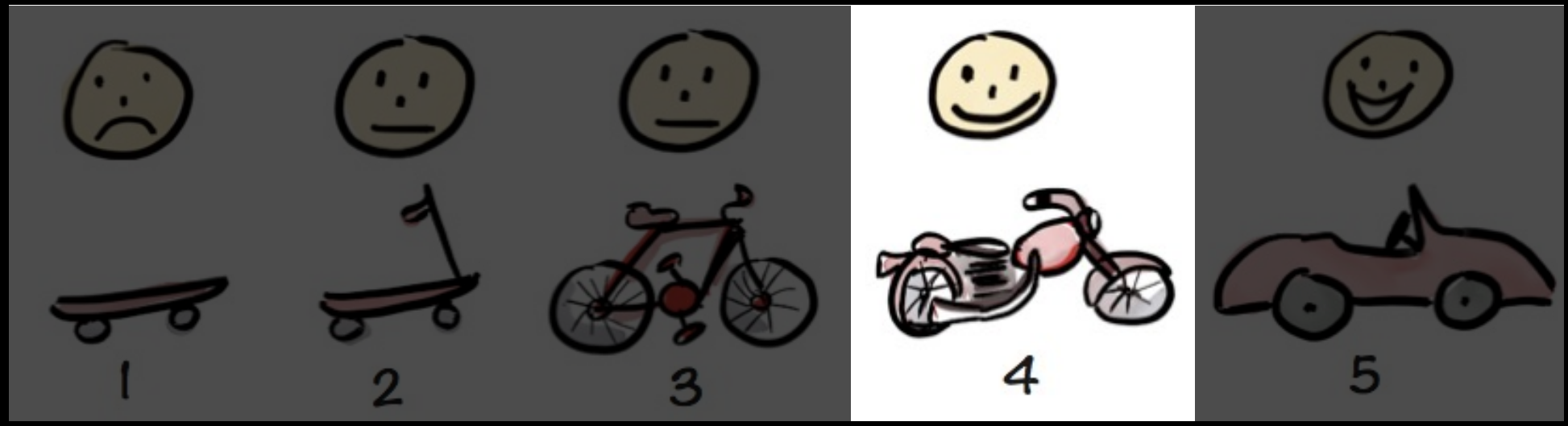
Errungenschaften der letzten Vorlesung

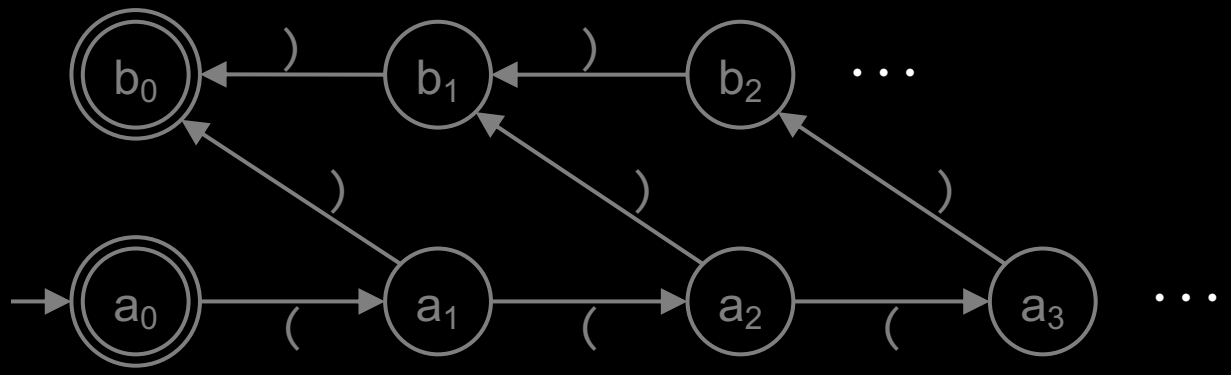


Die Syntax einer formalen Sprache wird mit formalen Grammatiken spezifiziert. Rechts- und linkslineare Grammatiken lassen sich mit endlichen Automaten verarbeiten. Für andere Grammatiken muss der Syntaxbaum noch manuell ermittelt werden. Alle Token, für die ein endlicher Akzeptor existiert, können automatisch zu einem Tokenizer / Lexer kombiniert werden. Zu jedem NEA kann mithilfe der Potenzmengenkonstruktion ein äquivalenter DEA berechnet werden. Zu jedem DEA kann ein minimaler DEA mit geringstmöglicher Zustandszahl berechnet werden.

Welche Sprachen sind nicht linear?

$$L = \{a^k b^k \mid k \in \mathbb{N}_0\} = \{\varepsilon, ab, aabb, aaabbb, \ldots\}$$

formaler Beweis in Meyer's Foliensatz Folie 46



Ein endlicher Automat kann nicht zählen. Er benötigt deshalb für das nächstlängere Wort jeweils zwei weitere Zustände. Weil der Parameter k hier nicht beschränkt ist, benötigt der Automat zur Erkennung der Sprache L unendlich viele Zustände. Wie der Name „endlicher Automat“ bereits suggeriert, hat ein endlicher Automat aber nur endlich viele Zustände. Er kann die Sprache L deshalb nicht erkennen. L ist demnach auch nicht linear.

Welche Sprachen sind nicht linear?

$$L = \{a^k b^k \mid k \in \mathbb{N}_0\} = \{\varepsilon, ab, aabb, aaabbb, \ldots\}$$

Vorsicht! Wir wissen bereits, dass die Sprache L nicht linear ist. Tatsächlich lässt sich die Sprache aber durch die nachfolgende Grammatik erzeugen:

```
a : '(' b ;     rechtslineare (Produktions-)Regel
a : ε ;         rechts- und linkslinear
b : a ')' ;     linkslinear
```

Eine lineare Grammatik ist entweder linkslinear oder rechtslinear. Eine rechtlineare Grammatik enthält ausschließlich rechtslineare Produktionen. Eine linkslineare enthält analog nur linkslineare Produktionen. Im Beispiel oben wurden Produktionen beider Typen verwendet. Die Grammatik ist deshalb kontextsensitiv und nicht linear.

context-free grammar (CFG)

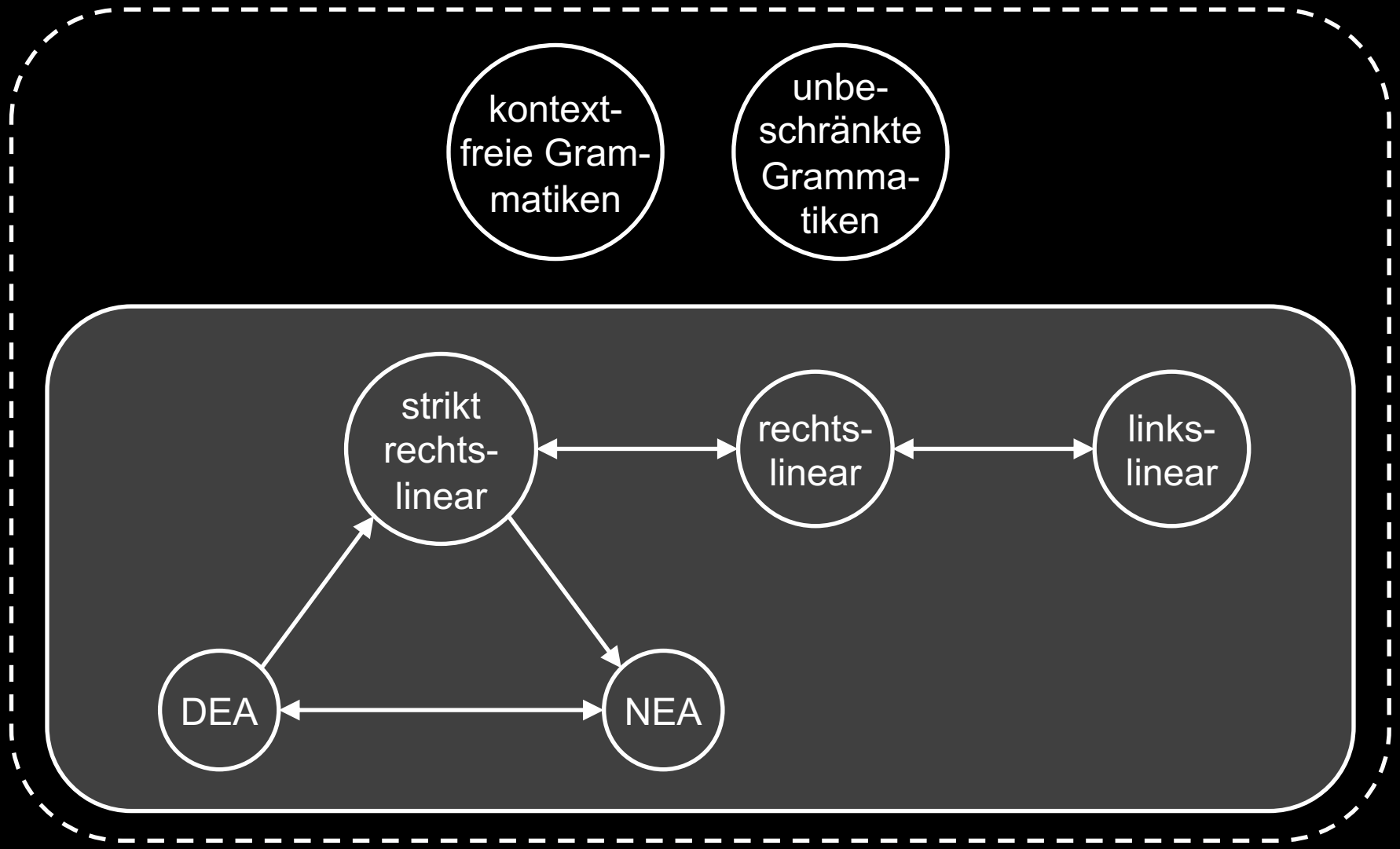
Eine kontextfreie Grammatik oder Typ-2-Grammatik ist eine Grammatik $G = (V_N, V_T, S, P)$, bei der alle Produktionen die folgende Form haben:

$$X \to w \qquad \text{mit } X \in N \text{ und } w \in (N \cup T)^*$$

Eine Sprache heißt kontextfrei, wenn Sie von einer kontextfreien Grammatik erzeugt werden kann.

Alle Regeln, die wir bislang allgemein für Grammatiken getroffen haben, gelten selbstverständlich auch für diese Untergruppe.

Modellübersicht



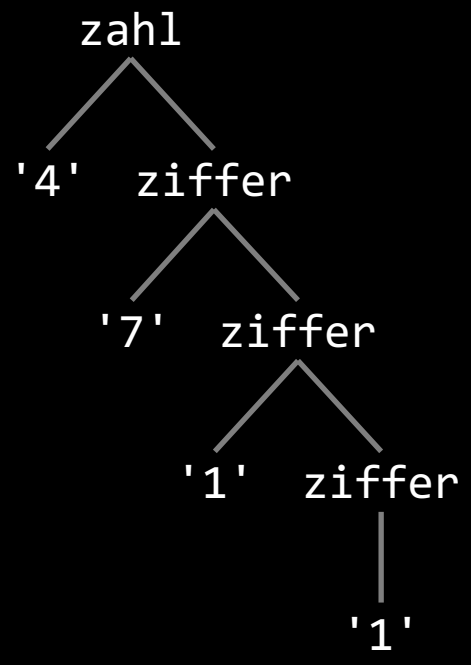
Unser Adventskalender

Typ	Name	Erlaubte Produktionen	Akzeptierender Automat	(Trennendes) Beispiel
3		$P \subseteq N \times (\{\varepsilon\} \cup T^* \cup T^*N)$ oder $P \subseteq N \times (\{\varepsilon\} \cup T^* \cup NT^*)$	endlicher Automat	$L = \{a^k b^l \mid k, l \in \mathbb{N}_0\}$
2	kontextfrei	$P \subseteq N \times (N \cup T)^*$		$L = \{a^k b^k \mid k \in \mathbb{N}_0\}$

Wiederholung

```
  zahl : '1' ziffer | '2' ziffer | … | '9' ziffer
       | '0' | '1' | '2' | … | '7' | '8' | '9' ;

ziffer : '0' ziffer | '1' ziffer | … | '9' ziffer
       | '0' | '1' | '2' | … | '7' | '8' | '9' ;
```



Syntaxbäume bei Typ-3 Sprachen sind immer listenartig

Wiederholung

```
e : e '+' e | e '-' e | e '*' e | e '/' e ;
e : '0' | '1' | '2' | … | '7' | '8' | '9' ;
```

$$\boldsymbol{e} \overset{1}{\Rightarrow} \boldsymbol{e} + e \overset{7}{\Rightarrow} 2 + \boldsymbol{e} \overset{3}{\Rightarrow} 2 + \boldsymbol{e} * e \overset{8}{\Rightarrow} 2 + 3 * \boldsymbol{e} \overset{9}{\Rightarrow} 2 + 3 * 4$$

$$\boldsymbol{e} \overset{1}{\Rightarrow} e + \boldsymbol{e} \overset{3}{\Rightarrow} e + e * \boldsymbol{e} \overset{9}{\Rightarrow} e + \boldsymbol{e} * 4 \overset{8}{\Rightarrow} \boldsymbol{e} + 3 * 4 \overset{7}{\Rightarrow} 2 + 3 * 4$$

Bei Typ-2 Sprachen gibt es immer

eine Linksableitung und

eine Rechtsableitung.

Ableitung

Sei $G = (V_N, V_T, S, P)$ eine kontextfreie Grammatik mit $X \to w \in P$ und $u, u' \in (N \cup T)^*$.

Falls $u \in T^*$, heißt ein Ableitungsschritt $uXu' \Rightarrow uwu'$ mit Produktion $X \to w \in P$ Linksableitungs-schritt, geschrieben $uXu' \overset{l}{\Rightarrow} uwu'$.

$w_0 \Rightarrow w_1 \Rightarrow \cdots \Rightarrow w_k$ heißt Linksableitung(sfolge), falls $w_0 \overset{l}{\Rightarrow} w_1 \overset{l}{\Rightarrow} \ldots \overset{l}{\Rightarrow} w_k$, geschrieben $w_0 \overset{l}{\Rightarrow}_* w_k$.

Analog: Rechtsableitungsschritt $\overset{r}{\Rightarrow}$ und Rechtsableitung $\overset{l}{\Rightarrow}_*$.

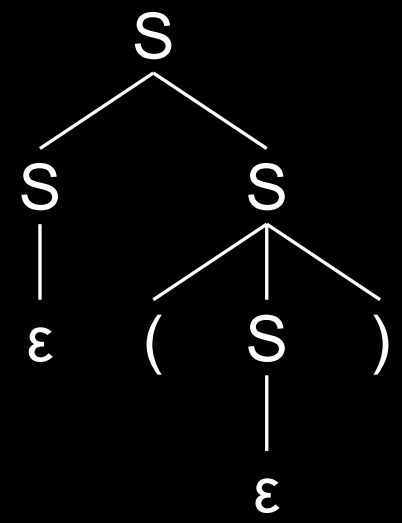
example: derivation

$G = (\{S\}, \{(,)\}, S, \{S \to \varepsilon | (S) | SS\})$

Linksableitung: S ⇒ **S**S ⇒ **S** ⇒ (**S**) ⇒ ()

Rechtsableitung: S ⇒ S**S** ⇒ S(**S**) ⇒ **S**() ⇒ ()

Weder noch: S ⇒ S**S** ⇒ **S**(S) ⇒ (**S**) ⇒ ()



Alle hier gezeigten Ableitungen liefern den gleichen Ableitungsbaum.

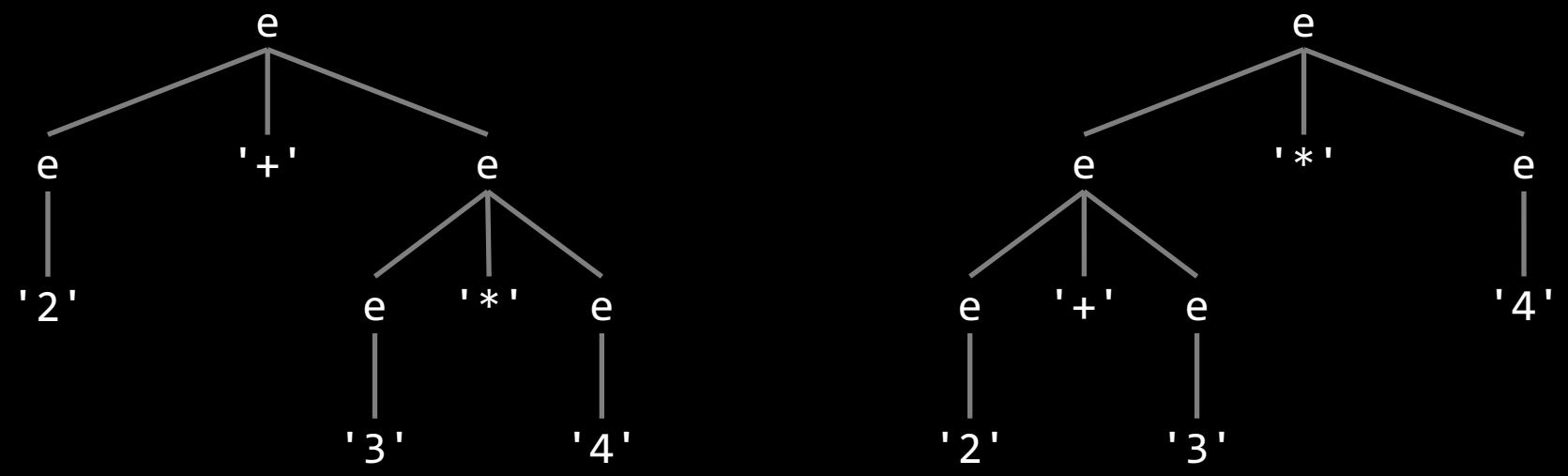
Ein Ableitungsbaum ist demnach „allgemeiner“ als eine Ableitung.

Aus jedem Ableitungsbaum ergibt sich genau eine Linksableitung und analog genau eine Rechtsableitung.

Also: Zu jeder Ableitung gibt es eine „äquivalente“ Links- bzw. Rechtsableitung.

Wiederholung

```
e : e '+' e | e '-' e | e '*' e | e '/' e ;
e : '0' | '1' | '2' | … | '7' | '8' | '9' ;
```



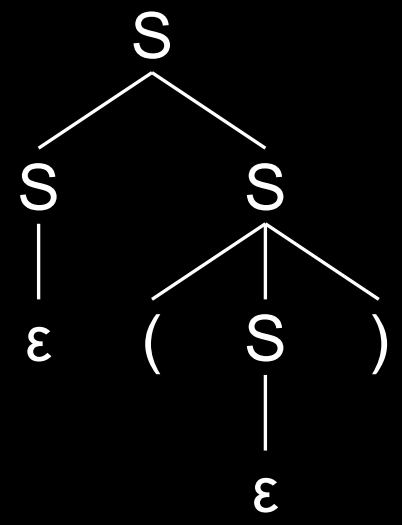
Eine Grammatik heißt **eindeutig**, wenn es für jedes Wort $w \in L(G)$ genau eine Linksableitung gibt. Nicht eindeutige Grammatiken nennt man auch **mehrdeutig**.

Mehrdeutigkeit

Eine Typ-2-Grammatik ist mehrdeutig, falls es für ein Wort zwei verschiedene Ableitungsbäume (Linksableitungen, Rechtsableitungen) gibt.

Sonst heißt die Grammatik eindeutig.

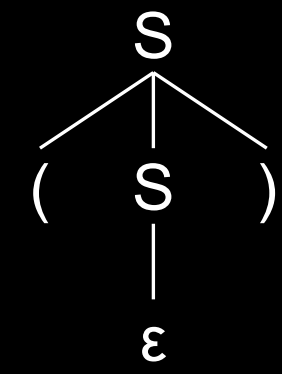
Eine kontextfreie Sprache ist (inhärent) mehrdeutig, falls jede sie erzeugende T2G mehrdeutig ist.



Die Grammatik $G = (\{S\}, \{(,)\}, S, \{S \to \varepsilon | (S) | SS\})$ ist mehrdeutig.

Die erzeugte formale Sprache ist aber nicht inhärent mehrdeutig.

Eine eindeutige Grammatik G' mit $L(G) = L(G')$ wäre
$G' = (\{S, T\}, \{(,)\}, S, \{S \to \varepsilon | ST, T \to (S)\})$



Offene Fragen

1. ~~Nicht jedes Token darf an jeder Stelle stehen. Wie lässt sich das regulieren?~~
2. ~~Sind die von endlichen Automaten akzeptierten Sprachen identisch zu den von Grammatiken erzeugten Sprachen?~~
3. ~~Lassen sich alle NEAs in DEAs umwandeln?~~
4. ~~Sind rechtslineare und linkslineare Grammatiken verschieden?~~
5. ~~Sind unbeschränkte Grammatiken mächtiger als die linearen?~~
6. ~~Gibt es “kompaktere” Automaten?~~
7. Kann man die Erstellung eines endlichen Automaten automatisieren?

Wie sieht unsere gegenwärtige Implementierung aus?

Beispiel: StateMachineTableNumber.java

```
@Override
public void initStateTable() {
        compiler.State firstDigit = new compiler.State("firstDigit");
        firstDigit.addTransition('0', "number0");
        firstDigit.addTransition('1', "nextDigit");
        firstDigit.addTransition('2', "nextDigit");
        firstDigit.addTransition('3', "nextDigit");
        firstDigit.addTransition('4', "nextDigit");
        firstDigit.addTransition('5', "nextDigit");
        firstDigit.addTransition('6', "nextDigit");
        firstDigit.addTransition('7', "nextDigit");
        firstDigit.addTransition('8', "nextDigit");
        firstDigit.addTransition('9', "nextDigit");
        m_stateMap.put("firstDigit", firstDigit);

        compiler.State number0 = new compiler.State("number0");
        m_stateMap.put("number0", number0);
                                                    }
        compiler.State nextDigit = new compiler.State("nextDigit");
        nextDigit.addTransition('0', "nextDigit");
        nextDigit.addTransition('1', "nextDigit");
        nextDigit.addTransition('2', "nextDigit");
        nextDigit.addTransition('3', "nextDigit");
        nextDigit.addTransition('4', "nextDigit");
        nextDigit.addTransition('5', "nextDigit");
        nextDigit.addTransition('6', "nextDigit");
        nextDigit.addTransition('7', "nextDigit");
        nextDigit.addTransition('8', "nextDigit");
        nextDigit.addTransition('9', "nextDigit");
        m_stateMap.put("nextDigit", nextDigit);
```