musikstueck
\end{aligned}
$$

Computerübung 9.1

Reduzierte Grammatik:

$G' = (N', T', P', s')$ mit

$$
\begin{aligned}
N' &= \{Musikstueck, Song, Note\},\\
T' &= \{\text{-}\},\\
P' &= \{Musikstueck \rightarrow Song\ ,\\
&\quad\ Song \rightarrow \varepsilon \mid Note\ Song\ ,\\
&\quad\ Note \rightarrow Ton\ Oktave\ \text{-}\ Laenge\ \},\\
s' &= Musikstueck
\end{aligned}
$$

Computerübung 9.2

Da nun das Token *Länge* über dem Token *Oktave* steht, wird, falls ein Ton in der 1. Oktave gespielt werden soll, nur die **1** ausgewertet und übrig bleibt ein Bindestrich mit einer *Laenge*. Dieser Bindestrich taucht jedoch bei keinem Token als erstes Zeichen auf, somit wird ein "Lexical Error" geworfen.

Computerübung 9.3

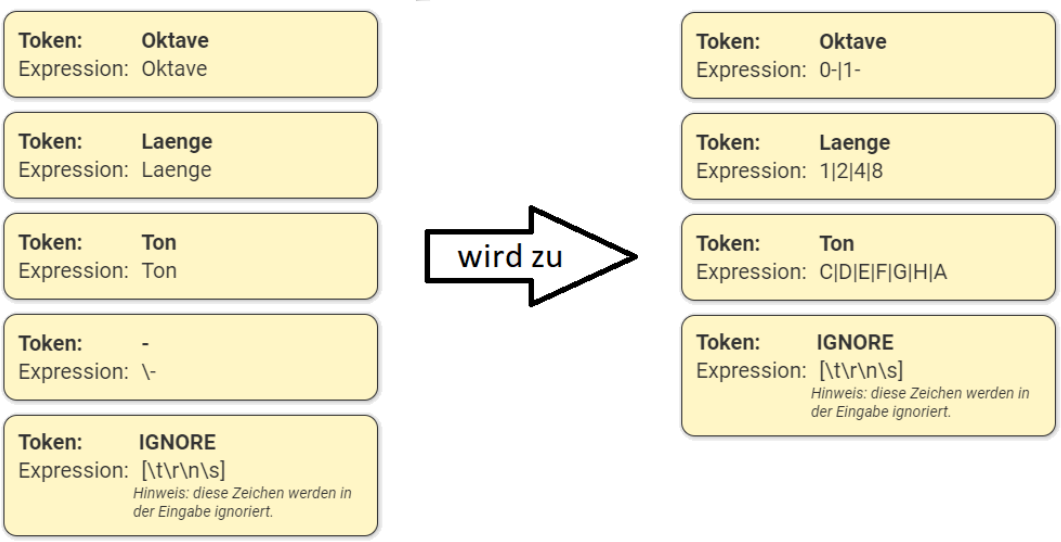
Folgen Sie den Anweisungen im Buch.

Computerübung 9.4

FLACI führt die Interpretation und die Compilation, in die akustisch hörbare Sprache, bis zum eingebauten Syntaxfehler durch. Das funktioniert, da ein Interpreter das Programm portionsweise analysiert und ausführt.

Computerübung 9.5

Zunächst erstellen wir einen Compilerbaustein mit der, in Computerübung 9.1, erstellten Grammatik. Diesen bearbeiten wir und ändern die Token folgendermaßen:



Aufgrund der *First-Wins-Strategie* bei Tokens in FLACI, ändern wir die Expression des Tokens **Oktave** von 0|1 zu **0-|1-**. Somit können wir das Token **-** löschen.
Anschließend müssen wir ein wenig Code zu den jeweiligen Definitionsregeln hinzufügen. Wir beginnen mit dem globalen Code.

Globaler Code für semantische Regeln

```
1 var xpos = 30;
```

xpos ist lediglich die x-Startkoordinate der ersten Note.
Nun fügen wir bei der Regel *Musikstueck -> Song* den Kopf einer SVG-Datei hinzu, da dieser nur einmal erstellt werden muss.
(Der ML -> SVG Compiler befindet sich ebenfalls in der FLACI Beispielsammlung)

S-Attribut für Musikstueck -> Song

```
1  return('<?xml version="1.0" ?> \n' +
2  '<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN" \n' +
3  '"http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"> \n' +
4  '<svg width="1200" height="150" xmlns="http://www.w3.org/2000/svg" \n' +
5  'xmlns:xlink="http://www.w3.org/1999/xlink"> \n' +
6      '<rect x="0" y="0" width="1200" height="150" fill="lightyellow" /> \n' +
7      '<line x1="0" y1="50" x2="1200" y2="50" style="stroke:gray;stroke-width:2"/> \n' +
8      '<line x1="0" y1="60" x2="1200" y2="60" style="stroke:gray;stroke-width:2"/> \n' +
9      '<line x1="0" y1="70" x2="1200" y2="70" style="stroke:gray;stroke-width:2"/> \n' +
10     '<line x1="0" y1="80" x2="1200" y2="80" style="stroke:gray;stroke-width:2"/> \n' +
11     '<line x1="0" y1="90" x2="1200" y2="90" style="stroke:gray;stroke-width:2"/> \n' +
12 $1 +
13 '</svg>');
```

In dem Kopf definieren wir die Größe der Zeichenfläche und wir bereiten den Hintergrund und die Linien vor.

Im Anschluss müssen noch die einzelnen Noten gezeichnet und richtig platziert werden. Dies erledigen wir, indem wir bei der Regel *Note -> Ton Oktave Laenge* folgenden Code hinzufügen:

S-Attribut für Note -> Ton Oktave Laenge

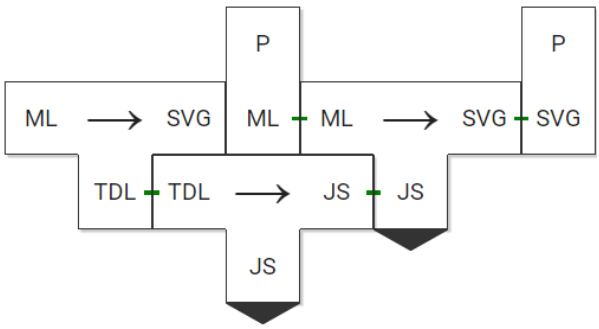
```
1  var okt = 0;
2  if($2 == 'b-') okt = 35;
3  var ton = 0;
4  switch($1){
5    case 'C': ton = 100; break;
6    case 'D': ton = 95; break;
7    case 'E': ton = 90; break;
8    case 'F': ton = 85; break;
9    case 'G': ton = 80; break;
10   case 'A': ton = 75; break;
11   case 'H': ton = 70; break;
12 }
13
14 switch($3){
15   case '1': $$ = '<ellipse cx="' + xpos +'" cy="' + (ton-okt) + '" rx="5" ry="3"
                    style="fill:none;stroke:black;stroke-width:2"/> \n';
16     break;
17
18   case '2': $$ = '<ellipse cx="' + xpos +'" cy="' + (ton-okt) +'" rx="5" ry="3"
                    style="fill:none;stroke:black;stroke-width:2"/> \n' +
19                 '<line x1="' + (xpos+5) + '" y1="' + (ton-okt) + '" x2="' + (xpos+5) +
                    '" y2="' + (ton-okt-25) + '" style="stroke:black;stroke-width:2"/> \n';
20     break;
21
22   case '4': $$ = '<ellipse cx="' + xpos + '" cy="' + (ton-okt) + '" rx="5" ry="3"
                    style="fill:black;stroke:black;stroke-width:2"/> \n' +
23                 '<line x1="' + (xpos+5) + '" y1="' + (ton-okt) + '" x2="' + (xpos+5) +
                    '" y2="' + (ton-okt-25) + '" style="stroke:black;stroke-width:2"/> \n';
24     break;
25
26   case '8': $$ = '<ellipse cx="' + xpos +'" cy="' + (ton-okt) +'" rx="5" ry="3"
                    style="fill:black;stroke:black;stroke-width:2"/> \n' +
27                 '<line x1="' + (xpos+5) +'" y1="' + (ton-okt) +'" x2="' + (xpos+5) +
                    '" y2="' + (ton-okt-25) +'" style="stroke:black;stroke-width:2"/> \n' +
28                 '<line x1="' + (xpos+5) +'" y1="' + (ton-okt-25) +'" x2="' + (xpos+15) +
                    '" y2="' + (ton-okt-15) +'" style="stroke:black;stroke-width:1"/> \n';
29     break;
30 }
31
32 xpos+= 30;
```

Hier eine kurze Erklärung, was der Code macht:

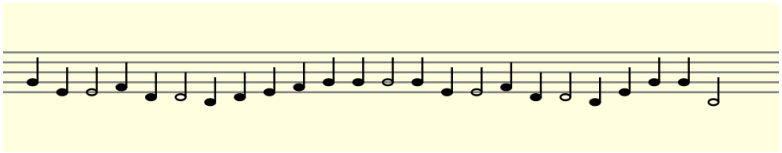
- Die Variable *okt* verschiebt die y-Koordinate um 35 Pixel, wenn die 1. Oktave gezeichnet werden soll
- Die Variable *ton* gibt die y-Koordinate für die Note an
- Die Switch-Anweisung ab Zeile 14, zeichnet die einzelnen Noten, abhängig von ihrer Länge
- Die global erstelle Variable *xpos* wird in der letzten Zeile um 30 erhöht, sodass die nächste Note weiter rechts gezeichnet wird

Zu guter Letzt müssen wir die Compilerdefinition (in TDL), mit dem Compiler Compiler, in einen lauffähigen Compiler übersetzen.

ML -> SVG Projekt sieht dann folgendermaßen aus:



Mit diesem lauffähigen Compiler können wir nun das Notenprogramm P, geschrieben in ML, in eine SVG-Datei umwandeln und es uns anschauen. Das sieht dann folgendermaßen aus:



Computerübung 9.6

Zunächst erstellen wir einen Interpreter mit TDL. Diesem fügen wir Regeln, Token und Code hinzu.

Computerübung 9.7

Nutzen Sie dazu die FLACI Beispielsammlung. Unter *Beispielsammlung → Compiler und Interpreter → QR-Code* finden Sie einen QR-Code Compiler. In den Eingabebaustein können Sie nun das Musikstück eingeben. Der generierte QR-Code wird dann im Ausgabefenster angezeigt.

Computerübung 9.8

Zunächst erstellen wir folgende Grammatik $G = (N, T, P, s)$ mit

$$
\begin{aligned}
N &= \{Text, Wort, Woerter\},\\
T &= \{\text{Notenwort,anderesWort}\},\\
P &= \{Text \rightarrow Woerter\ ,\\
&\quad\ Woerter \rightarrow Wort\ Woerter \mid \varepsilon\ ,\\
&\quad\ Wort \rightarrow \text{Notenwort} \mid \text{anderesWort}\ \},\\
s &= Text
\end{aligned}
$$

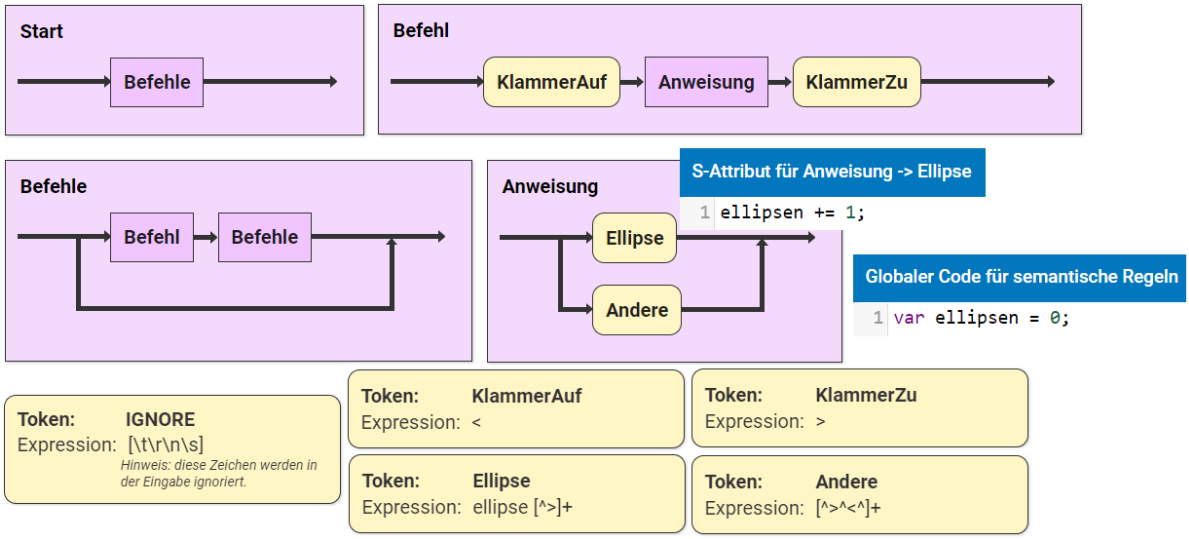


Abb. 42: Regeln, Token und Code für Ellipsen-Interpreter

Abb. 43: Musikbeispiel zu QR-Code

Mit dieser Grammatik erstellen wir nun einen Compilerbaustein. Den beiden Token fügen wir nun folgende reguläre Ausdrücke hinzu:

Token: **Notenwort**
Expression: [a-h^bA-H^B][0|1]-[1|2|4]

Token: **anderesWort**
Expression: [^\s]*

Folgende Regeln bekommen entsprechenden Code:

S-Attribut für Text -> Woerter : return($$);
S-Attribut für Wort -> Notenwort : $$ = $1 + " ";
S-Attribut für Wort -> anderesWort : $$ ="";

Der Compiler mit "Filterfunktion" ist fertig. Da der Compiler das Ergebnis in ML ausgibt,

kann man die Ausgabe des erstellten Compilers an einen ML-Eingabebaustein anknüpfen.

Übung 9.2

Aufstellung a)

r	n	b	q	k	b	n	r
p	p	p			p	p	p
				p			
			p				
			P	P			
P	P	P			P	P	
R	N	B	Q	K	B	N	R

Aufstellung b)

						r	
						p	p
							r
	B						K
	P				k		
N							
			R				

Da wir die Aufstellungen durch Tabellen problemlos darstellen können, sind diese FEN-Notationen fehlerfrei.

Computerübung 9.9

Die Grammatik akzeptiert jedes Wort, welches genau sieben "/" enthält. Vor und nach jedem "/" muss sich noch mindestens ein Terminal t, mit $t \neq$ \, befinden. Dabei spielt die Anzahl oder die Reihenfolge der Terminale keine Rolle.
So wird z.B. das Wort `1/1/1/1/1/1/1/1` von der Grammatik akzeptiert.

Computerübung 9.10

Siehe Lehrbuch S. 227, Abb. 9.12.

Computerübung 9.11

Keine Lösung.

Computerübung 9.12

Folgen Sie den Anweisungen im Buch.
Vergleich mit: `https://flaci.com/T54bqdrgr`

Computerübung 9.13

Ergebnis des fehlerhaften Eingabewortes:
`rnbqkbnr/ppp2ppp/4p3/3p4/311qqPP2/8/PPP2PPP/RNBQKBNR`

In Abb. 44 sieht man, dass zwei Damen platziert wurden und die maximale Anzahl an Figuren

Abb. 44: Fehlerhaftes Schachspiel

und Leerfeldern überschritten wurde, da (laut Eingabewort) nach den zwei Königinnen noch zwei Bauern und zwei Leerfelder folgen sollen. Das Ergebnis sieht jedoch anders aus - so folgen nach den zwei Königinnen nur ein Bauer, da das Spielfeld dann zuende ist.

Computerübung 9.14

Konvertieren wir den gegebenen NEA zunächst in einen äquivalenten DEA. So wird es deutlicher, dass auf eine Ziffer keine weitere Ziffer folgen kann, da man sonst im TRAP-Zustand landet.

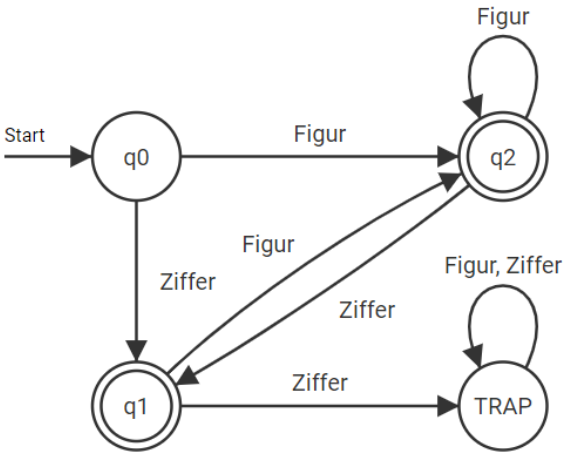


Abb. 45: Uberführungsfunktion eines DEA für FEN1 (genau eine Reihe)

Computerübung 9.15

Zunächst erstellen wir einen neuen Compiler mit der eben erstellten Grammatik. Anschließend ändern wir die Tokens Figur und Ziffer, wie im Buch beschrieben, zu [0-8] und `[PRNQKBprnqkb]`.

Um den erstellten FEN1 → FEN1 Compiler zu testen übergeben wir ihm ein Wort, welches offensichtlich gegen die 1. Forderung verstößt: `6r1/6pp/7r/1B5K/1P3k2/N7/3R4/521`.

Wie erwartet ist die Compilation fehlerhaft:

```
Error: Parse error on line 1:
.../1B5K/1P3k2/N7/3R4/521
-----------------------^
Expecting 'EOF', '/', 'Figur'
```

Computerübung 9.16

Wir übernehmen den FEN1 → FEN1 Compiler und fügen lediglich ein bisschen Code hinzu:

S-Attribut für Schachstellung -> Reihe / Reihe / Reihe / Reihe / Reihe / Reihe / Reihe / Reihe

```
$$ = $1 + $2 + $3 + $4 + $5 + $6 + $7 + $8 + $9 + $10 + $11 + $12 + $13 + $14 + $15;
return $$;
```

S-Attribut für Reihe -> Ziffer q1

```
$$ = "";
var n = parseInt($1);
for (var i = 0; i < n; i++) {
$$ = $$ + "e";
}
$$ = $$ + $2;
```

S-Attribut für Reihe -> Ziffer

```
$$ = "";
var n = parseInt($1);
for (var i = 0; i < n; i++) {
$$ = $$ + "e";
}
```

Somit ist der FEN1 → FENe Compiler fertig. Dieser gibt nun für ein gültiges Eingabewort `6r1/6pp/7r/1B5K/1P3k2/N7/3R4/8` folgende Ausgabe:
`eeeeeere/eeeeeepp/eeeeeeer/eBeeeeeK/ePeeekee/Neeeeeee/eeeReeee/eeeeeeee`

Computerübung 9.17

Folgen Sie den Anweisungen im Buch.
Das Endergebnis sieht folgendermaßen aus:

Computerübung 9.18

Keine Lösung.

Computerübung 9.19

Das vollständige T-Diagramm sieht folgendermaßen aus:

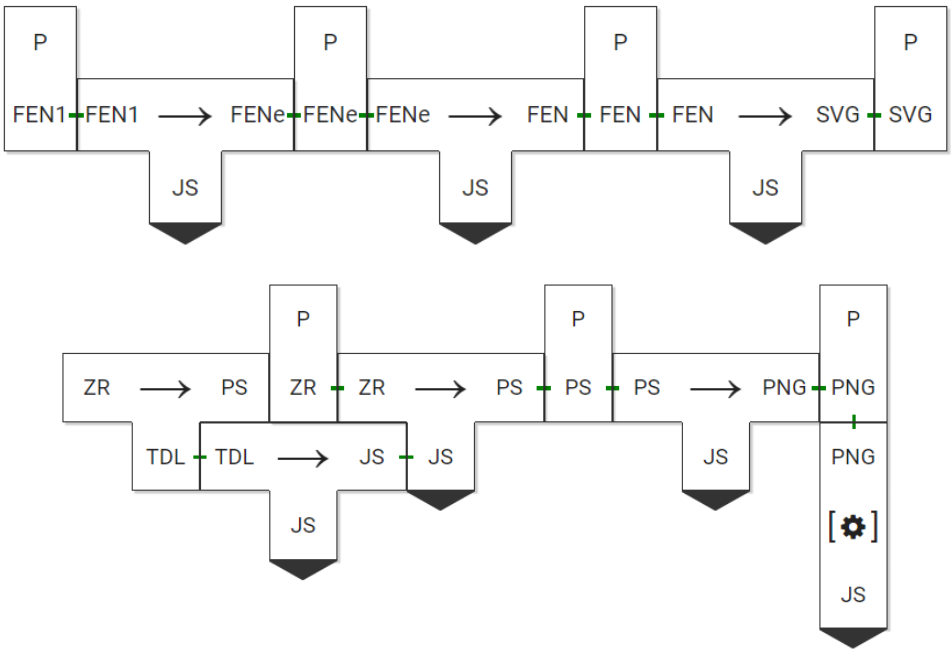


Abb. 46: Compilation und Interpretation von ZR → PNG

Computerübung 9.20

Zunächst fügen wir zwei neue Token hinzu. Zum einen das Token **GOTO** mit der Expression "*GOTO*" und zum anderen das Token **Komma** mit der Expression ",". Anschließend fügen wir die Regel der Form *Anweisung* → `GOTO Zahl Komma Zahl` hinzu und ergänzen in dieser folgenden Code:

```
$$ = $2 + " " + $4 + " goto";
```

Der Befehl GOTO kann nun genutzt werden.

Computerübung 9.21

Keine Lösung.

Computerübung 9.22

Zunächst erstellen wir folgende Grammatik:
$G = (N, T, P, s)$ mit

$$
\begin{aligned}
N &= \{Text, Woerter, Wort\},\\
T &= \{\text{Farbwort,anderesWort}\},\\
P &= \{Text \rightarrow Woerter\ ,\\
&\quad Woerter \rightarrow Wort\ Woerter \mid Wort\ ,\\
&\quad Wort \rightarrow \text{Farbwort} \mid \text{anderesWort}\ \},\\
s &= Text
\end{aligned}
$$

Weiterhin wird ein Compiler auf Basis der eben erstellten Grammatik erzeugt. Anschließend ändern wir die Tokens folgendermaßen:

Token: **Farbwort**
Expression: blau|rot|gruen|gelb|s

Token: **anderesWort**
Expression: [^\s]*|[\s]

Token: **IGNORE**
Expression: [\t\r\n\s]
Hinweis: diese Zeichen werden in der Eingabe ignoriert.

Zu guter Letzt fügen wir zu folgenden Regeln folgenden Code hinzu:

S-Attribut für Text -> Woerter

```
return($$);
```

S-Attribut für Wort -> Farbwort

```
var farbe = "";
switch($1){
  case "blau": farbe = "blue"; break;
  case "rot": farbe = "red"; break;
  case "gruen": farbe = "green"; break;
  case "gelb": farbe = "yellow"; break;
  case "schwarz": farbe = "black"; break;
  case "weiss": farbe = "white"; break;
}

$$ = " <font color=" + farbe + ">" + $1 + "</font> ";
```

S-Attribut für Wort -> anderesWort

```
$$ = $1 + " ";
```

Übung 9.3

Die wichtigsten Befehle von jsPDF, für die Zeichenrobotersprache, sind in Tabelle 5 aufgelistet.

Befehl	Parameter	Beschreibung
new jsPDF(orientation, unit, format)	orientation	Hochformat (p) oder Querformat (l)
	unit	Einheit in Punkten (pt), Millimeter (mm), Zentimeter (cm), Zoll (in)
	format	Seitenformat z.B.: a4
line(x1, y1, x2, y2)	x1	Start x-Koordinate
	y1	Start y-Koordinate
	x2	End x-Koordinate
	y2	End y-Koordinate
output(type)	type	mögliche Ausgabetypen z.B.: datauristring
setDrawColor(r, g, b)	r	Farbwert für rot (0-255)
	g	Farbwert für gruen (0-255)
	b	Farbwert für blau (0-255)
setLineWidth(width)	width	gibt die Breite der Linie an

Tabelle 5: Befehlsübersicht

Übung 9.4

Keine Lösung.

Computerübung 9.23

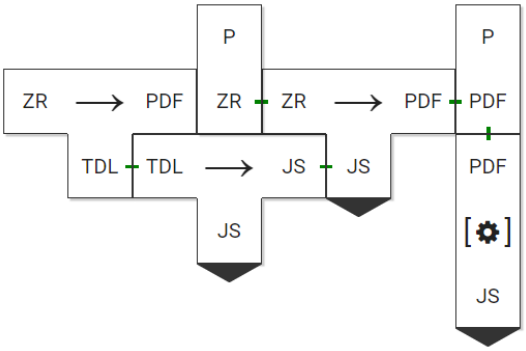


Abb. 47: ZR → PDF Compiler, inklusive Interpreter

Computerübung 9.24

Keine Lösung.

Computerübung 9.25

Keine Lösung.

Computerübung 9.26

Vergleichen Sie Ihre Lösung mit der aus der FLACI Beispielsammlung.

10 TURING-Maschine (TM) und CHOMSKY-Typ-0/1-Sprachen

Übung 10.1

Eingabewort	Konfigurationsfolge	Akzeptanz
$aacaaba$	$q_0aacaaba \vdash xq_1acaaba \vdash xxq_1caaba \vdash xq_2xxaaba$	akzeptiert
$abaaca$	$q_0abaaca \vdash xq_1baaca$	abgelehnt
aaa	$q_0aaa \vdash xq_1aa \vdash xxq_1a \vdash xxxq_1 \vdash xxxq_1 \vdash \cdots$	unentscheidbar

Tabelle 6: Konfigurationsfolgen der Wörter $aacaaba$, $abaaca$ und aaa

Übung 10.2

Die DTM M_1 aus Lehrbuch S. 251, Beispiel 10.1 akzeptiert alle Wörter, die mit dem regulären Ausdruck `a+c[abc]*` beschreiben werden können, da:

- ein Wort aufgrund der Regelmenge $\{\delta(q_0,k) = (q_j,m,b) | q_j \in Q; k,m \in \Gamma; b \in \{L,N,R\}\} = \{\delta(q_0,a) = (q_1,x,R)\}$ nur mit a beginnen kann
- aufgrund der Regel $\delta(q_1,a) = (q_1,x,R)$ beliebig viele weitere a folgen können
- die Maschine für ein anschließendes c in den Endzustand gelangt und dort crasht

Übung 10.3

Von allen Endzuständen $e_i \in E$ kann ein Übergang der Form $\delta(e_i,x) = (e,x,N)$ zu einem neuen Endzustand e führen, ohne den Lesekopf zu bewegen und x durch ein anderes Zeichen zu ersetzen.
Eine TM mit mehreren Endzuständen ist nicht mächtiger, da es zu jeder eine äquivalente TM mit einem Endzustand gibt. Nichtsdestotrotz liegt ein Vorteil darin, weniger Regeln definieren zu müssen.

Computerübung 10.1

Keine Lösung.

Computerübung 10.2

Der Kopf der Maschine "pendelt" zwischen dem ersten noch nicht gelesenen a und dem ersten noch nicht gelesenen c hin und her, solange ein a verfügbar ist. In solch einem Zyklus wird sichergestellt, dass ein a **mindestens** einen b-Partner hat, welcher wiederum **mindestens** einen c-Partner hat. Aus diesem Zyklus ausgetreten wird sichergestellt, dass ein a **genau** einen b-Partner hat, welcher wiederum **genau** einen c-Partner hat.

Übung 10.4

Beide Maschinen erweisen sich als funktionstüchtig und sind für "Extrem-Eingaben" wie ε oder I gewappnet.

Übung 10.5

Da die Zahlen in unärer Schreibweise notiert sind, reicht es bereits, das erste Eingabezeichen durch das Bandvorbelegungszeichen zu ersetzen (oder gar nicht zu verändern) und den Kopf nach rechts zu bewegen.

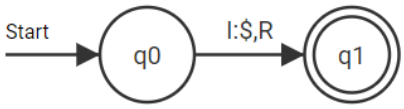


Abb. 48: Vorgänger einer Zahl in unärer Darstellung

Übung 10.6

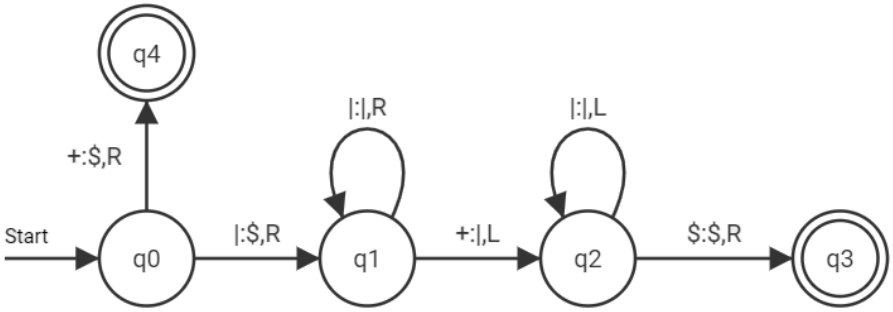


Abb. 49: Addition zweier Zahlen z, mit $z \in \mathbb{N}_0$

Übung 10.7

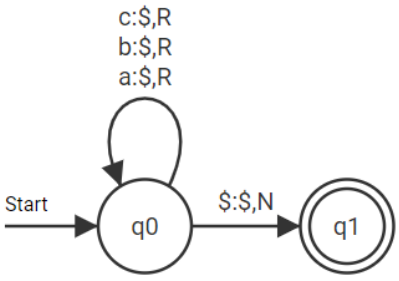


Abb. 50: "BandlöschTM

Übung 10.8

Bei dieser Aufgabe ist zu beachten, dass die TM auch auf mehrere b als Eingabezeichen vorbereitet sein muss. Würden wir auf q_2 verzichten, ergäbe sich eine neue Sprache L', mit $L' = L \cup L((aa)^*b[ab]^*)$, da die Maschine mit dem ersten b im Endzustand q_0 stoppen würde.

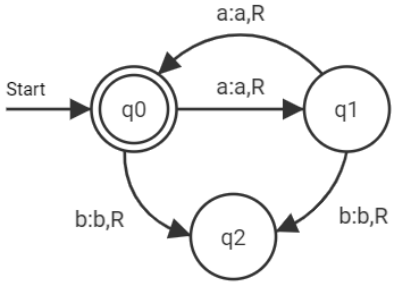


Abb. 51: TM als Akzeptor für $L = \{w \mid w \in \{a, b\}^* \text{ und } w = a^{2n} \mid n \geq 0\}$