# Lektion 17

In dieser Lektion werden zunächst Kellerautomaten eingeführt, eine Erweiterung von nichtdeterministischen Automaten mit einem Stack (Kellerspeicher) als Speichermöglichkeit. Dann wird der Zusammenhang zwischen Kellerautomaten und kontextfreien Grammatiken betrachtet. So wie DEAs bzw. ε-NEAs auf Automaten-Seite gewissermaßen das Gegenstück zu regulären Ausdrücken sind, so sind Kellerautomaten das Gegenstück zu den kontextfreien Grammatiken, d.h. beide können die gleiche Klasse von Sprachen beschreiben. Wir betrachten dazu, wie zu jeder kontextfreien Grammatik ein Kellerautomaten definiert werden kann, der genau die Sprache der Grammatik akzeptiert. Dabei wird dar Ansatz der sog. Top-Down-Syntaxanalyse verwendet, der später weiterverwendet wird.

# Kap. 10 Kellerautomaten

## *Inhalt*

- Nichtdeterministische Kellerautomaten
- Zusammenhang zwischen Kellerautomaten und kontextfreien Grammatiken
- Deterministische Kellerautomaten

## *Vorbemerkung*

Mit DEAs und ε-NEAs haben Sie zwei Arten von endlichen Automaten kennengelernt, die zur Beschreibung regulärer Sprachen verwendet werden können. Eine Gemeinsamkeit dieser Automatenarten ist, dass sie nur in sehr begrenzter Weise Informationen speichern können - letztendlich nur dadurch, dass sie sich in unterschiedlichen Zuständen befinden können - und die Zustandszahl ist endlich.

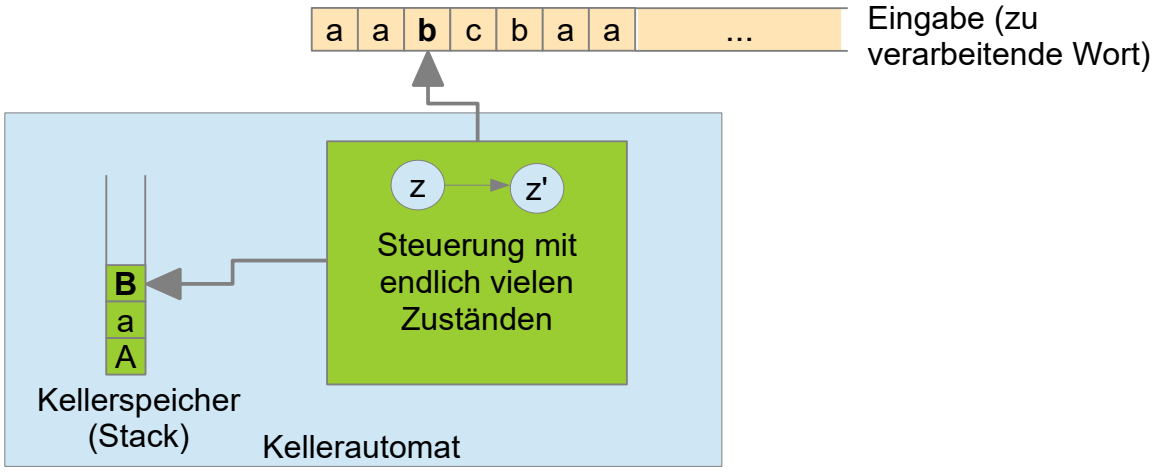
Sprachen, wie z.B. $L = \{a^n b^n \mid n \geq 0\}$, können von solchen Automaten nicht akzeptiert werden, da (etwas salopp formuliert) ein beliebig großes *n* gespeichert werden müsste. In diesem Kapitel lernen Sie eine weitere Art von Automaten kennen, die sog. *Kellerautomaten*, die eine unbegrenzte Speichermöglichkeit besitzen.

Diese Automaten haben zwar weiterhin nur eine endliche Anzahl von Zuständen, aber sie sind mit einem *Stack* (*Kellerspeicher*) ausgestattet, auf dem beliebig viele Zeichen abgelegt werden können. Wie Sie sehen werden, ist es beispielsweise kein Problem, einen Kellerautomaten für die nicht reguläre Sprache $L = \{a^n b^n \mid n \geq 0\}$ zu definieren.

Sie werden auch sehen, dass ein enger Zusammenhang zwischen kontextfreien

Grammatiken und Kellerautomaten besteht. Zunächst ist dieser Bezug eher theoretischer Natur. Im folgenden Kapitel werden wir dann aber auch betrachten, wie die Idee der Kellerautomaten ein wenig weiterentwickelt werden kann, um zu einem praktisch anwendbaren, effizienten Verfahren für die Syntaxanalyse bei kontextfreien Grammatiken zu kommen.

*Abbildung 10.1 - Prinzipieller Aufbau eines Kellerautomaten*




- Der Kellerspeicher (Stack) kann eine unbegrenzte Folge von Einträgen (Kellersymbolen) speichern.
- Die Steuerung hat endlich viele Zustände. Die Steuerung legt den nächsten Ausführungsschritt fest, abhängig von:
  - aktuellem Zustand,
  - oberstem Symbol auf dem Keller und
  - ggf. nächstem Symbol der Eingabe. ε-Schritte unabhängig von der Eingabe sind auch möglich.
- Ein Ausführungsschritt bewirkt, dass
  - in einen Nachfolgezustand gewechselt wird,
  - zum nächsten Symbol der Eingabe weiter gegangen wird, sofern es kein ε-Schritt ist, und
  - der oberste Kellereintrag entfernt und durch eine Folge neuer Einträge (die auch leer sein kann) ersetzt wird.

## 10.1 Nichtdeterministische Kellerautomaten

Das Verhalten eines Kellerautomaten ist üblicherweise nichtdeterministisch, aber es gibt auch eine deterministische Variante (siehe später). Die englische Bezeichnung für Kellerautomaten lautet *pushdown automata (PDA)*.

## *Definition 10.2 (nichtdeterministischer) Kellerautomat*

Ein (nichtdeterministischer) **Kellerautomat** $K = (Z, \Sigma, \Gamma, \Delta, z_0, k_0, E)$ ist definiert durch:

| | |
|---|---|
| $Z$ | endliche Menge von *Zuständen* |
| $\Sigma$ | endliches *Eingabealphabet* |
| $\Gamma$ | endliches *Kelleralphabet* |
| $\Delta$ | *Zustandsübergangsrelation*, eine endliche Teilmenge von $Z \times \Gamma \times (\Sigma \cup \{\varepsilon\}) \times Z \times \Gamma^*$ |
| $z_0 \in Z$ | *Startzustand* |
| $k_0 \in \Gamma$ | *Kellerstartsymbol* |
| $E \subseteq Z$ | Menge der *Endzustände* (akzeptierende Zustände) |

- Das Kelleralphabet legt fest, welche Symbole auf dem Keller abgelegt werden können. Das Kelleralphabet kann sich mit dem Eingabealphabet überschneiden, kann aber auch völlig unabhängig davon sein.
- Am Anfang liegt immer ein speziell ausgewähltes Symbol auf dem Keller, das sog. *Kellerstartsymbol*.

Die Zustandsübergangsrelation wird nachfolgend genauer erläutert.

## *Notation 10.3 für Zustandsübergangsrelation*

Statt $(z, k, x, z', k_1...k_n) \in \Delta$ notieren wir (wobei $x = \varepsilon$ oder $x \in \Sigma$):

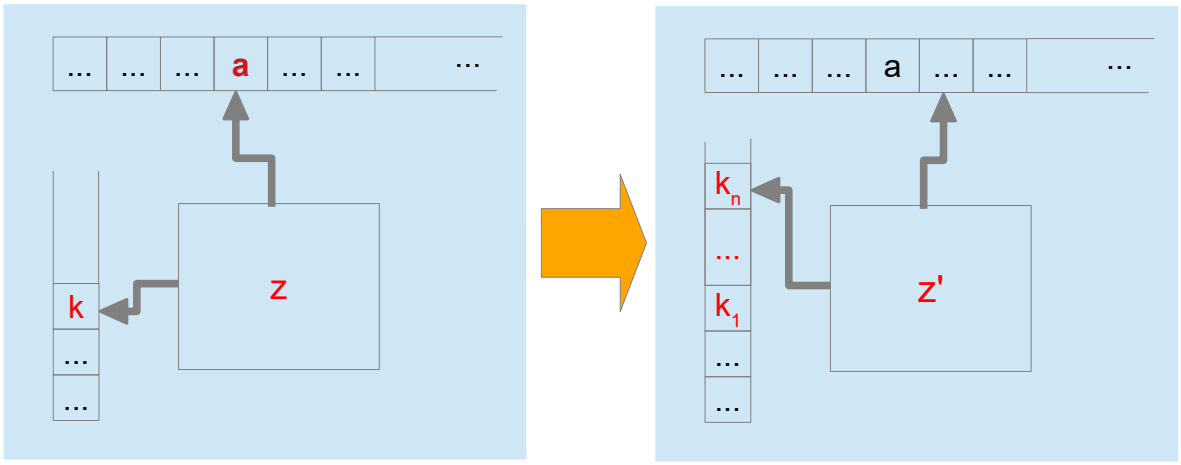
$$z, k \xrightarrow{x} z', k_1...k_n$$

(1) Ist der mögliche Zustandsübergang mit einem Zeichen $a \in \Sigma$ markiert

$$z, k \xrightarrow{a} z', k_1...k_n$$

dann bedeutet das: Wenn der Kellerautomat im Zustand $z$ ist, das Kellersymbol $k$ oben auf dem Keller liegt und $a$ das aktuelle Symbol in der Eingabe ist, dann

- geht der Kellerautomat in den Nachfolgezustand $z'$,
- die Eingabeposition geht zum nächsten Zeichen des Eingabeworts,
- das oberste Symbol $k$ wird vom Keller entfernt und die Kellersymbole $k_1...k_n$ werden oben auf dem Keller abgelegt (so dass dann $k_n$ ganz oben auf dem Keller ist).

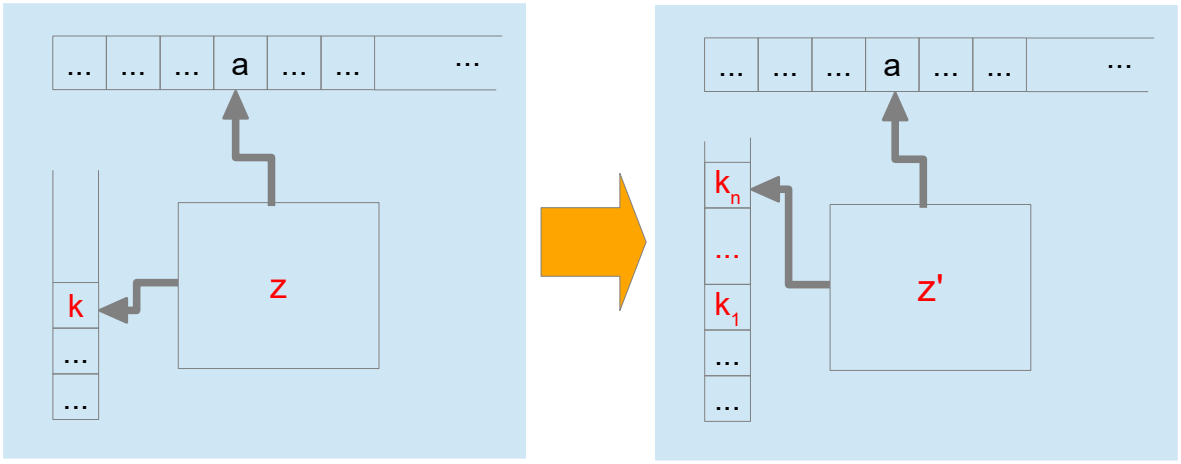



(2) Ist der Zustandsübergang mit ε markiert:

$$z, k \xrightarrow{\varepsilon} z', k_1 \ldots k_n$$

dann bedeutet das: Wenn der Kellerautomat im Zustand *z* ist, das Kellersymbol *k* oben auf dem Keller liegt, dann

- gehe der Kellerautomat in den Nachfolgezustand *z'*,
- die Eingabeposition bleibt unverändert,
- das oberste Symbol *k* wird vom Keller entfernt und die Kellersymbole $k_1 \ldots k_n$ werden oben auf dem Keller abgelegt.




## *Anmerkung*

Einige Grundoperationen im Umfang mit einem Stack können folgendermaßen formuliert werden:

- Mit $z, \boldsymbol{k} \xrightarrow{x} z', \boldsymbol{k}$ wird beschrieben, dass der Keller unverändert bleibt: k wird (wie immer) vom Keller entfernt und gleich wieder darauf abgelegt.
- Mit $z, \boldsymbol{k} \xrightarrow{x} z', \boldsymbol{ka}$ wird gewissermaßen eine push(a)-Operation beschrieben, die a auf dem Stack zusätzlich ablegt: Das oberste Symbol *k* wird

zunächst entfernt und dann zusammen mit *a* wieder abgelegt.

- Mit z, $k \xrightarrow{x}$ z', ε wird gewissermaßen eine pop()-Operation beschrieben, bei der der oberste Kellereintrag entnommen wird (d.h. durch die leere Folge ε ersetzt wird).

## 10.1.1 Verarbeitung eines Eingabeworts

Um formal definieren zu können, wir der Kellerautomat ein Eingabewort verarbeitet, wird der Begriff der *Ausführungskonfiguration* verwendet. Eine Ausführungskonfiguration beschreibt den aktuellen Status bei der Ausführung:

- In welchem Zustand befindet sich der Automat?
- Welche Folge von Kellersymbolen liegt auf dem Keller?
- Welcher Rest des Eingabeworts ist noch zu verarbeiten?

### *Definition 10.4 - Ausführungskonfiguration*

Eine **Konfiguration** eines Kellerautomaten $K = (Z, \Sigma, \Gamma, \Delta, z_0, k_0, E)$ ist ein Tripel

$$(z,\ k_1...k_m,\ a_1...a_n),$$

wobei

| | |
|---|---|
| $z \in Z$ | aktueller Zustand, |
| $k_1...k_m \in \Gamma^*$ | aktueller Kellerinhalt ($k_m$ ist oberster Eintrag) |
| $a_1...a_n \in \Sigma^*$ | restliches Eingabewort ($a_1$ ist aktuelles Zeichen) |

Einzelne Ausführungsschritte eines Kellerautomaten, so wie oben schon bildlich dargestellt, können nun in folgender Weise formal definiert werden:

### *Definition 10.5 - Nachfolgerelation für Konfigurationen*

(z', W', w') ist mögliche direkte **Nachfolgekonfiguration** von (z, W, w), notiert

$$(z, W, w) \rightarrow (z', W', w'),$$

wobei $z \in Z$ , $z' \in Z$ Zustände sind, $W \in \Gamma^*$, $W' \in \Gamma^*$ jeweils der Kellerinhalt und $w \in \Sigma^*$, $w' \in \Sigma^*$ die restliche Eingabe ist, in folgenden Fällen:

(1) $(z, Wk, aw) \rightarrow (z', Wk_1...k_n, w)$

falls in der Zustandsübergangsrelation Δ folgender Übergang für das Zeichen a enthalten ist:

$$z, k \xrightarrow{a} z', k_1...k_n$$

(2) $(z, Wk, w) \rightarrow (z', Wk_1...k_n, w)$

falls in der Zustandsübergangsrelation $\Delta$ folgender $\varepsilon$-Übergang enthalten ist:

$$z, k \xrightarrow{\varepsilon} z', k_1...k_n$$

### *Anmerkung*

- Ein Kellerautomat kann *nichtdeterministisch* sein: zu einer Konfiguration kann es mehrere mögliche Nachfolgekonfigurationen geben.
- Bei deterministischen Kellerautomaten darf es zu eine Konfiguration immer nur höchstens eine direkte Nachfolgekonfiguration geben.

## 10.1.2 Sprache eines Kellerautomaten

Welche Wörter werden von einem Kellerautomaten akzeptiert? Dies ist in folgender Weise definiert.

### *Definition 10.6 - Akzeptierte Sprache eines Kellerautomaten*

- Der Kellerautomat $K = (Z, \Sigma, \Gamma, \Delta, z_0, k_0, E)$ akzeptiert ein Eingabewort $w \in \Sigma^*$, wenn ausgehend von der Startkonfiguration
  $(z_0, k_0, w)$
  eine Konfiguration
  $(z_e, W, \varepsilon)$
  in endlich vielen Ausführungsschritten erreichbar ist mit
  - einem **Endzustand** $z_e \in E$,
  - einem beliebigen Kellerinhalt $W \in \Gamma^*$ und
  - der **leeren Eingabe.**
- Die vom Kellerautomat $K$ **akzeptierte Sprache** $L(K)$ ist
  $L(K) = \{ w \in \Sigma^* \mid K \text{ akzeptiert } w \}$.

### *Anmerkung*

- Um zu prüfen, ob das Wort $w$ akzeptiert wird, beginnt man mit der Startkonfiguration $(z_0, k_0, w)$, bei der der Automat im Startzustand $z_0$ ist, auf dem Keller nur das Kellerstartsymbol $k_0$ liegt und als Eingabe noch das komplette Wort $w$ zu verarbeiten ist.
- Wenn es möglich ist, in endlich vielen Schritten eine erfolgreiche Endkonfiguration zu erreichen, wird das Wort akzeptiert. Eine Konfiguration (z, W, w') ist dann erfolgreich, wenn der Zustand $z$ ein Endzustand ist und wenn das Eingabewort komplett abgearbeitet wurde,d.h. w' = $\varepsilon$. Der Inhalt $W$

des Kellers ist egal.

- In der Literatur finden Sie teilweise eine Variante von Kellerautomaten, bei welchen Wörter nur akzeptiert werden, wenn auch der Keller an Ende leer geworden ist. Dies macht aber keinen wesentlichen Unterschied: Es kann immer die eine Variante von Kellerautomaten auch in die andere Variante "übersetzt" werden.

### *Beispiel 10.7 - Kellerautomat für Palindrome*

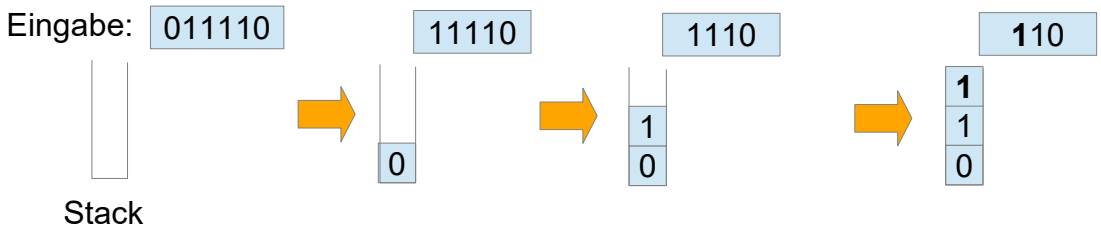
Es kann ein Kellerautomat definiert werden, der die Sprache aller Palindrome über dem Alphabet $\Sigma = \{0,1\}$ akzeptiert (dies ist eine nicht reguläre Sprache).

Die Grundidee dafür ist folgende:

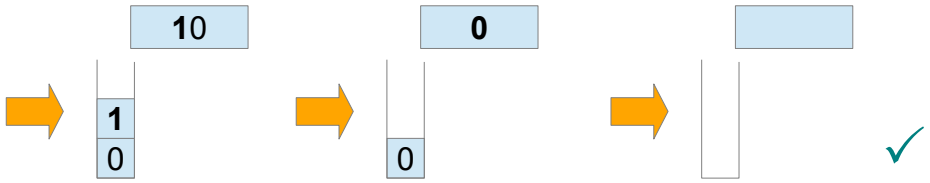
- In einer ersten Phase ("Aufbauphase") wird nach und nach die erste Hälfte des Worts auf dem Kellerspeicher abgelegt.
- In der zweiten Phase ("Abbauphase") wird dann immer das nächste Eingabesymbol mit dem obersten Symbol auf dem Keller verglichen. Beide müssen übereinstimmen. Wenn ja, dann wird das Symbol oben vom Keller entnommen und zum nächsten Symbol der Eingabe weitergegangen.
- Am Ende müssen alle abgelegten Zeichen auch wieder vom Keller entfernt worden und alle Zeichen der Eingabe verarbeitet worden sein.

Veranschaulichung: Prüfung des Palindroms 011110

- "Aufbauphase":




- "Abbauphase":




Es ist dabei zu berücksichtigen, dass ein Palindrom auch eine ungerade Länge haben kann, z.B. 00100. Dann ist das mittlere Zeichen beim Wechsel zwischen Aufbau- und Abbauphase zu überlesen.

Es stellt sich nun die Frage, wie der Kellerautomat erkennen kann, wann die Mitte

des Eingabeworts erreicht ist und ob es gerade oder ungerade Länge hat, so dass das mittlere Zeichen ggf. überlesen werden muss. Die Antwort darauf ist im ersten Moment überraschend: Da der Kellerautomat höchstens das nächste Zeichen "sieht", kann er nicht erkennen wie lange ein Wort ist oder ob die Mitte des Worts erreicht ist. Der Kellerautomat muss es aber auch nicht erkennen können! Hier kommt uns der Nichtdeterminismus zur Hilfe.

In der Aufbauphase hat der Automat immer nichtdeterministisch drei Möglichkeiten:

- Nächstes Zeichen auf den Keller legen
- Direkt in die Abbau-Phase wechseln, ohne ein Zeichen zu überlesen (für gerade Gesamtlänge)
- In die Abbau-Phase wechseln, aber aktuelles Zeichen überlesen (für ungerade Gesamtlänge).

Sofern das Eingabewort ein Palindrom ist, gibt es immer eine Möglichkeit richtig zu "raten" und dann zu einer erfolgreichen Endkonfiguration zu kommen. Ist das Eingabewort kein Palindrom, führt keine der Möglichkeiten zu einer erfolgreichen Endkonfiguration.

Diese Grundidee kann nun in folgender Weise formalisiert werden:

**Kellerautomat für Palindrome**

- Eingabealphabet $\Sigma = \{0,1\}$
- Kelleralphabet $\Gamma = \{0,1,k_0\}$
- Kellerstartsymbol: $k_0$
- Zustandsmenge $Z = \{z_0, z_1, z_e\}$
  $z_0$ entspricht "Aufbauphase"
  $z_1$ entspricht "Abbauphase"
- Startzustand: $z_0$
- Endzustände: $E = \{z_e\}$
- Zustandsübergangsrelation ($k \in \Gamma$ steht für ein beliebiges Kellersymbol)

(1) $z_0, k \xrightarrow{0} z_0, k\,0$ // "Aufbauphase": Eingabesymbol 0 auf Stack legen

(2) $z_0, k \xrightarrow{1} z_0, k\,1$ // "Aufbauphase": Eingabesymbol 1 auf Stack legen

(3) $z_0, k \xrightarrow{0} z_1, k$ // Wechsel in Abbauphase (ungerade Länge)
// "mittleres" Zeichen 0 wird überlesen.
// Stackinhalt bleibt unverändert (oberstes Symbol $k$
// wird entnommen und wieder zurückgelegt)

(4) $z_0, k \xrightarrow{1} z_1, k$ // Wechsel in Abbauphase (für ungerade Länge)
// "mittleres" Zeichen 1 wird überlesen.

// Stackinhalt bleibt unverändert

(5) $z_0, k \xrightarrow{\varepsilon} z_1, k$ // Wechsel in Abbauphase (für gerade Länge),
// Eingabeposition bleibt, Stack bleibt unverändert

(6) $z_1, 0 \xrightarrow{0} z_1, \varepsilon$ // "Abbauphase": 0 oben auf Stack und in
// Eingabe, 0 auf Stack wird gelöscht

(7) $z_1, 1 \xrightarrow{1} z_1, \varepsilon$ // "Abbauphase": 1 oben auf Stack und in
// Eingabe, 1 auf Stack wird gelöscht

(8) $z_1, k_0 \xrightarrow{\varepsilon} z_e, \varepsilon$ // Wechsel in Endzustand möglich, wenn alle
// abgelegten Symbole wieder entfernt wurden
// und das Kellerstartsymbol unten wieder
// sichtbar wird

## *Beispiel 10.8 - Verarbeitung des Worts 00100*

Gegeben sei der Kellerautomat für Palindrome s.o.

Folgende Ausführungsschritte zeigen, dass das Wort 00100 akzeptiert wird. Bei jedem Schritt in der Aufbauphase wird dabei passend "geraten", so dass am Ende eine erfolgreiche Endkonfiguration erreicht wird. Sobald der Automat im Zustand $z_1$ ist, d.h. in der Abbauphase, ist das weitere Verhalten deterministisch.

Mögliche Folge von Konfigurationen:

| | Zustand | Keller | Eingabe | |
|---|---|---|---|---|
| | (**$z_0$**, | **$k_0$**, | **0**0100) | // Startkonfiguration |
| → | (**$z_0$**, | $k_0$**0**, | **0**100) | // (1), 0 auf Stack legen |
| → | (**$z_0$**, | $k_0$0**0**, | 100) | // (1), 0 auf Stack legen |
| → | (**$z_1$**, | $k_0$0**0**, | **0**0) | // (4), "mittlere" 1 überlesen |
| → | ($z_1$, | $k_0$0, | 0) | // (6), 0 auf Stack und in Eingabe |
| → | ($z_1$, | $k_0$, | ε) | // (6), 0 auf Stack und in Eingabe |
| → | (**$z_e$**, | ε, | **ε**) | // (8), alle abgelegten Zeichen<br>// abgebaut, Kellerstartsymbol<br>// wieder oben, in Endzustand<br>// wechseln |

Es wird am Ende eine erfolgreiche Konfiguration mit Endzustand $z_e$ und leerer Eingabe erreicht. Somit wird das Wort akzeptiert.

## *Aufgabe 10.9 - Verarbeitung des Worts 1001*

Zeigen Sie, dass der Kellerautomat für Palindrome s.o. auch das Wort 1001 akzeptiert.

### *Aufgabe 10.10 - Kellerautomat für $a^n b^n$*

1. Definieren Sie einen Kellerautomaten, der die Sprache L = $\{a^n b^n \mid n \geq 0\}$ akzeptiert.
2. Zeigen Sie, dass Ihr Kellerautomat das Wort aaabbb akzeptiert, indem Sie eine Ausführungsfolge angeben, die mit einer erfolgreichen Konfiguration endet.

## 10.2 Kellerautomaten und kontextfreie Grammatiken

### *Satz 10.11 - Kellerautomaten und kontextfreie Sprachen*

Die von ***nichtdeterministischen* Kellerautomaten** akzeptierten Sprachen stimmen genau mit den durch kontextfreien Grammatiken darstellbaren Sprachen überein (genannt "**kontextfreie Sprachen**").

**Begründung**

- Zu jeder kontextfreien Grammatik kann ein Kellerautomat konstruiert werden, der die Sprache der Grammatik akzeptiert (siehe folgende Konstruktion 10.14).
- Zu jedem Kellerautomaten kann (sehr aufwendig) eine äquivalente Grammatik konstruierte werden, die genau die vom Kellerautomaten akzeptierte Sprache generiert (siehe z.B. Hopcroft, Motwani, Ullmann: Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie, 2. Auflage, Abschnitt 6.3.2)

Ein direkter Zusammenhang zwischen Kellerautomaten und kontextfreien Grammatiken ist im ersten Moment nicht erkennbar. Wir betrachten deshalb zunächst mit der sog. *Top-Down-Syntaxanalyse* einen prinzipiellen Ansatz, wie Syntaxanalyse für kontextfreie Grammatiken angegangen werden könnte, d.h. wie man prüfen kann, ob ein Wort zur Sprache einer Grammatik gehört. Dabei wird versucht, ausgehend vom Startsymbol (Wurzel des Syntaxbaums) nach und nach einen vollständigen Syntaxbaum für das Wort zu erstellen. Später werden Sie dann sehen, dass sich dieser Ansatz auch mittels Kellerautomaten einfach umsetzen lässt.

### 10.2.1 Prinzip der Top-Down-Syntaxanalyse

### *Beispiel 10.12 - Top-Down-Syntaxanalyse*

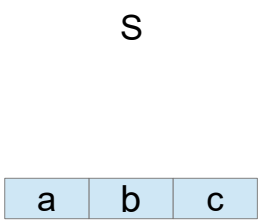
Gegeben sei folgende Grammatik:

$$S \rightarrow aA \mid Ac$$
$$A \rightarrow aAb \mid \varepsilon$$

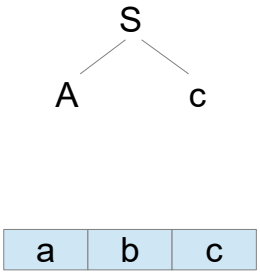
Wie kann man sich überlegen, ob ein Wort, z.B. *abc*, zur Sprache der Grammatik gehört?

Eine naheliegende Herangehensweise ist die, zu versuchen, nach und nach, ausgehend vom Startsymbol, also "top-down", einen Syntaxbaum für das Wort zu erstellen:

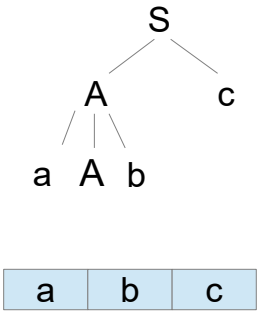
(1) Man beginnt mit dem Startsymbol S als Wurzel des Syntaxbaum. Es ist noch das komplette Wort **abc** zu analysieren.



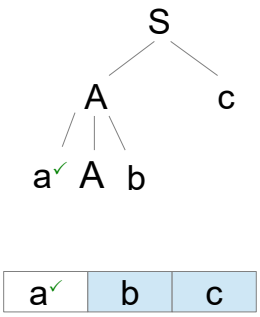

(2) Nach genauerer Betrachtung der Grammatik sieht man, dass man zunächst die Produktion S → Ac nehmen sollte, um den Syntaxbaum zu erweitern:



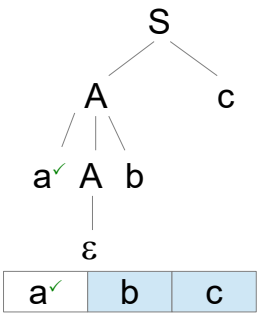

(3) Im nächsten Schritt können wir den Baum erweitern, indem wir die Produktion A → aAb wählen.



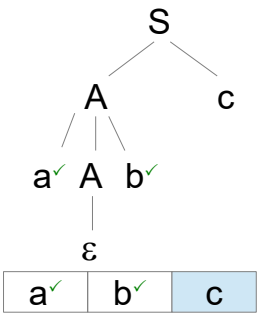

(4) Wir sehen, dass das erste Zeichen a des Eingabeworts zum bisher gebildeten Ableitungsbaum passt und können das schon mal als erkannt "abhaken".



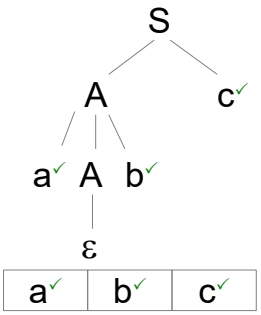
(5) Im nächsten Schritt wenden wir die Produktion A → ε an:




(6) Aus dem nichtterminalen Symbol A entsteht das leere Wort, so dass nun auch das Symbol b abgehakt werden kann:




(7) Und zum Schluss kann auch das c als erkannt markiert werden:



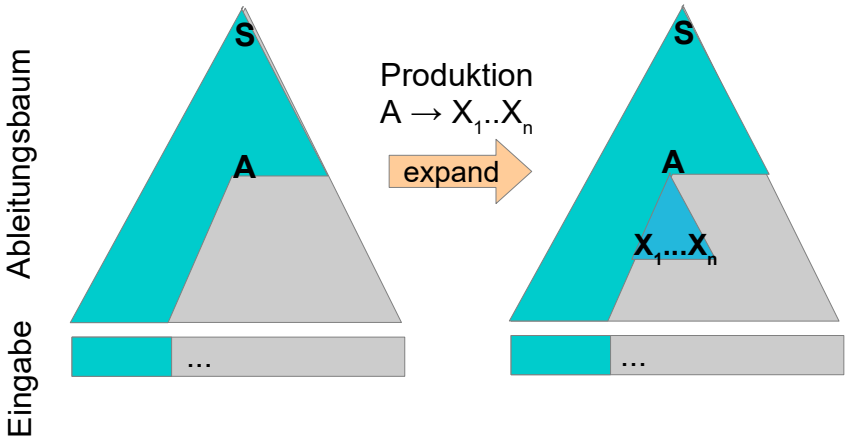

Nun ist der Ableitungsbaum vollständig "top-down" gebildet und alle Zeichen des Eingabeworts sind erkannt worden. Somit ist gezeigt, dass das Wort **abc** zur Sprache der Grammatik gehört.

Betrachtet man die vorgestellt Vorgehensweise nochmals im Überblick, stellt man fest, dass zwei Arten von Schritten durchgeführt wurden:

- **Expand**: Der Syntaxbaum wird erweitert, indem für das am weitesten links stehende nichtterminale Symbol eine Produktion gewählt wird.
- **Consume**: Ist das nächste zu betrachtende Zeichen im Syntaxbaum ein terminales Zeichen, dann muss das gleiche Zeichen auch im Eingabewort als nächstes kommen. Ist das der Fall, kann das Zeichen in Baum und Eingabe als erkannt betrachtet werden (d.h. das aktuelle betrachtete Zeichen der Eingabe wird dabei "konsumiert").

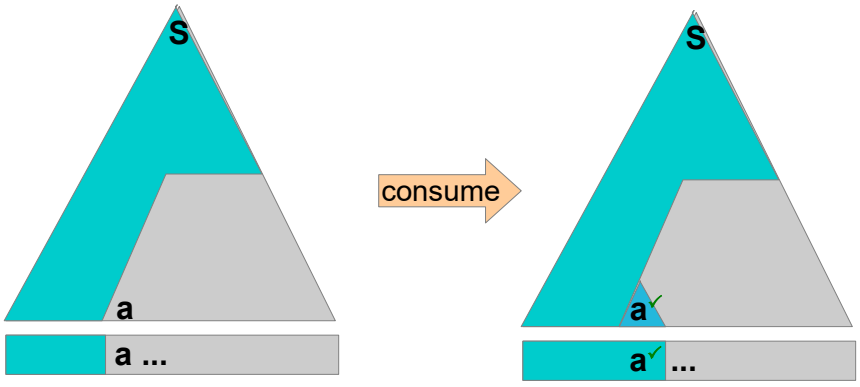
### *Veranschaulichung der Operation*

- **Expand**-Operation:




Die bereits erkannten Teile der Eingabe und des Syntaxbaums sind hier grün dargestellt. Im Syntaxbaum steht als nächstes ein nichtterminales Zeichen A (d.h. "es muss jetzt etwas kommen, was aus A generiert werden kann"). Es wird eine geeignete A-Produktion ausgewählt, um den Syntaxbaum zu erweitern.

- **Consume-Operation**:

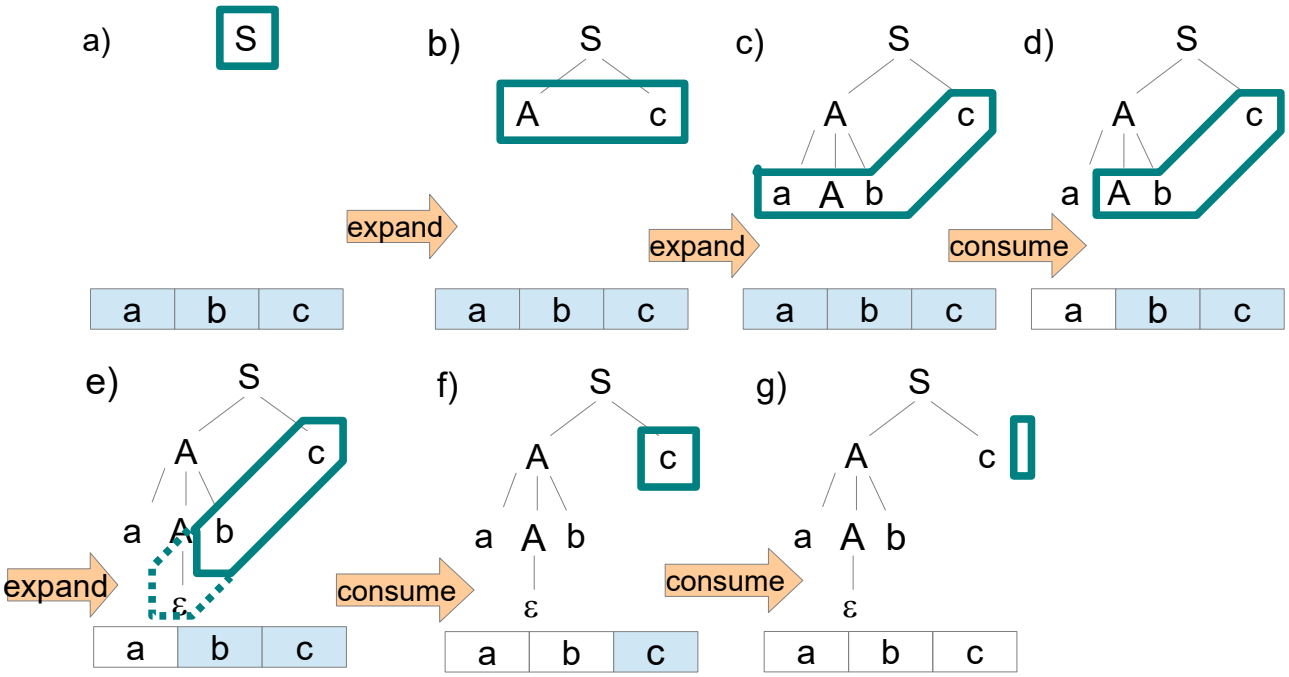



Dem Syntaxbaum ist zu entnehmen, dass als nächstes eine terminales Zeichen a kommen muss. Kommt das gleiche Zeichen auch in der Eingabe, kann das Zeichen in Eingabe und Baum als erkannt betrachtet werden.

## 10.2.2 Konstruktion eines Kellerautomaten für eine Grammatik

Kann das Prinzip der Top-Down-Syntaxanalyse mit Hilfe von Kellerautomaten umgesetzt werden? Wir betrachten dazu nochmals das vorige Beispiel. Für die Vorgehensweise bei der Analyse ist jeweils der "Rand" des Syntaxbaums mit den terminalen und nichtterminalen Symbolen, die noch erwartet werden, von Bedeutung. In der folgenden Abbildung sind diese Teile grün umrandet.

## *Beispiel 10.13 - Expand- und Consume-Schritte*




Der grün gekennzeichnete "Rand" des Ableitungsbaum ist eine Folgen von terminalen und nichtterminalen Symbolen und gibt an, was in der Eingabe noch erwartet wird. Durch die Expand- und Consume-Operationen wird diese Folge immer nur am vorderen (linken) Ende verändert:

- Steht vorne ein nichtterminales Symbol, muss ein Expand-Schritt ausgeführt werden, der das Nichtterminal entfernt und durch die Folge der rechten Seite der Produktion ersetzt.
- Steht vorne ein terminales Zeichen, das auch mit dem nächsten Zeichen in der Eingabe übereinstimmt, kann das Zeichen aus der Folge entfernt werden und die Eingabeposition geht auch um ein Zeichen weiter.

Da immer nur am "vorderen" Ende gearbeitet wird, kann die Folge der noch erwarteten Symbole einfach als Stack (Kellerspeicher) repräsentiert werden.

Ist am Ende der Stack (d.h. die Folge der noch erwarteten Symbole) leer und das Eingabewort komplett verarbeitet worden, ist eine vollständiger Syntaxbaum für das Wort gebildet worden.

## *Anmerkung - Stack*

- Ein *Stack* (*Kellerspeicher*) ist eine Datenstruktur für eine Sammlung von Einträgen, die nach Last-in-first-out-Strategie (LIFO) verwaltet werden. Das, was zuletzt dazugenommen wurde, wird auch wieder als erstes entfernt. Die Operationen werden üblicherweise mit push(x) und pop() bezeichnet.

Diese Idee der Top-Down-Analyse ist die Basis für die nachfolgend beschriebene Konstruktion eines Kellerautomaten zu einer kontextfreien Grammatik.

## *Konstruktion 10.14 - Kellerautomat für kontextfreie Grammatik*

Gegeben sei eine kontextfreie Grammatik $G = (N, T, P, S)$. Der zugehöriger Kellerautomat $K_G$ wird folgendermaßen definiert:

- Eingabealphabet: $\Sigma = T$ (terminale Symbole der Grammatik)
- Zustände: $Z = \{z_0, z_1, z_e\}$,
- $z_0$ ist Startzustand
- $z_e$ ist einziger Endzustand
- $k_0$ ist das Kellerstartsymbol
- Kelleralphabet: $\Gamma = N \cup T \cup \{k_0\}$,
- Zustandsübergangsrelation wird abhängig von Startsymbol $S$, den Produktionen und den terminalen Symbolen der Grammatik folgendermaßen definiert:

(1) $z_0, k_0 \xrightarrow{\varepsilon} z_1, k_0S$

(2) $z_1, a \xrightarrow{a} z_1, \varepsilon$ für jedes Zeichen $a \in T$

(3) $z_1, A \xrightarrow{\varepsilon} z_1, X_n...X_1$ für jede Produktion $A \rightarrow X_1...X_n \in P$

(4) $z_1, k_0 \xrightarrow{\varepsilon} z_e, k_0$

## *Erläuterung*

- Auf dem Stack werden terminale und nichtterminale Symbole abgelegt. Das Kellersymbol $k_0$ wird verwendet, um zu erkennen, ob der Kellerinhalt vollständig abgearbeitet ist
- Die Zustandsübergänge bewirken folgendes:

  (1) Initialisierung: Das Startsymbol der Grammatik wird auf dem Stack abgelegt (als Wurzel des Syntaxbaums)

  (2) "*consume*": Der Stack enthält oben ein erwartetes terminales Zeichen, das auch in der Eingabe kommt. Das Zeichen wird vom Stack entfernt, die Eingabe geht zum nächsten Zeichen.

  (3) "*expand*": Der Stack enthält oben das nächste erwartete Nichtterminal: Das nichtterminale Zeichen kann durch die rechte Seite einer Produktion mit passender linker Seite ersetzt werden. Die Symbole der rechten Seite der Produktion werden *in umgekehrter Reihenfolge auf den Stack* gelegt, so dass das oberste, als erstes erwartete Symbol, oben auf dem Stack liegt (in der Notation

hier rechts am Ende der Folge). Es ist ein ε-Übergang, so dass die Eingabeposition unverändert bleibt.

(4) Der Kellerautomat kann in den Endzustand gehen, wenn der Stack bis auf das Kellerstartsymbol $k_0$, das unten im Stack liegen bleibt, wieder leer geworden ist.

### *Anmerkung*

- Der so zu einer Grammatik konstruierte Kellerautomat ist *nichtdeterministisch,* da bei einem "expand"-Schritten zu einem Nichtterminal *A* jede Produktion mit *A* als linker Seite verwendet werden kann.
- Um bei der Analyse am Ende erfolgreich zu sein, muss also "geraten" werden, welche Produktion genommen werden muss, wenn es mehrere Möglichkeiten gibt.

### *Beispiel 10.15 - Kellerautomat für kontextfreie Grammatik*

Es soll ein Kellerautomat zu folgender Grammatik konstruieren werden.

$S \rightarrow aA \mid Ac$
$A \rightarrow aAb \mid \varepsilon$

- Definition der Übergangsrelation des Kellerautomaten:

| | | |
|---|---|---|
| (1) | $z_0, k_0 \xrightarrow{\varepsilon} z_1, k_0S$ | Startsymbol auf Keller legen |
| (2.1) | $z_1, a \xrightarrow{a} z_1, \varepsilon$ | "consume a": Zeichen a von Stack entfernen. |
| (2.2) | $z_1, b \xrightarrow{b} z_1, \varepsilon$ | "consume b": Zeichen b von Stack entfernen. |
| (2.3) | $z_1, c \xrightarrow{c} z_1, \varepsilon$ | "consume c": Zeichen c von Stack entfernen. |
| (3.1) | $z_1, S \xrightarrow{\varepsilon} z_1, Aa$ | "expand" mit Produktion S → aA |
| (3.2) | $z_1, S \xrightarrow{\varepsilon} z_1, cA$ | "expand" mit Produktion S → Ac |
| (3.3) | $z_1, A \xrightarrow{\varepsilon} z_1, bAa$ | "expand" mit Produktion A → aAb |
| (3.4) | $z_1, A \xrightarrow{\varepsilon} z_1, \varepsilon$ | "expand" mit Produktion A → ε |
| (4) | $z_1, k_0 \xrightarrow{\varepsilon} z_e, k_0$ | in Endzustand wechseln, wenn Stack wieder leer geworden ist (bis auf $k_0$) |

- Analyse des Eingabeworts abc:

| | | | |
|---|---|---|---|
| | $(z_0,$ | $k_0,$ abc) | Startkonfiguration |
| → | $(z_1,$ | $k_0S,$ abc) | (1) Startsymbol auf Keller gelegt |
| → | $(z_1,$ | $k_0cA,$ abc) | (3.2) "expand" mit S → Ac |
| → | $(z_1,$ | $k_0cbAa,$ abc) | (3.3) "expand" mit S → aAb |
| → | $(z_1,$ | $k_0cbA,$ bc) | (2.1) "consume" a |

$\rightarrow$ $(z_1, \quad k_0cb, \quad bc)$ (3.4) "expand" mit $A \rightarrow \varepsilon$

$\rightarrow$ $(z_1, \quad k_0c, \quad c)$ (2.2) "consume" b

$\rightarrow$ $(z_1, \quad k_0, \quad \varepsilon)$ (2.3) "consume" c

$\rightarrow$ $(z_e, \quad k_0, \quad \varepsilon)$ (4) Endzustand erreicht,

Betrachtet man, welche Expand- und Consume-Schritte hier ausgeführt wurde, sieht man, das der Kellerautomat genau die in Beispiel 10.13 dargestellten Analyseschritte ausführt.

### *Aufgabe 10.16 - Kellerautomat für kontextfreie Grammatik*

Gegeben ist folgende kontextfreie Grammatik:

$$\begin{aligned} S &\rightarrow \quad aSb \\ &\quad\ | \quad cAc \\ A &\rightarrow \quad aA \\ &\quad\ | \quad \varepsilon \end{aligned}$$

(1) Konstruieren Sie einen Kellerautomaten, der die Sprache der Grammatik akzeptiert.

(2) Zeigen Sie durch einen Ableitungsbaum, dass das Wort `aacabbcbb` ableitbar ist.

(3) Zeigen Sie, dass dieses Wort auch vom Kellerautomaten akzeptiert wird.

(4) Welche Sprache wird durch den Kellerautomaten akzeptiert?

### *Satz 10.17 - Äquivalenz von Kellerautomat zur Grammatik*

Der gemäß Konstruktion 10.14 zu einer kontextfreien Grammatik *G* gebildete nichtdeterministische Kellerautomat $K_G$ akzeptiert genau die Sprache der Grammatik.

Auf einen formalen Beweis soll hier verzichtet werden. Bei genauerer Betrachtung sieht man,

- dass der Kellerautomat immer zu einer erfolgreichen Endkonfiguration kommen kann, wenn ein Wort zur Sprache der Grammatik gehört und
- dass sich ein Ableitungsbaum bilden lässt (und das Wort somit zur Sprache der Grammatik gehört), wenn der Kellerautomat erfolgreich das Wort analysieren kann.

### *Anmerkung - Kellerautomaten und praktische Syntaxanalyse*

- Durch die Möglichkeit, den Kellerautomaten nichtdeterministisch definieren zu können, indem passende Produktionen für die Expand-Schritte "geraten" werden können, ist es einfach, einen Automaten für eine Grammatik anzugeben. Für die praktische, effiziente Syntaxanalyse, wie sie

beispielsweise für die Implementierung von Programmiersprachen benötigt wird, taugen Kellerautomaten aufgrund des Nichtdeterminismus, der dadurch entsteht, allerdings nicht.

- Es ist auch nicht möglich, nichtdeterministische Kellerautomaten systematisch in äquivalente, effizient ausführbare deterministische Kellerautomaten umzuwandeln.
- Im nächsten Kapitel wird dann vorgestellt werden, wie das Konzept der Top-Down-Analyse mit Kellerautomaten adaptiert und weiterentwickelt werden kann, um ein praktisch einsetzbares, effizientes Syntaxanalyse-Verfahren zu erhalten. Bei den expand-Schritten wird dazu eine "Vorausschau" auf die Eingabe verwendet, um die passende Produktion auswählen zu können, so dass der Ablauf deterministisch wird. Es gibt dann allerdings Einschränkungen, dies funktioniert nicht bei jeder kontextfreien Grammatik.

## 10.3 Deterministische Kellerautomaten

Im Unterschied zu den endlichen Automaten, bei denen die deterministische Variante der DEAs genauso mächtig ist wie die nichtdeterministischen ε-NEAs, gibt es bei Kellerautomaten einen Unterschied. Nichtdeterministische Kellerautomaten sind echt mächtiger als deterministische Kellerautomaten, d.h. können mehr Sprachen akzeptieren.

### *Definition 10.18 - Deterministische Kellerautomaten*

Ein **Kellerautomat** heißt **deterministisch**, wenn es zu jeder Konfiguration höchstens eine mögliche Nachfolgekonfiguration gibt.

### *Definition 10.19 - Deterministisch kontextfreie Sprachen*

Eine **Sprache** *L* heißt **deterministisch kontextfrei**, falls es einen *deterministischen* Kellerautomaten gibt, der die Sprache *L* akzeptiert.

### *Eigenschaft 10.20 - Deterministisch kontextfreie Sprachen*

Die **deterministisch kontextfreien Sprachen** sind eine *echte Teilmenge* der kontextfreien Sprachen.

D.h. es gibt Sprachen, die von einem nichtdeterministischen Kellerautomaten erkannt werden können, aber nicht von einem deterministischen. Während es bei ε-NEAs immer möglich war, einen äquivalenten DEA zu bilden (mittels Teilmengenkonstruktion), kann nicht zu jedem nichtdeterministischen Kellerautomat auch eine äquivalenter deterministischer Kellerautomat gebildet werden.

### *Beispiel 10.21 - Nicht deterministisch kontextfreie Sprache*

Ein Beispiel für eine Sprache, die zwar *kontextfrei* ist, die aber nicht *deterministisch kontextfrei* ist, ist die Sprache der Palindrome über einem gegebenen Alphabet $\Sigma$.

Es kann leicht dafür eine kontextfreie Grammatik und somit auch ein (nichtdeterministischer) Kellerautomat s.o. angegeben werden. Es kann aber bewiesen werden, dass es keinen *deterministischen* Kellerautomaten gibt, der die Sprache der Palindrome erkennen kann.

## Zusammenfassung zu Lektion 17

### *Diese Fragen sollten Sie nun beantworten können*

- Wie ist ein Kellerautomat aufgebaut und wie funktioniert er?
- Wann wird ein Wort von einem Kellerautomaten akzeptiert?
- Wie wird bei der Top-Down-Syntaxanalyse prinzipiell vorgegangen?
- Welcher Zusammenhang besteht zwischen Kellerautomaten und kontextfreien Grammatiken?
- Wie kann zu einer kontextfreien Grammatik ein äquivalenter Kellerautomat konstruiert werden?
- Welchen wichtigen Unterschied gibt es zwischen deterministischen und nichtdeterministischen Kellerautomaten?