

FGI 1

Automaten, Formale Sprachen,
Berechenbarkeit

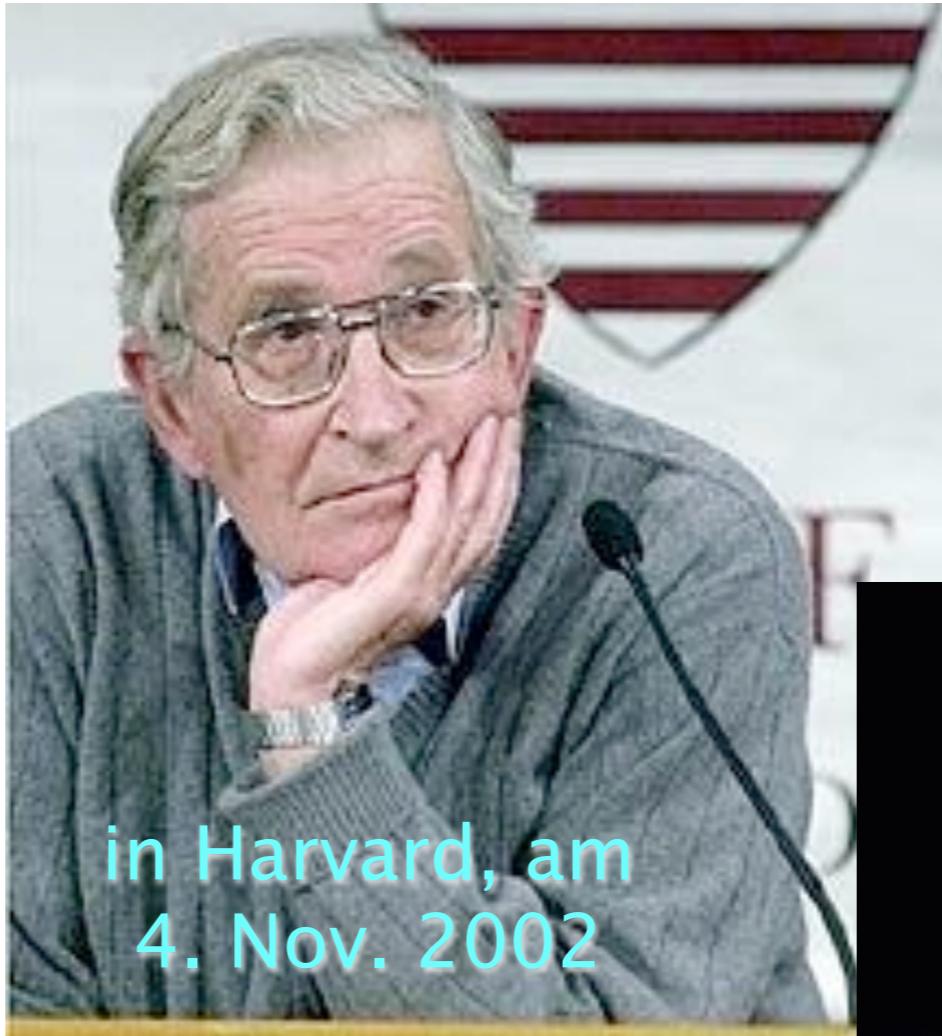
Kap. 15 Kontextfreie Grammatiken

Michael Köhler-Bußmeier

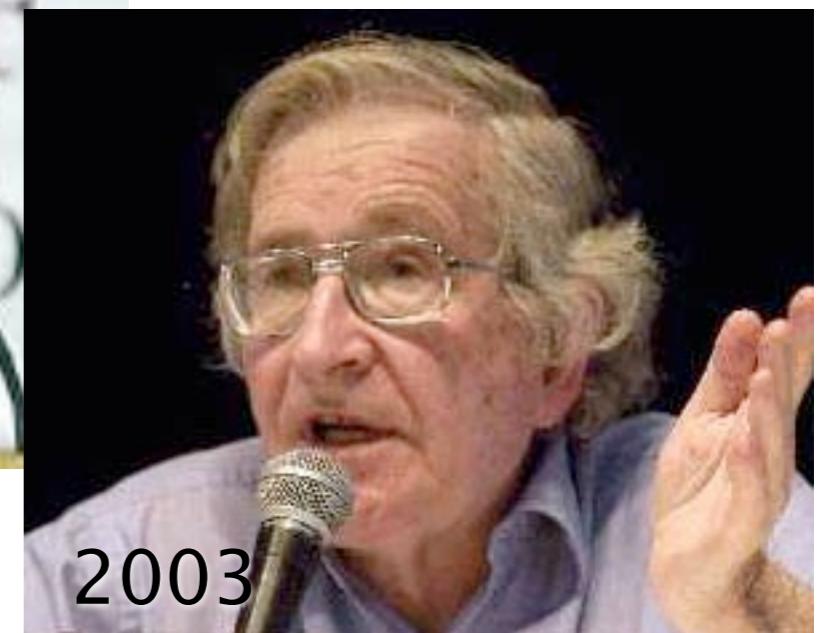
Grammatiken für natürliche Sprache

- Bekannt aus der natürlichen Sprache: Grammatik
- Regeln zur Bildung von Sätzen.
- Z.B. in der englischen Grammatik: Ein **Satz** besteht aus einer **Nominalphrase** gefolgt von einer **Verbalphrase**.
- **Wichtig:** die Reihenfolge der Satzbestandteile.

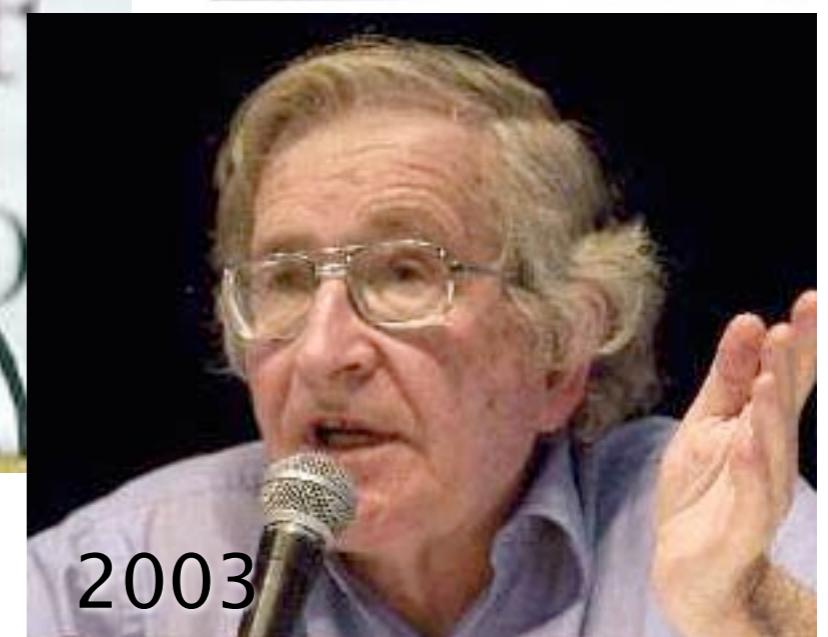
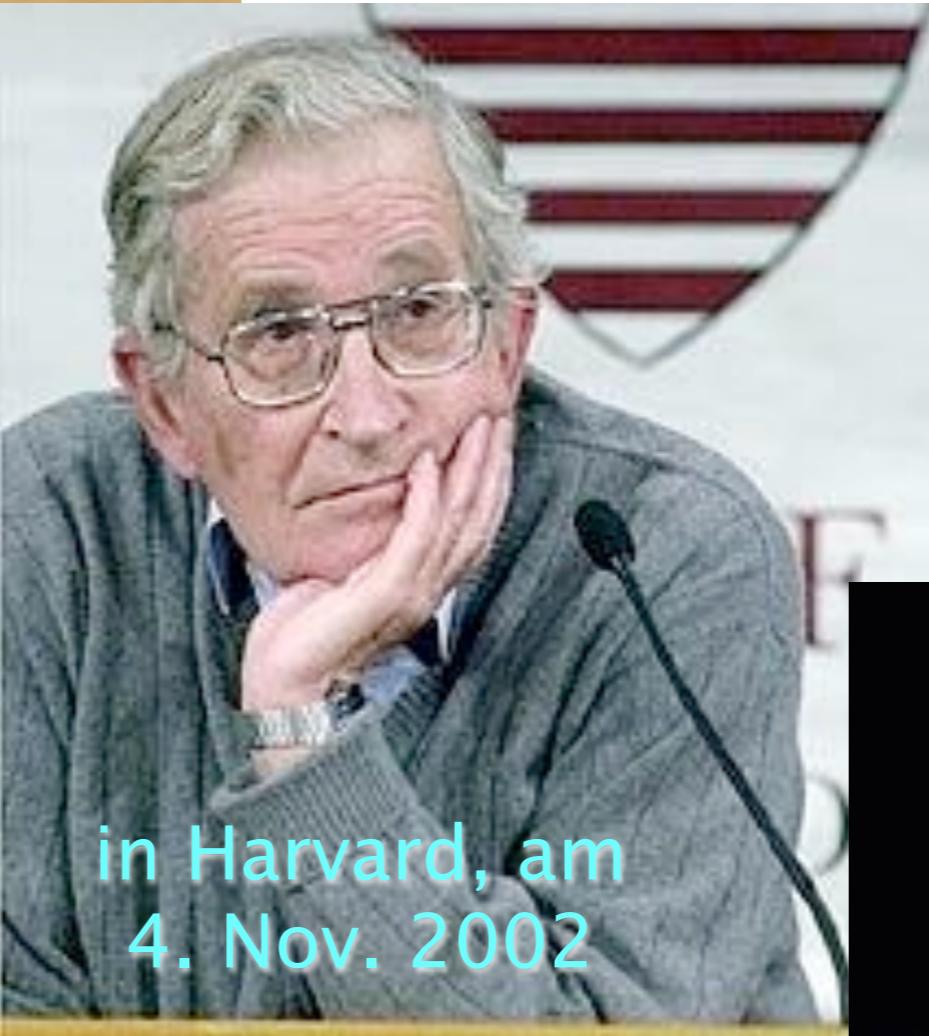
Noam Chomsky, (*7. Dez. 1928 ...)
Prof. für Linguistik am MIT.



Seine Überlegungen ermöglichen
die Syntaxdefinition vieler
bekannter Programmiersprachen!



Noam Chomsky, (*7. Dez. 1928 ...)
Prof. für Linguistik am MIT.



Seine Überlegungen ermöglichen
die Syntaxdefinition vieler
bekannter Programmiersprachen!

englische Grammatik

$\langle \text{Satz} \rangle \longrightarrow \langle \text{Nominalphrase} \rangle \langle \text{Verbalphrase} \rangle$

$\langle \text{Nominalphrase} \rangle \longrightarrow \langle \text{Artikel} \rangle \langle \text{Nomen} \rangle$

$\langle \text{Verbalphrase} \rangle \longrightarrow \langle \text{Verb} \rangle \langle \text{Objekt} \rangle$

$\langle \text{Objekt} \rangle \longrightarrow \langle \text{Nominalphrase} \rangle$

$\langle \text{Artikel} \rangle \longrightarrow _a \mid _the$

$\langle \text{Nomen} \rangle \longrightarrow _woman \mid _man \mid _book \mid _golf$

$\langle \text{Verb} \rangle \longrightarrow _reads \mid _plays \mid _buys$

- $\langle Name \rangle$ ist eine **metalinguistische Variable**.

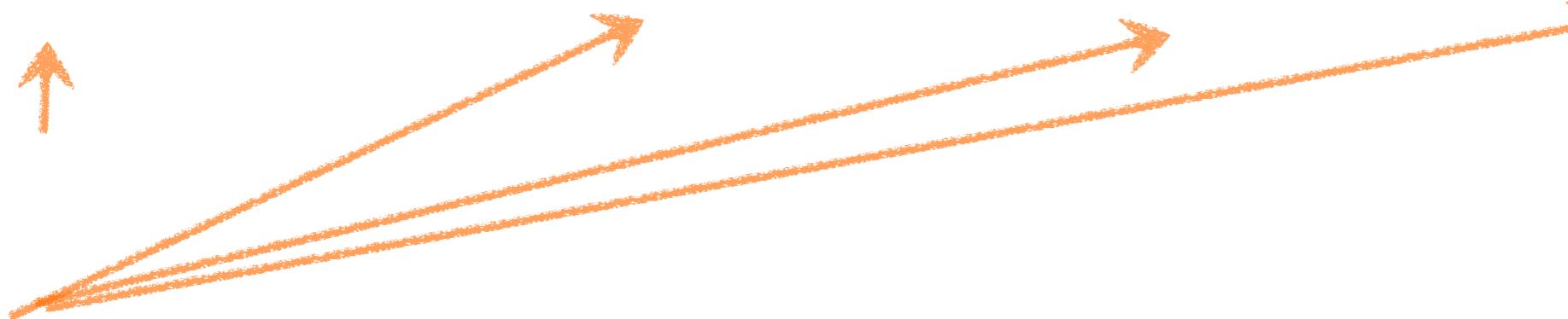
J.W. Backus und P. Naur

John W. Backus, (*3. Dez. 1924 – 17. März 2007)

IBM, 1977 Turing Preis, 1991 Rückzug aus der Welt der EDV.

Peter Naur, (*25. Okt. 1928 ...)

1969-98 Professor für Datalogi an der Univ. Kopenhagen.



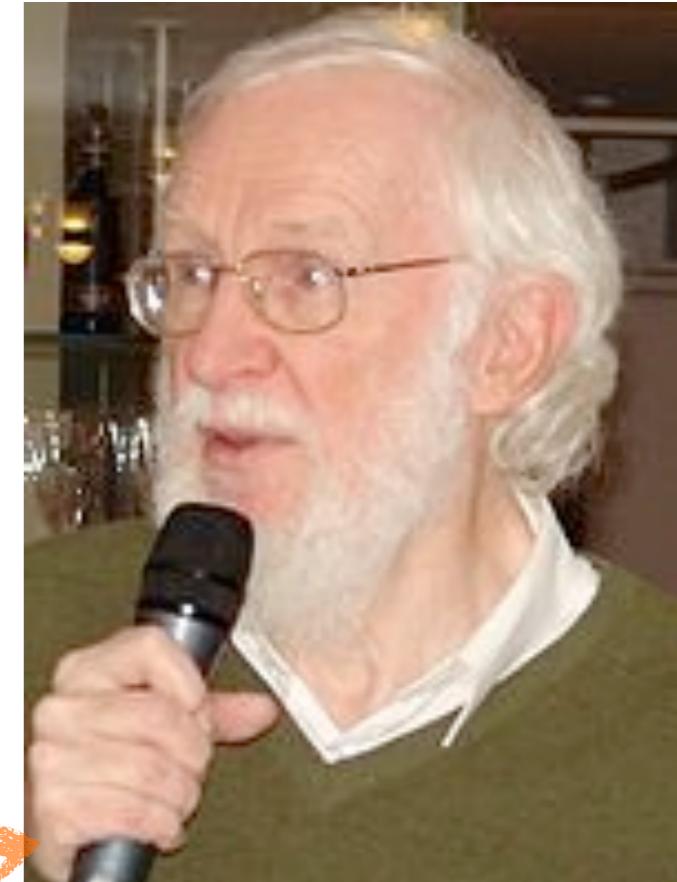
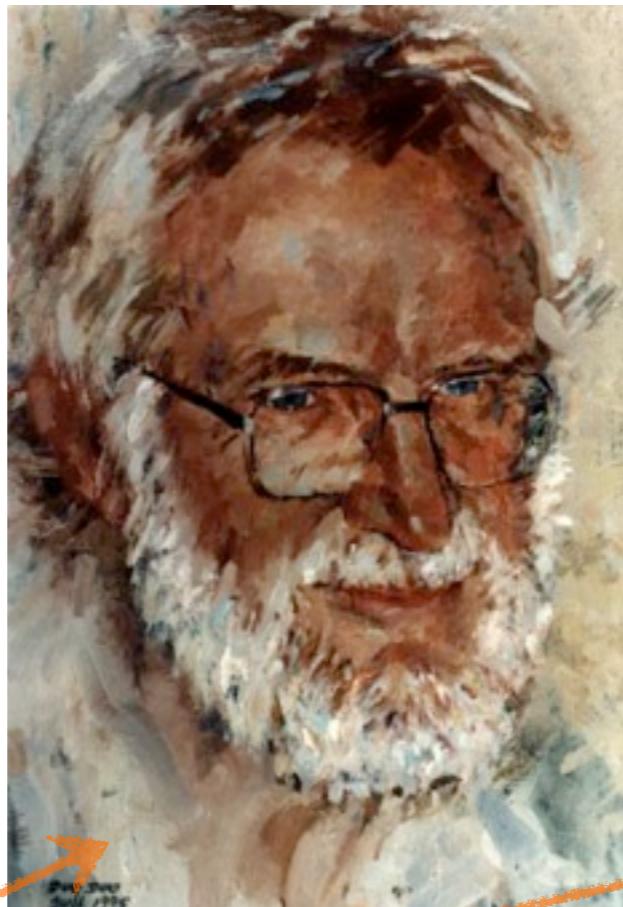
J.W. Backus und P. Naur

John W. Backus, (*3. Dez. 1924 – 17. März 2007)

IBM, 1977 Turing Preis, 1991 Rückzug aus der Welt der EDV.

Peter Naur, (*25. Okt. 1928 ...)

1969-98 Professor für Datalogi an der Univ. Kopenhagen.



(**Backus** entwickelte Fortran und später eine der ersten funktionalen Programmiersprachen, **BNF-Darstellung** kontextfreier Grammatiken, **Naur** war an der Entwicklung von Algol 60 beteiligt.)

Anwendungsgebiete

Hauptanwendungsgebiete von Grammatiken in der Praxis:

- **Compiler-Design:**

Die Syntax der Programmiersprachen wird als Grammatik spezifiziert. Ein **Compiler-Compiler** (z.B. yacc, javacc) generiert dann einen Parser.

- **DTD/XML:**

Die **Document Type Definition (DTD)** ist eine standardisierte Form, die generelle (Baum-)Struktur von XML-Dateien zu definieren.

kontextfreie Grammatik (kfG, CFG)

Definition 15.1:

Eine **kontextfreie Grammatik** wird beschrieben durch ein Quadrupel $G = (\Sigma, N, P, S)$ mit:

Σ ist ein mit N disjunktes endliches Alphabet von **Terminalsymbolen** (auch: Terminalen).

N ist ein endliches Alphabet von **Variablen** (auch: Nichtterminalen, Nonterminalen, metalinguistischen Variablen, Hilfszeichen oder syntaktischen Kategorien). $N \cap \Sigma = \emptyset$.

P ist eine endliche Menge von **Produktionen** oder **Regeln**. Es gilt

$$P \subseteq N \times (N \cup \Sigma)^*$$

S ist das **Startsymbol** und es gilt $S \in N$.

Beispiel

$S \longrightarrow aSa \mid bSb \mid \epsilon$ ist Kurzschreibweise der Regelmenge
 $P := \{(S, aSa), (S, bSb), (S, \epsilon)\}$ einer Grammatik $G_1 := (\{a, b\}, \{S\}, P, S)$.

Definition 15.1:

Eine **kontextfreie Grammatik** wird beschrieben durch ein Quadrupel $G = (\Sigma, N, P, S)$ mit:

Σ ist ein mit N disjunktes endliches Alphabet von **Terminalsymbolen** (auch: Terminalen).

N ist ein endliches Alphabet von **Variablen** (auch: Nichtterminalen, Nonterminalen, metalinguistischen Variablen, Hilfszeichen oder syntaktischen Kategorien). $N \cap \Sigma = \emptyset$.

P ist eine endliche Menge von **Produktionen** oder **Regeln**. Es gilt

$$P \subseteq N \times (N \cup \Sigma)^*$$

S ist das **Startsymbol** und es gilt $S \in N$.

Ableitungsrelation bei CFG

Definition 15.3:

- Für kontextfreie Grammatiken ist die *einschrittigen Ableitungsrelation* $\xrightarrow[G]{\cdot}$ definiert:

$$\forall u, v \in (\Sigma \cup N)^*: u \xrightarrow[G]{\cdot} v \quad \text{gdw.} \quad \exists u_1, u_2 \in (N \cup \Sigma)^*: \quad$$

$$\exists A \in N: \exists A \rightarrow w \in P: u = u_1 A u_2 \wedge v = u_1 w u_2$$

- Es gilt stets $w \xrightarrow[G]{*} w$.
- $u \in (N \cup \Sigma)^*$, mit $S \xrightarrow[G]{*} u$ nennt man **Satzform**.

Letztendlich interessant sind nur die Satzformen über dem Terminalalphabet:

- $L(G) := \{w \in \Sigma^* \mid S \xrightarrow[G]{*} w\}$.
- Mit \mathcal{Cf} sei die Familie aller kontextfreien Sprachen bezeichnet ($TYP2_\Sigma$ oder kfS_Σ bei Vossen/Witt).

Länge einer Ableitung

- Als **Ableitung** der Länge n eines Wortes w in einer CFG bezeichnen wir eine *Sequenz* von Wörtern $w_0, w_1, w_2, \dots, w_{n-1}, w$ mit $w_0 := S$ und $w := w_n$, wobei $w_i \xrightarrow[G]{} w_{i+1}$ für $0 \leq i \leq n$.
- Eine solche Ableitung schreiben wir häufig einfach als: $S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \dots \Rightarrow w_{n-1} \Rightarrow w$
- $S \Rightarrow aSa \Rightarrow abSba \Rightarrow abba$ ist Ableitung der Länge 3 mit den Regeln $S \longrightarrow aSa \mid bSb \mid \epsilon$.

Eine Ableitung in einer CFG heißt **Linksableitung** (bzw. **Rechtsableitung**) gdw. in jedem Schritt der Ableitung das ersetzte Zeichen das am weitesten links (bzw. rechts) stehende Nonterminal ist.

(Ableitungsrelationen $\xrightarrow{\text{li}}$ und $\xrightarrow{\text{re}}$)

Das Beispiel: Expression

Programmiersprache:

S : Start. E : expression, I : identifier, C : condition

$$S \longrightarrow I = E;$$

$$I \longrightarrow u$$

$$E \longrightarrow (E) \mid -(E) \mid (E + E) \mid (E * E) \mid (E / E) \mid (E - E) \mid v$$

$$S \longrightarrow S \text{ while } (C)\{S\}; S \mid \lambda$$

$$C \longrightarrow E > E$$

$$E \longrightarrow a \mid b$$

Linksableitung im Detail

S

$S \longrightarrow I = E;$

$I \longrightarrow \text{u}$

$E \longrightarrow (E) \mid -(E) \mid (E+E) \mid (E*E) \mid (E/E)$

$S \longrightarrow S \text{ while } (C)\{S\}; S \mid \lambda$

$C \longrightarrow E > E$

$E \longrightarrow \text{a} \mid \text{b}$

Linksableitung im Detail

S

\Rightarrow

$S \rightarrow I = E;$

$I \rightarrow \mathbf{u}$

$E \rightarrow (E) \mid -(E) \mid (E+E) \mid (E * E) \mid (E/E)$

$S \rightarrow S \mathbf{while} (C)\{S\}; S \parallel \lambda$

$C \rightarrow E > E$

$E \rightarrow \mathbf{a} \mid \mathbf{b}$

Linksableitung im Detail

$S \Rightarrow S \text{ while}(C)\{S\}; S$

$S \rightarrow I = E;$

$I \rightarrow \mathbf{u}$

$E \rightarrow (E) \mid -(E) \mid (E+E) \mid (E * E) \mid (E/E)$

$S \rightarrow S \text{ while}(C)\{S\}; S \mid \lambda$

$C \rightarrow E > E$

$E \rightarrow \mathbf{a} \mid \mathbf{b}$

Linksableitung im Detail

$S \Rightarrow S \text{while}(C)\{S\}; S$

$S \rightarrow I = E;$

$I \rightarrow \text{u}$

$E \rightarrow (E) \mid -(E) \mid (E+E) \mid (E * E) \mid (E/E)$

$S \rightarrow S \text{while}(C)\{S\}; S \mid \lambda$

$C \rightarrow E > E$

$E \rightarrow \text{a} \mid \text{b}$

Linksableitung im Detail

$$\begin{array}{lcl} S & \Rightarrow & S \text{ while}(C)\{S\}; S \\ & \Rightarrow & \text{while}(C)\{S\}; S \end{array}$$

$$\begin{array}{l} S \longrightarrow I = E; \\ I \longrightarrow \mathbf{u} \\ E \longrightarrow (E) \mid -(E) \mid (E+E) \mid (E * E) \mid (E/E) \\ S \longrightarrow S \text{ while}(C)\{S\}; S \mid \lambda \\ C \longrightarrow E > E \\ E \longrightarrow \mathbf{a} \mid \mathbf{b} \end{array}$$

Linksableitung im Detail

$$\begin{array}{lcl} S & \Rightarrow & S \text{ while}(C)\{S\}; S \\ & \Rightarrow & \text{while } (C) \{S\}; S \end{array}$$

$$\begin{array}{l} S \longrightarrow I = E; \\ I \longrightarrow \text{u} \\ E \longrightarrow (E) \mid -(E) \mid (E+E) \mid (E * E) \mid (E/E) \\ S \longrightarrow S \text{ while } (C)\{S\}; S \mid \lambda \\ C \longrightarrow E > E \\ E \longrightarrow \text{a} \mid \text{b} \end{array}$$

Linksableitung im Detail

		$S \rightarrow I = E;$
		$I \rightarrow \mathbf{u}$
$S \Rightarrow S \mathbf{while}(C)\{S\}; S$		$E \rightarrow (E) \mid -(E) \mid (E+E) \mid (E * E) \mid (E/E)$
$\Rightarrow \mathbf{while}(C)\{S\}; S$		$S \rightarrow S \mathbf{while}(C)\{S\}; S \mid \lambda$
$\Rightarrow \mathbf{while}(E > E)\{S\}; S$		$C \rightarrow E > E$
		$E \rightarrow \mathbf{a} \mid \mathbf{b}$

Linksableitung im Detail

		$S \longrightarrow I = E;$
		$I \longrightarrow \text{u}$
$S \quad \Rightarrow \quad S \text{ while}(C)\{S\}; S$		$E \longrightarrow (E) \mid -(E) \mid (E+E) \mid (E * E) \mid (E/E)$
$\Rightarrow \quad \text{while}(C)\{S\}; S$		$S \longrightarrow S \text{ while}(C)\{S\}; S \mid \lambda$
$\Rightarrow \quad \text{while}(\textcolor{red}{E} > E)\{S\}; S$	$C \longrightarrow E > E$	$E \longrightarrow \text{a} \mid \text{b}$

Linksableitung im Detail

		$S \rightarrow I = E;$
		$I \rightarrow \mathbf{u}$
$S \Rightarrow S \mathbf{while}(C)\{S\}; S$		$E \rightarrow (E) \mid -(E) \mid (E+E) \mid (E * E) \mid (E/E)$
$\Rightarrow \mathbf{while}(C)\{S\}; S$		$S \rightarrow S \mathbf{while}(C)\{S\}; S \mid \lambda$
$\Rightarrow \mathbf{while}(E > E)\{S\}; S$		$C \rightarrow E > E$
$\Rightarrow \mathbf{while}(\mathbf{a} > E)\{S\}; S$		$E \rightarrow \mathbf{a} \mid \mathbf{b}$

Linksableitung im Detail

		$S \longrightarrow I = E;$
		$I \longrightarrow u$
$S \Rightarrow S \text{ while}(C)\{S\}; S$		$E \longrightarrow (E) \mid -(E) \mid (E+E) \mid (E * E) \mid (E/E)$
$\Rightarrow \text{while}(C)\{S\}; S$		$S \longrightarrow S \text{ while}(C)\{S\}; S \mid \lambda$
$\Rightarrow \text{while}(E > E)\{S\}; S$		$C \longrightarrow E > E$
$\Rightarrow \text{while}(a > E)\{S\}; S$		$E \longrightarrow a \mid b$

Linksableitung im Detail

		$S \longrightarrow I = E;$
		$I \longrightarrow u$
$S \quad \Rightarrow \quad S \text{ while}(C)\{S\}; S$		$E \longrightarrow (E) \mid -(E) \mid (E+E) \mid (E * E) \mid (E/E)$
$\Rightarrow \quad \text{while}(C)\{S\}; S$		$S \longrightarrow S \text{ while}(C)\{S\}; S \mid \lambda$
$\Rightarrow \quad \text{while}(E > E)\{S\}; S$		$C \longrightarrow E > E$
$\Rightarrow \quad \text{while}(a > E)\{S\}; S$		$E \longrightarrow a \mid b$
$\Rightarrow \quad \text{while}(a > b)\{S\}; S$		

Linksableitung im Detail

$S \Rightarrow S \text{ while}(C)\{S\}; S$
 $\Rightarrow \text{while}(C)\{S\}; S$
 $\Rightarrow \text{while}(E > E)\{S\}; S$
 $\Rightarrow \text{while}(\text{a} > E)\{S\}; S$
 $\Rightarrow \text{while}(\text{a} > \text{b})\{S\}; S$

$S \rightarrow I = E;$

$I \rightarrow \text{u}$

$E \rightarrow (E) \mid -(E) \mid (E+E) \mid (E * E) \mid (E/E)$

$S \rightarrow S \text{ while}(C)\{S\}; S \mid \lambda$

$C \rightarrow E > E$

$E \rightarrow \text{a} \mid \text{b}$

Linksableitung im Detail

		$S \rightarrow I = E;$
		$I \rightarrow u$
$S \Rightarrow S \text{ while}(C)\{S\}; S$		$E \rightarrow (E) \mid -(E) \mid (E+E) \mid (E * E) \mid (E/E)$
$\Rightarrow \text{while}(C)\{S\}; S$		$S \rightarrow S \text{ while}(C)\{S\}; S \mid \lambda$
$\Rightarrow \text{while}(E > E)\{S\}; S$		$C \rightarrow E > E$
$\Rightarrow \text{while}(a > E)\{S\}; S$		$E \rightarrow a \mid b$
$\Rightarrow \text{while}(a > b)\{S\}; S$		
$\Rightarrow \text{while}(a > b)\{I = E; \}; S$		

Linksableitung im Detail

$S \Rightarrow S \text{ while}(C)\{S\}; S$

$S \rightarrow I = E;$

$I \rightarrow u$

$\Rightarrow \text{while}(C)\{S\}; S$

$S \rightarrow S \text{ while}(C)\{S\}; S \mid \lambda$

$\Rightarrow \text{while}(E > E)\{S\}; S$

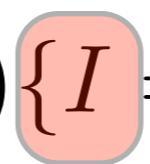
$C \rightarrow E > E$

$\Rightarrow \text{while}(a > E)\{S\}; S$

$E \rightarrow a \mid b$

$\Rightarrow \text{while}(a > b)\{S\}; S$

$\Rightarrow \text{while}(a > b)\{I = E; \}; S$



Linksableitung im Detail

		$S \rightarrow I = E;$
		$I \rightarrow u$
$S \Rightarrow S \text{ while}(C)\{S\}; S$		$E \rightarrow (E) \mid -(E) \mid (E+E) \mid (E * E) \mid (E/E)$
$\Rightarrow \text{while}(C)\{S\}; S$		$S \rightarrow S \text{ while}(C)\{S\}; S \mid \lambda$
$\Rightarrow \text{while}(E > E)\{S\}; S$		$C \rightarrow E > E$
$\Rightarrow \text{while}(a > E)\{S\}; S$		$E \rightarrow a \mid b$
$\Rightarrow \text{while}(a > b)\{S\}; S$		
$\Rightarrow \text{while}(a > b)\{I = E; \}; S$		
$\Rightarrow \text{while}(a > b)\{u = E; \}; S$		

Linksableitung im Detail

		$S \rightarrow I = E;$
		$I \rightarrow u$
$S \Rightarrow S \text{ while}(C)\{S\}; S$		$E \rightarrow (E) \mid -(E) \mid (E+E) \mid (E * E) \mid (E/E)$
$\Rightarrow \text{while}(C)\{S\}; S$		$S \rightarrow S \text{ while}(C)\{S\}; S \mid \lambda$
$\Rightarrow \text{while}(E > E)\{S\}; S$		$C \rightarrow E > E$
$\Rightarrow \text{while}(a > E)\{S\}; S$		$E \rightarrow a \mid b$
$\Rightarrow \text{while}(a > b)\{S\}; S$		
$\Rightarrow \text{while}(a > b)\{I = E; \}; S$		
$\Rightarrow \text{while}(a > b)\{u = E; \}; S$		

Linksableitung im Detail

		$S \rightarrow I = E;$
		$I \rightarrow u$
$S \Rightarrow S \text{ while}(C)\{S\}; S$		$E \rightarrow (E) \mid -(E) \mid (E+E) \mid (E * E) \mid (E/E)$
$\Rightarrow \text{while}(C)\{S\}; S$		$S \rightarrow S \text{ while}(C)\{S\}; S \mid \lambda$
$\Rightarrow \text{while}(E > E)\{S\}; S$		$C \rightarrow E > E$
$\Rightarrow \text{while}(a > E)\{S\}; S$		$E \rightarrow a \mid b$
$\Rightarrow \text{while}(a > b)\{S\}; S$		
$\Rightarrow \text{while}(a > b)\{I = E; \}; S$		
$\Rightarrow \text{while}(a > b)\{u = E; \}; S$		
$\Rightarrow \text{while}(a > b)\{u = (E + E); \}; S$		

Linksableitung im Detail

		$S \rightarrow I = E;$
		$I \rightarrow u$
$S \Rightarrow S \text{ while}(C)\{S\}; S$		$E \rightarrow (E) \mid -(E) \mid (E+E) \mid (E * E) \mid (E/E)$
$\Rightarrow \text{while}(C)\{S\}; S$		$S \rightarrow S \text{ while}(C)\{S\}; S \mid \lambda$
$\Rightarrow \text{while}(E > E)\{S\}; S$		$C \rightarrow E > E$
$\Rightarrow \text{while}(a > E)\{S\}; S$		$E \rightarrow a \mid b$
$\Rightarrow \text{while}(a > b)\{S\}; S$		
$\Rightarrow \text{while}(a > b)\{I = E; \}; S$		
$\Rightarrow \text{while}(a > b)\{u = E; \}; S$		
$\Rightarrow \text{while}(a > b)\{u = (E + E); \}; S$		

Linksableitung im Detail

		$S \rightarrow I = E;$
		$I \rightarrow u$
$S \Rightarrow S \text{ while}(C)\{S\}; S$		$E \rightarrow (E) \mid -(E) \mid (E+E) \mid (E * E) \mid (E/E)$
$\Rightarrow \text{while}(C)\{S\}; S$		$S \rightarrow S \text{ while}(C)\{S\}; S \mid \lambda$
$\Rightarrow \text{while}(E > E)\{S\}; S$		$C \rightarrow E > E$
$\Rightarrow \text{while}(a > E)\{S\}; S$		$E \rightarrow a \mid b$
$\Rightarrow \text{while}(a > b)\{S\}; S$		
$\Rightarrow \text{while}(a > b)\{I = E; \}; S$		
$\Rightarrow \text{while}(a > b)\{u = E; \}; S$		
$\Rightarrow \text{while}(a > b)\{u = (E + E); \}; S$		
$\Rightarrow \text{while}(a > b)\{u = (v + E); \}; S$		

Linksableitung im Detail

		$S \rightarrow I = E;$
		$I \rightarrow u$
$S \Rightarrow S \text{ while}(C)\{S\}; S$		$E \rightarrow (E) \mid -(E) \mid (E+E) \mid (E * E) \mid (E/E)$
$\Rightarrow \text{while}(C)\{S\}; S$		$S \rightarrow S \text{ while}(C)\{S\}; S \mid \lambda$
$\Rightarrow \text{while}(E > E)\{S\}; S$		$C \rightarrow E > E$
$\Rightarrow \text{while}(a > E)\{S\}; S$		$E \rightarrow a \mid b$
$\Rightarrow \text{while}(a > b)\{S\}; S$		
$\Rightarrow \text{while}(a > b)\{I = E; \}; S$		
$\Rightarrow \text{while}(a > b)\{u = E; \}; S$		
$\Rightarrow \text{while}(a > b)\{u = (E + E); \}; S$		
$\Rightarrow \text{while}(a > b)\{u = (v + E); \}; S$		

Linksableitung im Detail

		$S \rightarrow I = E;$
		$I \rightarrow u$
$S \Rightarrow S \text{ while}(C)\{S\}; S$		$E \rightarrow (E) \mid -(E) \mid (E+E) \mid (E * E) \mid (E/E)$
$\Rightarrow \text{while}(C)\{S\}; S$		$S \rightarrow S \text{ while}(C)\{S\}; S \mid \lambda$
$\Rightarrow \text{while}(E > E)\{S\}; S$		$C \rightarrow E > E$
$\Rightarrow \text{while}(a > E)\{S\}; S$		$E \rightarrow a \mid b$
$\Rightarrow \text{while}(a > b)\{S\}; S$		
$\Rightarrow \text{while}(a > b)\{I = E; \}; S$		
$\Rightarrow \text{while}(a > b)\{u = E; \}; S$		
$\Rightarrow \text{while}(a > b)\{u = (E + E); \}; S$		
$\Rightarrow \text{while}(a > b)\{u = (v + E); \}; S$		
$\Rightarrow \text{while}(a > b)\{u = (v + v); \}; S$		

Linksableitung im Detail

		$S \rightarrow I = E;$
		$I \rightarrow u$
$S \Rightarrow S \text{ while}(C)\{S\}; S$		$E \rightarrow (E) \mid -(E) \mid (E+E) \mid (E * E) \mid (E/E)$
$\Rightarrow \text{while}(C)\{S\}; S$		$S \rightarrow S \text{ while}(C)\{S\}; S \quad \lambda$
$\Rightarrow \text{while}(E > E)\{S\}; S$		$C \rightarrow E > E$
$\Rightarrow \text{while}(a > E)\{S\}; S$		$E \rightarrow a \mid b$
$\Rightarrow \text{while}(a > b)\{S\}; S$		
$\Rightarrow \text{while}(a > b)\{I = E; \}; S$		
$\Rightarrow \text{while}(a > b)\{u = E; \}; S$		
$\Rightarrow \text{while}(a > b)\{u = (E + E); \}; S$		
$\Rightarrow \text{while}(a > b)\{u = (v + E); \}; S$		
$\Rightarrow \text{while}(a > b)\{u = (v + v); \}; S$		S

Linksableitung im Detail

		$S \rightarrow I = E;$
		$I \rightarrow u$
$S \Rightarrow S \text{ while}(C)\{S\}; S$		$E \rightarrow (E) \mid -(E) \mid (E+E) \mid (E * E) \mid (E/E)$
$\Rightarrow \text{while}(C)\{S\}; S$		$S \rightarrow S \text{ while}(C)\{S\}; S \mid \lambda$
$\Rightarrow \text{while}(E > E)\{S\}; S$		$C \rightarrow E > E$
$\Rightarrow \text{while}(a > E)\{S\}; S$		$E \rightarrow a \mid b$
$\Rightarrow \text{while}(a > b)\{S\}; S$		
$\Rightarrow \text{while}(a > b)\{I = E; \}; S$		
$\Rightarrow \text{while}(a > b)\{u = E; \}; S$		
$\Rightarrow \text{while}(a > b)\{u = (E + E); \}; S$		
$\Rightarrow \text{while}(a > b)\{u = (v + E); \}; S$		
$\Rightarrow \text{while}(a > b)\{u = (v + v); \}; S$		
$\Rightarrow \text{while}(a > b)\{u = (v + v); \};$		

rechtslineare CFG

Definition 15.2:

Eine $CFG\ G := (\Sigma, N, P, S)$ heißt:

rechtslinear gdw. $P \subseteq N \times \Sigma^*(N \cup \{\epsilon\})$,

linkslinear gdw. $P \subseteq N \times (N \cup \{\epsilon\})\Sigma^*$,

linear gdw. $P \subseteq N \times \Sigma^*(N \cup \{\epsilon\})\Sigma^*$.

rechtslineare CFG

Definition 15.2:

Eine $CFG\ G := (\Sigma, N, P, S)$ heißt:

Die ersten 2
Definitionen
heißen Typ-3
Grammatiken

rechtslinear gdw. $P \subseteq N \times \Sigma^*(N \cup \{\epsilon\})$,

linkslinear gdw. $P \subseteq N \times (N \cup \{\epsilon\})\Sigma^*$,

linear gdw. $P \subseteq N \times \Sigma^*(N \cup \{\epsilon\})\Sigma^*$.

rechtslineare CFG

Definition 15.2:

Eine $CFG\ G := (\Sigma, N, P, S)$ heißt:

Die ersten 2
Definitionen
heißen Typ-3
Grammatiken

rechtslinear gdw. $P \subseteq N \times \Sigma^*(N \cup \{\epsilon\})$,

linkslinear gdw. $P \subseteq N \times (N \cup \{\epsilon\})\Sigma^*$,

linear gdw. $P \subseteq N \times \Sigma^*(N \cup \{\epsilon\})\Sigma^*$.

- $S \rightarrow SS \mid AB, A \rightarrow AA \mid a, B \rightarrow BB \mid b$ ist nicht linear.
- $S \rightarrow aA, A \rightarrow abA \mid aB, B \rightarrow bbB \mid b$ ist rechtslinear.
- $S \rightarrow Ab, A \rightarrow Sb \mid B \mid \epsilon, B \rightarrow Bab \mid \epsilon$ ist linkslinear.
- $S \rightarrow aSb, S \rightarrow a \mid b \mid \epsilon$ ist linear.

Einseitig lineare CFG erzeugt reguläre Menge

Satz 15.1:

Zu jeder rechtslinearen CFG G kann effektiv ein äquivalenter ϵ -FA A mit $L(G) = L(A)$ konstruiert werden.

Wir geben nur die Idee für die (einfache) Konstruktion:

Definiere Zustandsmenge $Q := \{z_X \mid X \in N\} \cup \{z_e\}$ eines NFA mit einzigm Endzustand z_e .

Zur Produktion $A \rightarrow xB$ definiere Kante (z_A, x, z_B) .

Zur Produktion $A \rightarrow x$ definiere Kante (z_A, x, z_e) .

REG durch einseitig lineare CFG erzeugen

Satz 15.2:

Zu jeder regulären Menge L kann eine rechtslineare CFG G_L konstruiert werden mit $L(G_L) = L$.

Wir geben auch hier nur die Idee der Konstruktion:

Definiere Nonterminalalphabet $N := \{H_p \mid p \in Q\}$ einer rechtslinearen CFG mit Startsymbol H_{z_0} , wobei z_0 Startzustand des zu simulierenden DFA ist.

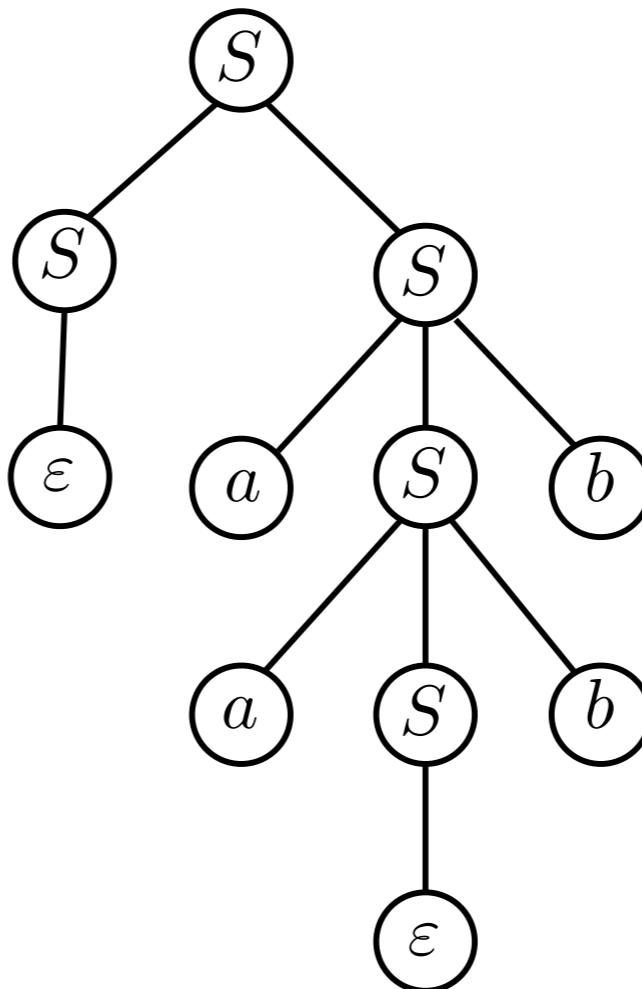
Zur Kante (p, x, q) definiere Produktion $H_p \longrightarrow xH_q$.

Falls q Endzustand, dann noch die Produktion $H_p \longrightarrow x$.

Ableitungsbäume

Als graphische Repräsentationen von Ableitungen einer kontextfreien Grammatik benutzen wir oft die übersichtlicheren Ableitungsbäume.

Ableitungsbau
für $w=aabb$

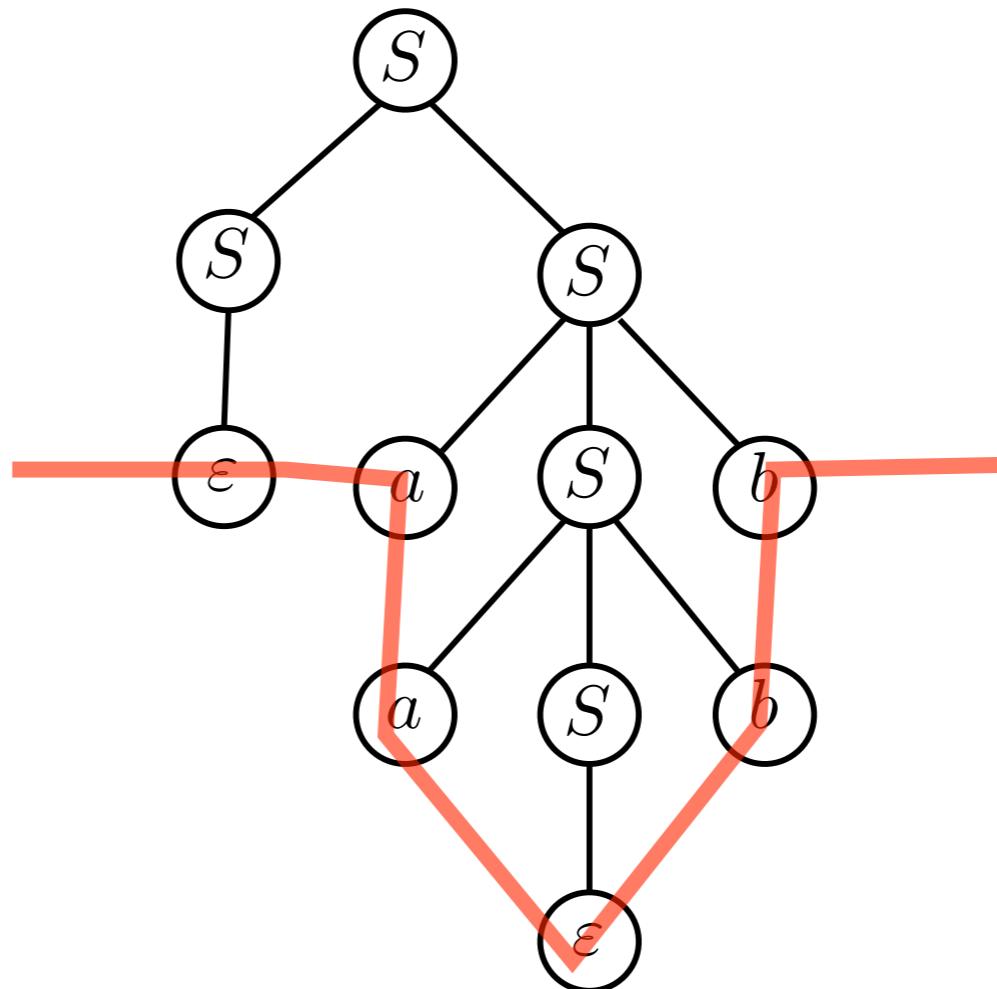


Ein **Baum** ist ein gerichteter, geordneter, zyklenfreier und knotenmarkierter Graph mit einer **Wurzel**, die keinen Vorgängerknoten besitzt.

Ableitungsbäume

Als graphische Repräsentationen von Ableitungen einer kontextfreien Grammatik benutzen wir oft die übersichtlicheren Ableitungsbäume.

Ableitungsbäum
für $w=aabb$



Ein **Baum** ist ein gerichteter, geordneter, zyklenfreier und knotenmarkierter Graph mit einer **Wurzel**, die keinen Vorgängerknoten besitzt.

Ableitungsbaum einer CFG

Definition 15.4: (Ableitungsbaum)

- Sei $G := (\Sigma, N, P, S)$ eine beliebige CFG , dann wird für jedes Symbol $A \in (N \cup \Sigma)$ ein **A -Ableitungsbaum** $B(A)$ wie folgt erklärt:
 1. Jedes Blatt von $B(A)$ wird als Symbol $a \in \Sigma \cup \{\epsilon\}$ in einem Kreis dargestellt.
 2. Jeder Knoten von $B(A)$, der kein Blatt ist, ist mit einem Symbol $H \in N$, die Wurzel mit dem Symbol A beschriftet.
 3. Ein (innerer) Knoten ist mit dem Symbol $H \in N$ beschriftet gdw. seine Nachfolgerknoten von links nach rechts mit H_1, H_2, \dots, H_k beschriftet sind und $H \rightarrow H_1 H_2 \dots H_k$ eine Produktion von G ist.

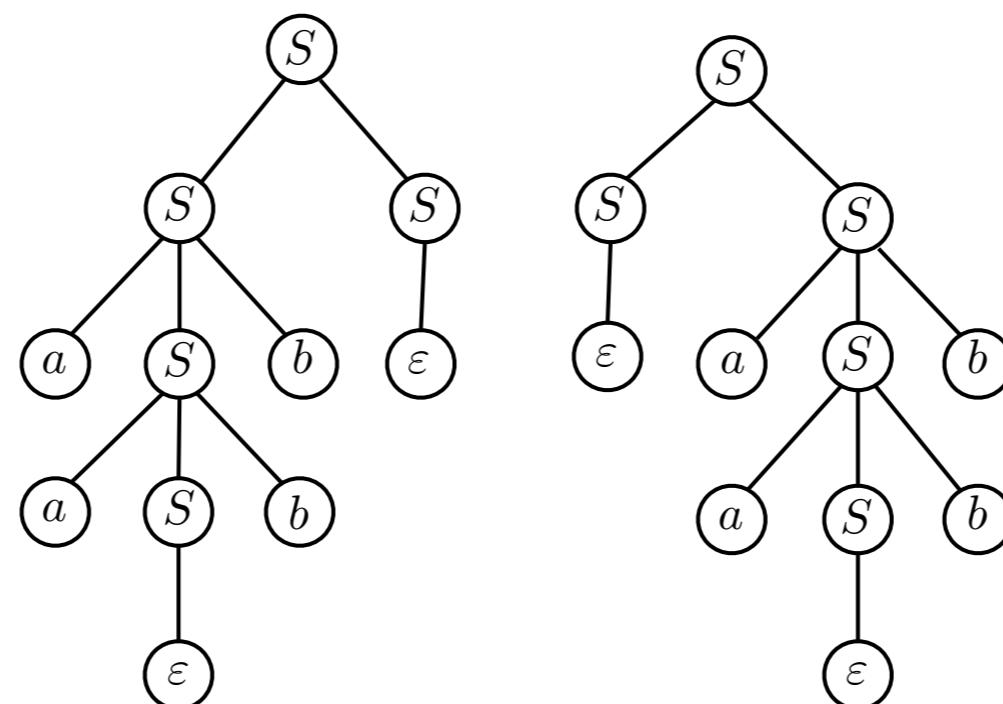
Dyck-Sprache

Beispiel (Dyck-Sprache)

- Betrachte die KFG G_1 mit folgenden Produktionen:

$$S \longrightarrow SS \mid aSb \mid \epsilon$$

- $L(G_1)$ ist die **Dyck-Sprache** D_1 aller korrekt geklammerten Ausdrücke über einem Klammerpaar.
- Zwei unterschiedliche Ableitungsbäume für $aabb$:



mehrdeutige/eindeutige CFG

Definition 15.5: (Mehrdeutigkeit)

- Eine $CFG\ G$ heißt **mehrdeutig** (*ambiguous*) gdw. es für mindestens ein Wort $w \in L(G)$ zwei verschiedene Ableitungsbäume gibt.
Andernfalls heißt G **eindeutig**.
- Eine kontextfreie Sprache $L \in Cf$ heißt **eindeutig** (*unambiguous*) gdw. es eine eindeutige $CFG\ G$ mit $L = L(G)$ gibt. Andernfalls heißt L **mehrdeutig**.

Es gibt beweisbar mehrdeutige Sprachen:

Satz (ohne Beweis):

$$L_M := \{a^r b^s c^t \mid \exists r, s, t \in \mathbb{N} : (r = s) \vee (s = t)\}$$

ist *nicht* eindeutig.

D. h. es gibt **keine** eindeutige CFG , die L_M erzeugt!

FGI 1

Automaten, Formale Sprachen,
Berechenbarkeit

Kap. 15
(Normalformen kontextfreier
Grammatiken)

Michael Köhler-Bußmeier

Nutzlose Nonterminale

Definition 15.6:

- Eine kontextfreie Grammatik $G := (\Sigma, N, P, S)$ heißt genau dann **reduziert**, wenn gilt:
 1. Jedes Nonterminal ist **produktiv**, d.h.
 $\forall A \in N$ gilt $\exists w \in \Sigma^* : A \xrightarrow[G]^* w.$
 2. Jedes Nonterminal ist **erreichbar**, d.h.
 $\forall A \in N$ gilt $\exists u, v \in (N \cup \Sigma)^* : S \xrightarrow[G]^* uAv.$
- Eine reduzierte Grammatik enthält möglicherweise trotzdem noch unnötig viele Symbole!

reduzierte CFG

- $S \rightarrow A, A \rightarrow B, B \rightarrow aBa \mid bBb \mid \epsilon$ ist reduziert.
- Die Grammatik hat fünf Regeln und drei Non-terminalen.
- Die äquivalente Grammatik mit nur drei Regeln $S \rightarrow aSa \mid bSb \mid \epsilon$ benötigt nur ein Nonterminal!

Zunächst trotzdem ein Ergebnis zu der Existenz reduzierter Grammatiken:

Satz 15.3:

Zu jeder CFG G kann effektiv eine äquivalente reduzierte CFG G' konstruiert werden.

Konstruktion reduzierter CFG

Die Konstruktion besteht aus zwei separaten Teilen:

- **Teil 1** entfernt Symbole (und Produktionen), so dass danach alle verwendeten Nonterminale auf reine Terminalwörter abgeleitet werden können, d.h. *G enthält danach nur noch produktive Nonterminal-symbole.*
- **Teil 2** entfernt danach alle Symbole, die vom Startsymbol aus *nicht* erreichbar sind.
- Die Reihenfolge der Anwendung dieser Schritte ist wichtig!

Das Verfahren (1)

Erzeuge Grammatik mit produktiven Nonterminalen:

Teil 1 der Konstruktion:

Zu gegebener $CFG\ G := (\Sigma, N, P, S)$ definieren wir Mengen $M_i \subseteq (N \cup \Sigma)$ durch:

1. $M_0 := \Sigma$
2. $M_{i+1} := M_i \cup \{A \in N \mid \exists w \in M_i^* : A \longrightarrow w \in P\}.$
3. Da die Menge $M_i \subseteq N$ für jedes $i \in \mathbb{N}$ endlich ist, gibt es einen Index k mit $M_k = M_{k+1}$.
4. M_k enthält dann offensichtlich nur produktive Symbole.
5. Bilde neue $CFG\ G'$ mit $L(G') = L(G)$ durch
$$G' = (\Sigma', N', P', S') := (\Sigma, \{S\} \cup (M_k \cap N), P \cap (M_k \cap M_k^*), S).$$

Das Verfahren (2)

Erzeuge Grammatik mit erreichbaren Nonterminalen:

Teil 2 der Konstruktion:

Es werden alle erreichbaren Symbole in G' bestimmt:

1. $M_0 := \{S'\}.$
2. $M_{i+1} := M_i \cup \{B \in N' \mid \exists A \in M_i \exists u, v \in (N' \cup \Sigma)^*: A \longrightarrow uBv \in P'\}$
3. Wieder gilt $\exists k \in \mathbb{N} : M_k = M_{k+1}.$
4. M_k ist die Menge aller erreichbaren Nonterminalen in G' .
5. Definiere $G'' := (\Sigma'', N', P'', S'')$, mit Produktionen, deren rechte und linke Seiten nur mit erreichbaren Nonterminalen gebildet werden.
6. $N'' := M_k$, $P'' := P' \setminus \{(x, y) \mid x \notin M_k \vee y \notin (M_k \cup \Sigma')^*\}$
7. Die Menge Σ'' der erreichbaren Terminalzeichen kann kleiner werden.

Anwendungen (richtig und falsch)

- Gegeben: $S \rightarrow AB \mid \epsilon, \quad A \rightarrow a$
- 1. \rightarrow 2.
 - B ist unproduktiv, $S \rightarrow AB$ wird gestrichen.
 - Das Nonterminal A ist nicht erreichbar, also wird $A \rightarrow a$ gestrichen.
 - Resultat: $S \rightarrow \epsilon$
- 2. \rightarrow 1.
 - Alle Nonterminale sind erreichbar, somit wird keines gestrichen.
 - B ist unproduktiv, $S \rightarrow AB$ wird gestrichen.
 - Resultat: $S \rightarrow \epsilon$ und $A \rightarrow a$
- Beide Grammatiken erzeugen nur ϵ . Die zweite ist nicht reduziert!

ϵ -freie Grammatiken

Eine Grammatik $G = (\Sigma, N, P, S)$ heißt ϵ -frei, wenn sie keine Produktion der Form $A \rightarrow \epsilon$ besitzt.

Aussage 1 Zu jeder Grammatik $G = (\Sigma, N, P, S)$ kann effektiv eine ϵ -freie Grammatik $G' = (\Sigma, N', P', S')$ konstruiert werden, für die gilt:

- Gilt $\epsilon \notin L(G)$, dann ist $L(G') = L(G)$.
- Gilt $\epsilon \in L(G)$, dann ist $L(G') = L(G) \setminus \{\epsilon\}$.

Gilt $\epsilon \in L(G)$, dann konstruieren wir aus G' eine Grammatik G'' , indem wir ein neues Nonterminal S'' als Startsymbol verwenden und folgende Produktionen hinzufügen:

$$S'' \rightarrow \epsilon \mid S'$$

Dann gilt $L(G'') = L(G)$ und da $S'' \rightarrow \epsilon$ die einzige ϵ -Produktion in G'' ist, ist ϵ das *einzige* Wort, für das eine Ableitung mit ϵ -Produktion existiert.

Welche Nonterminale sind auf ϵ ableitbar?

Bestimmung der auf ϵ ableitbaren Nonterminale:

- Für jedes $A \in N$ entscheiden wir, ob $A \xrightarrow[G]{}^* \epsilon$ gilt. Bestimmen der Menge $N_\epsilon \subseteq N$:
- $M_0 := \{A \in N \mid A \longrightarrow \epsilon \in P\}$ und
 $M_{i+1} := \{A \in N \mid \exists w \in M_i^* : A \longrightarrow w \in P\} \cup M_i$
- Es existiert ein Index $k \in \mathbb{N}$ mit $M_k = M_{k+1}$ und es ist $N_\epsilon := M_k$ die gesuchte Menge.

Ist $A, B, C \in N$ und $A, B \in N_\epsilon$, so kann jedes der Wörter ABC, BC, AC , oder C aus ABC abgeleitet werden.

Das gleiche Ergebnis erzielt die Substitution σ , durch folgende Zuordnung:

$A \mapsto \{A, \epsilon\}$, $B \mapsto \{B, \epsilon\}$ und $C \mapsto \{C\}$. Es gilt hier:

$$\sigma(ABC) = \{ABC, BC, AC, C\}$$

Elimination von ϵ -Produktionen

Beispiel: ϵ -frei-machen

- $S \rightarrow ABC \mid AB,$
 $A \rightarrow aaaA \mid B,$
 $B \rightarrow bB \mid \epsilon,$
 $C \rightarrow AC \mid BC$

- $M_0 = \{B\}$, $M_1 = \{B, A\}$, und $M_3 = \{B, A, S\}$ ergibt
 $V_\epsilon = \{S, A, B\}.$

- Anwenden der Substitution aus der Konstruktion:
 $S \rightarrow ABC \mid AB \mid AC \mid BC \mid C \mid A \mid B \mid \epsilon,$
 $A \rightarrow aaaA \mid B \mid \epsilon \mid aaa,$
 $B \rightarrow bB \mid \epsilon \mid b,$
 $C \rightarrow AC \mid BC \mid C$

Elimination von ϵ -Produktionen

Beispiel: ϵ -frei-machen

- $S \rightarrow ABC \mid AB,$
 $A \rightarrow aaaA \mid B,$
 $B \rightarrow bB \mid \epsilon,$
 $C \rightarrow AC \mid BC$
- $M_0 = \{B\}$, $M_1 = \{B, A\}$, und $M_3 = \{B, A, S\}$ ergibt
 $V_\epsilon = \{S, A, B\}.$

- Anwenden der Substitution aus der Konstruktion:

$$\begin{aligned}S &\rightarrow ABC \mid AB \mid AC \mid BC \mid C \mid A \mid B \mid \epsilon, \\A &\rightarrow aaaA \mid B \mid \epsilon \mid aaa, \\B &\rightarrow bB \mid \epsilon \mid b, \\C &\rightarrow AC \mid BC \mid C\end{aligned}$$

Elimination von ϵ -Produktionen

Beispiel: ϵ -frei-machen

- $S \rightarrow ABC \mid AB,$
 $A \rightarrow aaaA \mid B,$
 $B \rightarrow bB \mid \epsilon,$
 $C \rightarrow AC \mid BC$

- $M_0 = \{B\}$, $M_1 = \{B, A\}$, und $M_3 = \{B, A, S\}$ ergibt
 $V_\epsilon = \{S, A, B\}.$

- Anwenden der Substitution aus der Konstruktion:

$$\begin{aligned}S &\rightarrow ABC \mid AB \mid AC \mid BC \mid C \mid A \mid B \mid \epsilon, \\A &\rightarrow aaaA \mid B \mid \epsilon \mid aaa, \\B &\rightarrow bB \mid \epsilon \mid b, \\C &\rightarrow AC \mid BC \mid C\end{aligned}$$

Elimination von ϵ -Produktionen

Beispiel: ϵ -frei-machen

- $S \rightarrow ABC \mid AB,$
 $A \rightarrow aaaA \mid B,$
 $B \rightarrow bB \mid \epsilon,$
 $C \rightarrow AC \mid BC$

- $M_0 = \{B\}$, $M_1 = \{B, A\}$, und $M_3 = \{B, A, S\}$ ergibt
 $V_\epsilon = \{S, A, B\}.$

- Anwenden der Substitution aus der Konstruktion:

$$\begin{aligned} S &\rightarrow ABC \mid AB \mid AC \mid BC \mid C \mid A \mid B \cancel{\mid \epsilon}, \\ A &\rightarrow aaaA \mid B \cancel{\mid \epsilon} \mid aaa, \\ B &\rightarrow bB \cancel{\mid \epsilon} \mid b, \\ C &\rightarrow AC \mid BC \mid C \end{aligned}$$

Elimination von ϵ -Produktionen

Beispiel: ϵ -frei-machen

- $S \rightarrow ABC \mid AB,$
 $A \rightarrow aaaA \mid B,$
 $B \rightarrow bB \mid \epsilon,$
 $C \rightarrow AC \mid BC$

- $M_0 = \{B\}$, $M_1 = \{B, A\}$, und $M_3 = \{B, A, S\}$ ergibt
 $V_\epsilon = \{S, A, B\}.$

- Anwenden der Substitution aus der Konstruktion:

$$S \rightarrow ABC \mid AB \mid AC \mid BC \mid C \mid A \mid B \cancel{\mid \epsilon},$$

$$A \rightarrow aaaA \mid B \cancel{\mid \epsilon} \mid aaa,$$

$$B \rightarrow bB \cancel{\mid \epsilon} \mid b,$$

$$C \rightarrow AC \mid BC \mid C$$

*G' := Substitution +
 ϵ -Produktionen
streichen*

Chomsky-Normalform (CNF)

- Es ist manchmal unhandlich, viele mögliche Gestalten *rechter Seiten von Regeln* haben zu können.
- Auf den *linken Seiten* steht bei CFGs sowieso immer *genau ein* Nonterminal.
- **Normalformen** helfen bei Konstruktionen und Beweisen.

Definition 15.7:

Eine $CFG\ G := (\Sigma, N, P, S)$ ist in **Chomsky-Normalform (CNF)** gdw. alle Produktionen in P von folgender Form sind:
 $A \rightarrow BC$ oder $A \rightarrow a$ für $A, B, C \in N$ und $a \in \Sigma$.

Greibach-Normalform (GNF)

Definition 15.8:

Eine CFG $G := (\Sigma, N, P, S)$ ist in **Greibach-Normalform** gdw. $P \subseteq N \times \Sigma \cdot N^*$.

Normalformen sind nur sinnvoll, wenn sie für eine genügend große Menge von Grammatiken existieren.

Demnächst:

- (i) Konstruktion einer CNF für beliebige CFG
- (ii) Eigenschaften der kontextfreien Sprachen
- (iii) Grenzen der kontextfreien Sprachen

Satz zur Greibach-Normalform

Satz 15.4:

Zu jeder $CFG\ G$ kann effektiv eine äquivalente $CFG\ G' := (\Sigma, N', P', S')$ konstruiert werden, für die folgendes gilt:

- Falls $\epsilon \notin L(G)$, so ist G' in Greibach-Normalform.
- Gilt $\epsilon \in L(G)$, so ist $S' \rightarrow \epsilon$ einzige ϵ -Produktion in P' . Die Produktionen in $P' \setminus \{S' \rightarrow \epsilon\} \subseteq N' \times (\Sigma \cdot N'^*)$ sind in GNF . (S' nicht auf rechter Seite!)

Beweis:

Dieser wird hier weggelassen, siehe Vossen & Witt, Satz 6.3 und Seite 204!

Beispiele zu CNF und GNF

- $S \rightarrow SS \mid AB, A \rightarrow AA \mid a, B \rightarrow BB \mid b$ ist in Chomsky-Normalform, aber nicht in Greibach-Normalform.
- $S \rightarrow aA, A \rightarrow aA \mid aB, B \rightarrow bB \mid b$ ist in Greibach-Normalform, jedoch nicht in Chomsky-Normalform.
- $S \rightarrow aAbB, A \rightarrow aA \mid \epsilon, B \rightarrow bB \mid \epsilon$ ist weder in Greibach- noch in Chomsky-Normalform.
- Wie sieht es mit $S \rightarrow aA, aA \rightarrow aaA \mid abB, bB \rightarrow bbB \mid b$ aus?

Diese Grammatik ist nicht einmal kontextfrei!

- $S \rightarrow a \mid b$ ist sowohl in CNF, als auch in GNF!

Satz zur Chomsky-Normalform

Satz 15.5:

Zu jeder $CFG\ G$ kann effektiv eine äquivalente $CFG\ G' := (\Sigma', N', P', S')$ konstruiert werden, für die folgendes gilt:

- Falls $\epsilon \notin L(G)$, so ist G' in Chomsky-Normalform.
- Gilt $\epsilon \in L(G)$, so ist $S' \rightarrow \epsilon$ einzige ϵ -Produktion in P' . Die Produktionen in $P' \setminus \{S' \rightarrow \epsilon\} \subseteq N' \times (N' \cdot N' \cup \Sigma')$ sind in CNF . (S' nicht auf rechter Seite!)

Beweis:

In mehreren Schritten werden äquivalente Grammatiken erstellt

...

Konstruktionsplan zur Chomsky-Normalform

Die Konstruktion erfolgt in sechs Schritten:

1. ϵ -frei-machen
2. Reduzieren
3. Kettenregeln entfernen
4. Ersetzen langer Terminalregeln
5. Verkürzen zu langer Regeln
6. Wiederherstellen der ursprünglichen Sprache durch evtl. Hinzunahme einer ϵ -Regel

1. Elimination von ϵ -Produktionen

$M_0 = \{B\}$, $M_1 = \{B, A\}$, und $M_3 = \{B, A, S\}$ ergibt
 $V_\epsilon = \{S, A, B\}$.

Beispiel: ϵ -frei-machen

- $S \rightarrow ABC \mid AB,$
 $A \rightarrow aaaA \mid B,$
 $B \rightarrow bB \mid \epsilon,$
 $C \rightarrow AC \mid BC$

- Löschen der ϵ -Regeln:

$$\begin{aligned} S &\rightarrow ABC \mid AB \mid AC \mid BC \mid C \mid A \mid B, \\ A &\rightarrow aaaA \mid B \mid aaa, \\ B &\rightarrow bB \mid b, \\ C &\rightarrow AC \mid BC \mid C \end{aligned}$$

- Es gilt $L(G_{neu}) = L(G_{alt}) \setminus \{\epsilon\}$.

Setze $G_1 := G_{neu}$ und setze die Konstruktion mit G_1 fort:
Reduzieren von G_1 zu G_2

2. Reduziere die ϵ -freie CFG G_1

- G_1 enthielt die Regeln $S \rightarrow ABC \mid AB \mid AC \mid BC \mid C \mid A \mid B$,
 $A \rightarrow aaaA \mid B \mid aaa$,
 $B \rightarrow bB \mid b$,
 $C \rightarrow AC \mid BC \mid C$
- A , B , und S sind produktiv:
 $S \rightarrow AB \mid A \mid B$,
 $A \rightarrow aaaA \mid B \mid aaa$,
 $B \rightarrow bB \mid b$
- Nun sind alle Nonterminale auch noch erreichbar.
Also sind wir mit dem 2. Schritt fertig.

C ist reichlich nutzlos, da es nie verschwinden kann!

3. Kettenregeln entfernen

- In G_2 kann es noch Produktionen der Form
$$A \longrightarrow B \in N \times N$$
 geben.
- Wir definieren die Relation $\ll \subseteq N \times N$ mit: $A \ll B$ gdw.
$$A \longrightarrow B \in P_2.$$
A \rightarrow B ist (Ketten-)Regel !
- $G_3 := (\Sigma, N_3, P_3, S)$ mit $N_3 := N_2$ und $P_3 := \{A \longrightarrow w \mid w \notin N_3 \wedge \exists B \longrightarrow w \in P_2 \wedge A \overset{*}{\ll} B\}$ B ist aus A über Kettenregeln erreichbar!
- Das Startsymbol bleibt S , denn sollte $L(G) = \emptyset$ gelten, so war bereits $P_2 = \emptyset$.
- Die Gleichheit von $L(G_3)$ und $L(G_2)$ zeigt man auch leicht formal.

Kettenregeln am Beispiel:

- $S \rightarrow AB \mid A \mid B,$
 $A \rightarrow aaaA \mid B \mid aaa,$
 $B \rightarrow bB \mid b$
- $S \ll A, S \ll B$ und $A \ll B$
- $S \rightarrow AB \mid aaaA \mid aaa \mid bB \mid b,$
 $A \rightarrow aaaA \mid bB \mid b \mid aaa,$
 $B \rightarrow bB \mid b$
- Somit sind alle Kettenregeln beseitigt!

Nun sind die rechten Seiten der Regeln
vielleicht noch zu lang...

4. Ersetzen langer Terminalregeln

- Terminalsymbole dürfen in Chomsky-Normalform nur durch Produktionen der Form $A \longrightarrow a$ generiert werden.
- In allen rechten Seiten, deren Länge größer 1 ist, ersetzen wir jedes Terminal a durch ein neues Nonterminal $\langle a \rangle$.
- Wir erweitern P_3 zu P_4 durch Hinzunahme der Produktionen $\langle a \rangle \longrightarrow a$.
- Wir erhalten so im vierten Schritt $G_4 := (\Sigma, N_4, P_4, S)$.
- $S \longrightarrow AB \mid \langle a \rangle \langle a \rangle \langle a \rangle A \mid \langle a \rangle \langle a \rangle \langle a \rangle \mid \langle b \rangle B \mid b,$
 $A \longrightarrow \langle a \rangle \langle a \rangle \langle a \rangle A \mid \langle b \rangle B \mid b \mid \langle a \rangle \langle a \rangle \langle a \rangle,$
 $B \longrightarrow \langle b \rangle B \mid b,$
 $\langle a \rangle \longrightarrow a, \quad \langle b \rangle \longrightarrow b$

5. Verkürzen zu langer Regeln

- Es kann noch Regeln $A \rightarrow w$ mit $|w| > 2$ geben.
- Neue Nonterminale $\langle v \rangle$ für jeden echten Präfix v der rechten Seiten in P_4 mit $|v| \geq 2$:

$G_5 := (\Sigma, N_5, P_5, S)$ mit

$$N_5 := \left\{ \langle v \rangle \mid |v| \geq 2, \exists u \neq \epsilon \exists A \rightarrow w \in P_4 : w = vu \right\} \cup N_4 \text{ und } P_5 :=$$

$$\{A \rightarrow \langle v \rangle x \mid A \rightarrow w \in P_4 : |w| \geq 3 \wedge w = vx \wedge x \in N_4\} \cup$$

$$\{\langle v \rangle \rightarrow \langle u \rangle y \mid \langle u \rangle, \langle v \rangle \in (N_5 \setminus N_4) \wedge y \in N_4 \wedge v = uy\} \cup$$

$$\{\langle v \rangle \rightarrow xy \mid \langle v \rangle \in (N_5 \setminus N_4) \wedge x, y \in N_4 \wedge v = xy\} \cup$$

$$\{A \rightarrow w \mid A \rightarrow w \in P_4 \wedge |w| \leq 2\}$$

- G_5 ist in Chomsky-Normalform!!
- $L(G_5) = L(G)$, falls $\epsilon \notin L(G)$ gilt.

Beispiel zum Verkürzen

- $S \rightarrow AB \mid \langle a \rangle \langle a \rangle \langle a \rangle A \mid \langle a \rangle \langle a \rangle \langle a \rangle \mid \langle b \rangle B \mid b,$
 $A \rightarrow \langle a \rangle \langle a \rangle \langle a \rangle A \mid \langle b \rangle B \mid b \mid \langle a \rangle \langle a \rangle \langle a \rangle,$
 $B \rightarrow \langle b \rangle B \mid b,$
 $\langle a \rangle \rightarrow a,$
 $\langle b \rangle \rightarrow b$
- . . . wird zu:
 - $S \rightarrow AB \mid \langle\langle a \rangle \langle a \rangle \langle a \rangle \rangle A \mid \langle\langle a \rangle \langle a \rangle \rangle \langle a \rangle \mid \langle b \rangle B \mid b,$
 - $A \rightarrow \langle\langle a \rangle \langle a \rangle \langle a \rangle \rangle A \mid \langle b \rangle B \mid b \mid \langle\langle a \rangle \langle a \rangle \rangle \langle a \rangle,$
 - $B \rightarrow \langle b \rangle B \mid b,$
 - $\langle a \rangle \rightarrow a,$
 - $\langle b \rangle \rightarrow b,$
 - $\langle\langle a \rangle \langle a \rangle \langle a \rangle \rangle \rightarrow \langle\langle a \rangle \langle a \rangle \rangle \langle a \rangle,$
 - $\langle\langle a \rangle \langle a \rangle \rangle \rightarrow \langle a \rangle \langle a \rangle$

Wiederherstellen von Epsilon

- Im ersten Schritt wurde die Menge N_ϵ zur Ausgangsgrammatik G bestimmt.
- Dort konnte also die Frage „ $\epsilon \in L(G)$?“ auf die Frage „ $S \in N_\epsilon$?“ reduziert, und damit entschieden werden.
- Gilt also $\epsilon \in L(G)$, so konstruieren wir G_6 , indem wir zu den Nonterminalen von G_5 ein neues Startsymbol S_{neu} , sowie die neuen Produktionen

$$S_{\text{neu}} \longrightarrow \epsilon \text{ und } \{S_{\text{neu}} \longrightarrow w \mid \exists S \longrightarrow w \in P_5\}$$

zu P_5 hinzufügen. (Nur erforderlich, wenn S in rechten Seiten von Produktionen vorkommt!)

Am Beispiel: Es werden die Regeln $S_{\text{neu}} \longrightarrow \epsilon$, $S_{\text{neu}} \longrightarrow AB \mid \langle\langle a \rangle\langle a \rangle\langle a \rangle\rangle A \mid \langle\langle a \rangle\langle a \rangle\rangle\langle a \rangle \mid \langle b \rangle B \mid b$ hinzugefügt.

FGI 1

Automaten, Formale Sprachen,
Berechenbarkeit

Kap. 16

Pumping-Lemma, Abschlusseigenschaften
und Entscheidbarkeit

Michael Köhler-Bußmeier

Grenzen der CFG ?

- Wo sind die Grenzen von $\mathcal{C}f$?
- Was für Abschlusseigenschaften hat $\mathcal{C}f$?
- Was für Entscheidbarkeitsresultate kennen wir?
- Gibt es ein Automatenmodell für $\mathcal{C}f$?

Zur Beantwortung der ersten Frage benötigen wir Eigenschaften, die für jede kontextfreie Sprache gelten und für formale Sprachen leicht falsifizierbar sind! Einige **Abschlusseigenschaften** und **Entscheidbarkeitsfragen** sind schon bekannt, und auch die letzte Frage wird positiv beantwortet werden!

Abschlusseigenschaften der Familie Cf

- + Vereinigung
- + Durchnittsbildung mit regulären Mengen
- Durchnittsbildung
- Komplement
- + Konkatenation (richtiger: Komplexprodukt)
- + Kleeneabschluss (+-Bildung/*-Bildung)
- + Homomorphismen
- + kontextfreie Substitutionen

Einfacher Beweis mit
Kellerautomaten!

Es gibt noch einige Abschlusseigenschaften mehr,
aber diese sind die wichtigsten!

Im Einzelnen:

Satz 16.1:

Sind $L_1, L_2 \in \mathcal{Cf}$ so sind auch die Sprachen $L_3 := L_1 \cup L_2$, $L_4 := L_1 \cdot L_2$, $L_5 := L_1^+$ und $L_6 := L_1^*$ kontextfrei.

Beweis:

Seien $G_i := (\Sigma_i, N_i, P_i, S_i)$, $i \in \{1, 2\}$, CFG's mit $L_i = L(G_i)$.

$G_k := (\Sigma_1 \cup \Sigma_2, N_1 \cup N_2, P_k, S_k)$, $k \in \{3, 4, 5, 6\}$, erzeugt $L_1 \cup L_2$, $L_1 \cdot L_2$, L_1^+ und L_1^* mit passender Regelmenge P_k :

Im Einzelnen:

Satz 16.1:

Sind $L_1, L_2 \in \mathcal{Cf}$ so sind auch die Sprachen $L_3 := L_1 \cup L_2$, $L_4 := L_1 \cdot L_2$, $L_5 := L_1^+$ und $L_6 := L_1^*$ kontextfrei.

Beweis:

Seien $G_i := (\Sigma_i, N_i, P_i, S_i)$, $i \in \{1, 2\}$, CFG's mit $L_i = L(G_i)$.

$G_k := (\Sigma_1 \cup \Sigma_2, N_1 \uplus N_2, P_k, S_k)$, $k \in \{3, 4, 5, 6\}$, erzeugt $L_1 \cup L_2$, $L_1 \cdot L_2$, L_1^+ und L_1^* mit passender Regelmenge P_k :

1. $L(G_3) = L_1 \cup L_2$: $P_3 := P_1 \uplus P_2 \uplus \{S_3 \rightarrow S_1 \mid S_2\}$,

Im Einzelnen:

Satz 16.1:

Sind $L_1, L_2 \in \mathcal{Cf}$ so sind auch die Sprachen $L_3 := L_1 \cup L_2$, $L_4 := L_1 \cdot L_2$, $L_5 := L_1^+$ und $L_6 := L_1^*$ kontextfrei.

Beweis:

Seien $G_i := (\Sigma_i, N_i, P_i, S_i)$, $i \in \{1, 2\}$, CFG's mit $L_i = L(G_i)$.

$G_k := (\Sigma_1 \cup \Sigma_2, N_1 \uplus N_2, P_k, S_k)$, $k \in \{3, 4, 5, 6\}$, erzeugt $L_1 \cup L_2$, $L_1 \cdot L_2$, L_1^+ und L_1^* mit passender Regelmenge P_k :

1. $L(G_3) = L_1 \cup L_2$: $P_3 := P_1 \uplus P_2 \uplus \{S_3 \rightarrow S_1 \mid S_2\}$,
2. $L(G_4) = L_1 \cdot L_2$: $P_4 := P_1 \uplus P_2 \uplus \{S_4 \rightarrow S_1 S_2\}$,

Im Einzelnen:

Satz 16.1:

Sind $L_1, L_2 \in \mathcal{Cf}$ so sind auch die Sprachen $L_3 := L_1 \cup L_2$, $L_4 := L_1 \cdot L_2$, $L_5 := L_1^+$ und $L_6 := L_1^*$ kontextfrei.

Beweis:

Seien $G_i := (\Sigma_i, N_i, P_i, S_i)$, $i \in \{1, 2\}$, CFG's mit $L_i = L(G_i)$.

$G_k := (\Sigma_1 \cup \Sigma_2, N_1 \uplus N_2, P_k, S_k)$, $k \in \{3, 4, 5, 6\}$, erzeugt $L_1 \cup L_2$, $L_1 \cdot L_2$, L_1^+ und L_1^* mit passender Regelmenge P_k :

1. $L(G_3) = L_1 \cup L_2$: $P_3 := P_1 \uplus P_2 \uplus \{S_3 \rightarrow S_1 \mid S_2\}$,
2. $L(G_4) = L_1 \cdot L_2$: $P_4 := P_1 \uplus P_2 \uplus \{S_4 \rightarrow S_1 S_2\}$,
3. $L(G_5) = L_1^+$: $P_5 := P_1 \uplus \{S_5 \rightarrow S_1 S_5 \mid S_1\}$,

Im Einzelnen:

Satz 16.1:

Sind $L_1, L_2 \in \mathcal{Cf}$ so sind auch die Sprachen $L_3 := L_1 \cup L_2$, $L_4 := L_1 \cdot L_2$, $L_5 := L_1^+$ und $L_6 := L_1^*$ kontextfrei.

Beweis:

Seien $G_i := (\Sigma_i, N_i, P_i, S_i)$, $i \in \{1, 2\}$, CFG's mit $L_i = L(G_i)$.

$G_k := (\Sigma_1 \cup \Sigma_2, N_1 \uplus N_2, P_k, S_k)$, $k \in \{3, 4, 5, 6\}$, erzeugt $L_1 \cup L_2$, $L_1 \cdot L_2$, L_1^+ und L_1^* mit passender Regelmenge P_k :

1. $L(G_3) = L_1 \cup L_2$: $P_3 := P_1 \uplus P_2 \uplus \{S_3 \rightarrow S_1 \mid S_2\}$,
2. $L(G_4) = L_1 \cdot L_2$: $P_4 := P_1 \uplus P_2 \uplus \{S_4 \rightarrow S_1 S_2\}$,
3. $L(G_5) = L_1^+$: $P_5 := P_1 \uplus \{S_5 \rightarrow S_1 S_5 \mid S_1\}$,
4. $L(G_6) = L_1^*$: $P_6 := P_2 \uplus \{S_6 \rightarrow S_2 S_6 \mid \epsilon\}$

fehlende Abschlusseigenschaften bei Cf

- Die Familie der kontextfreien Sprachen ist **nicht** abgeschlossen gegenüber folgenden Operatoren:
 1. Durchschnittsbildung
 2. Komplementbildung
 3. Bildung einer Mengendifferenz
- Den **Beweis** führen wir indirekt:
 - $L_1 := \{a^n b^n \mid n \in \mathbb{N}\}$ wie auch $L_2 := \{b^m c^m \mid m \in \mathbb{N}\}$ sind kontextfrei.
 - $L_3 := L_1 \cdot \{c\}^*$ und $L_4 := \{a\}^* \cdot L_2$ ist kontextfrei.
 - $L_5 := L_3 \cap L_4 = \{a^n b^n c^n \mid n \in \mathbb{N}\} \in \mathcal{Cf}$. L_5 ist aber nicht kontextfrei!
- 2. und 3. folgen somit direkt!

fehlende Abschlusseigenschaften bei Cf

- Die Familie der kontextfreien Sprachen ist **nicht** abgeschlossen gegenüber folgenden Operatoren:
 1. Durchschnittsbildung
 2. Komplementbildung
 3. Bildung einer Mengendifferenz
- Den **Beweis** führen wir indirekt:
 - $L_1 := \{a^n b^n \mid n \in \mathbb{N}\}$ wie auch $L_2 := \{b^m c^m \mid m \in \mathbb{N}\}$ sind kontextfrei.
 - $L_3 := L_1 \cdot \{c\}^*$ und $L_4 := \{a\}^* \cdot L_2$ ist kontextfrei.
 - $L_5 := L_3 \cap L_4 = \{a^n b^n c^n \mid n \in \mathbb{N}\} \in \mathcal{Cf}$. L_5 ist aber nicht kontextfrei!
- 2. und 3. folgen somit direkt!

Dies zeigt man mit dem uvwxy-Theorem!

fehlende Abschlusseigenschaften bei Cf

- Die Familie der kontextfreien Sprachen ist **nicht** abgeschlossen gegenüber folgenden Operatoren:
 1. Durchschnittsbildung
 2. Komplementbildung
 3. Bildung einer Mengendifferenz
- Den **Beweis** führen wir indirekt:
 - $L_1 := \{a^n b^n \mid n \in \mathbb{N}\}$ wie auch $L_2 := \{b^m c^m \mid m \in \mathbb{N}\}$ sind kontextfrei.
 - $L_3 := L_1 \cdot \{c\}^*$ und $L_4 := \{a\}^* \cdot L_2$ ist kontextfrei.
 - $L_5 := L_3 \cap L_4 = \{a^n b^n c^n \mid n \in \mathbb{N}\} \in \mathcal{Cf}$. L_5 ist aber nicht kontextfrei!
- 2. und 3. folgen somit direkt! warum?

Dies zeigt man mit dem uvwxy-Theorem!

fehlende Abschlusseigenschaften bei Cf

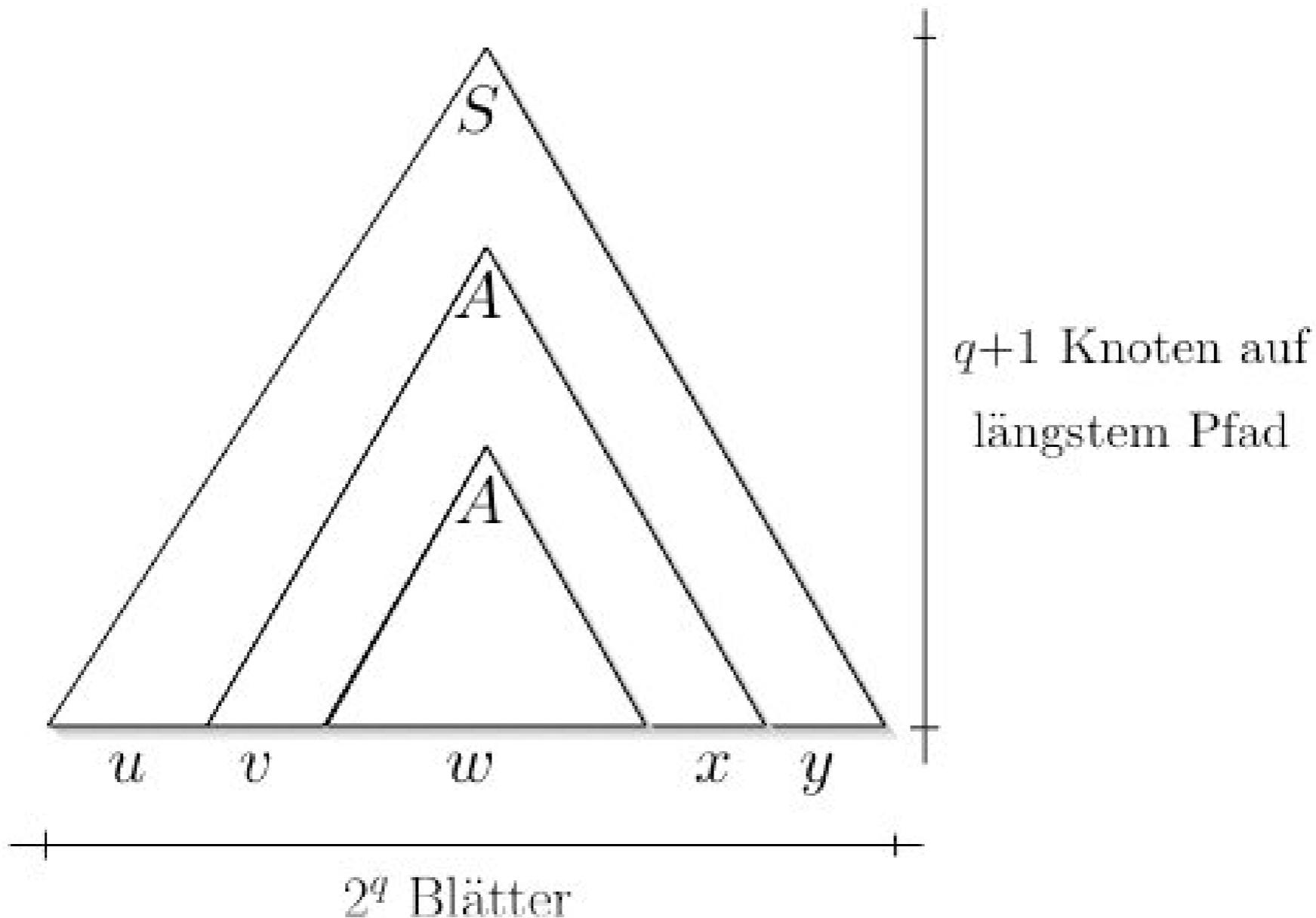
- Die Familie der kontextfreien Sprachen ist **nicht** abgeschlossen gegenüber folgenden Operatoren:
 1. Durchschnittsbildung
 2. Komplementbildung
 3. Bildung einer Mengendifferenz
- Den **Beweis** führen wir indirekt:
 - $L_1 := \{a^n b^n \mid n \in \mathbb{N}\}$ wie auch $L_2 := \{b^m c^m \mid m \in \mathbb{N}\}$ sind kontextfrei.
 - $L_3 := L_1 \cdot \{c\}^*$ und $L_4 := \{a\}^* \cdot L_2$ ist kontextfrei.
 - $L_5 := L_3 \cap L_4 = \{a^n b^n c^n \mid n \in \mathbb{N}\} \in \mathcal{Cf}$. L_5 ist aber nicht kontextfrei!
- 2. und 3. folgen somit direkt!

Dies zeigt man mit dem uvwxy-Theorem!

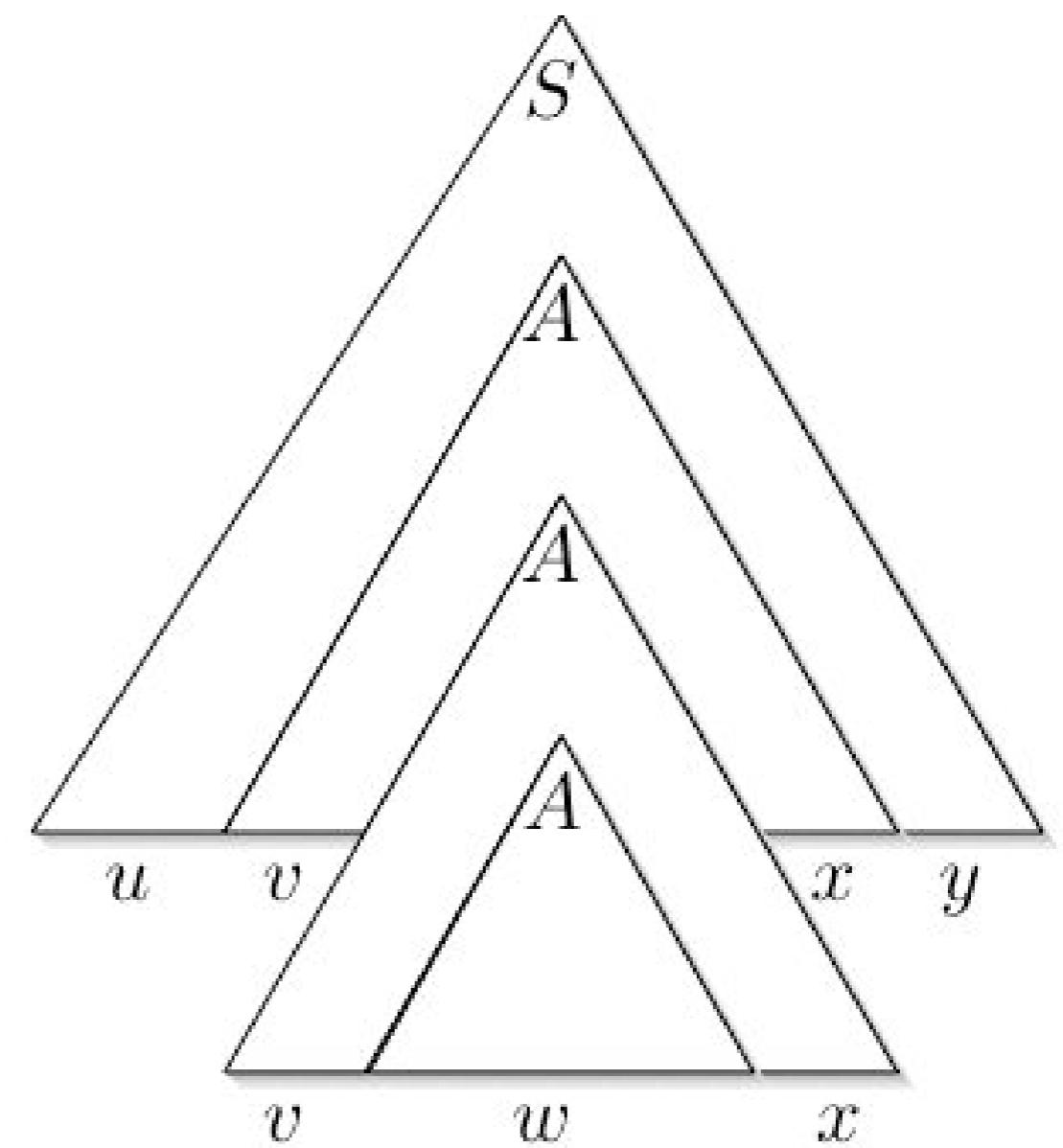
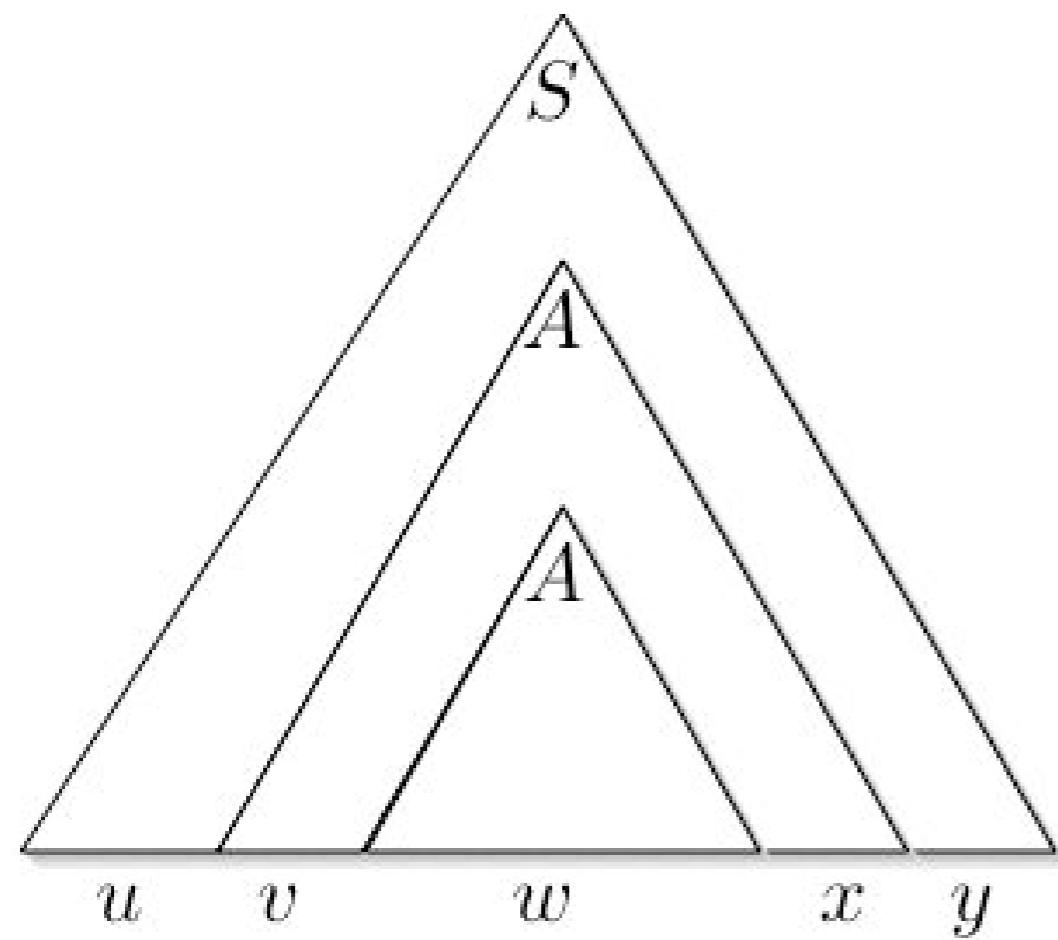
Wie sehen Ableitungen aus?

- Eine kontextfreie Grammatik besitzt nur endlich viele Nonterminale, z.B.: n .
- Ist in einem Ableitungsbaum ein Pfad länger als n , so kommt ein Nonterminal doppelt vor.
- Daraus ergeben sich weitere Ableitungsbäume, die ebenfalls zu Terminalwörtern führen!
- Ableitungsbäume von *CNF*-Grammatiken haben besonders schöne Eigenschaften.

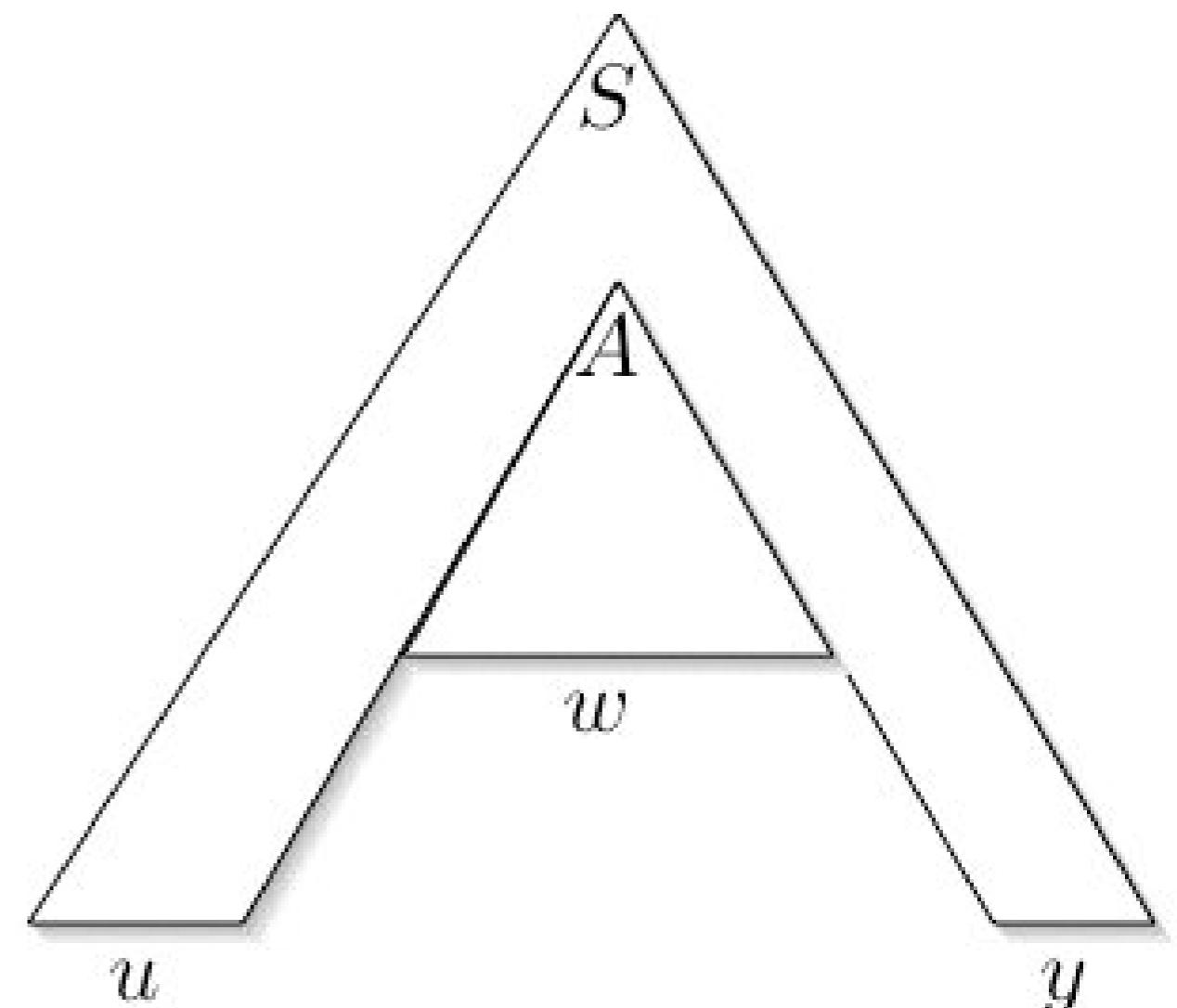
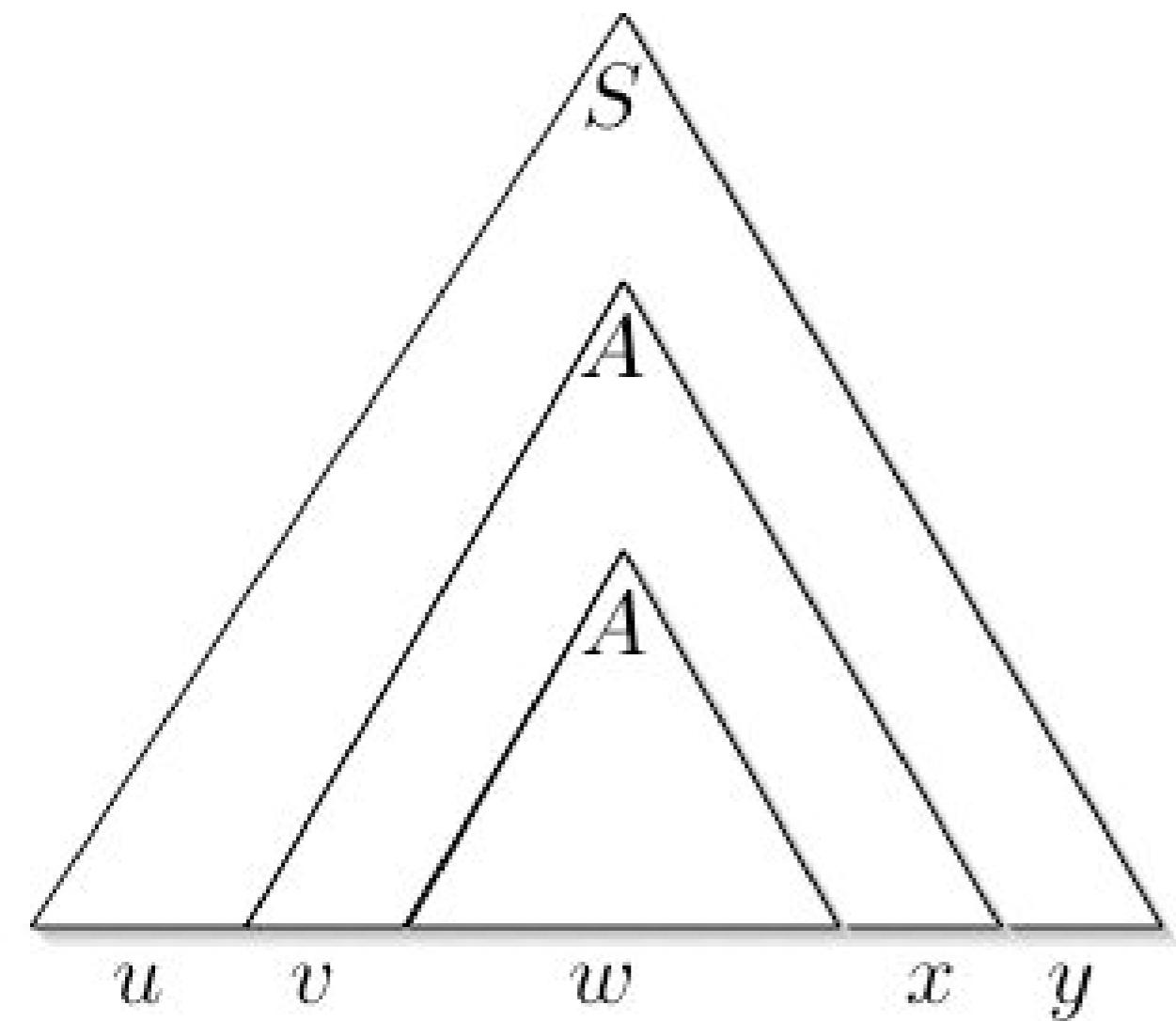
großer Ableitungsbaum



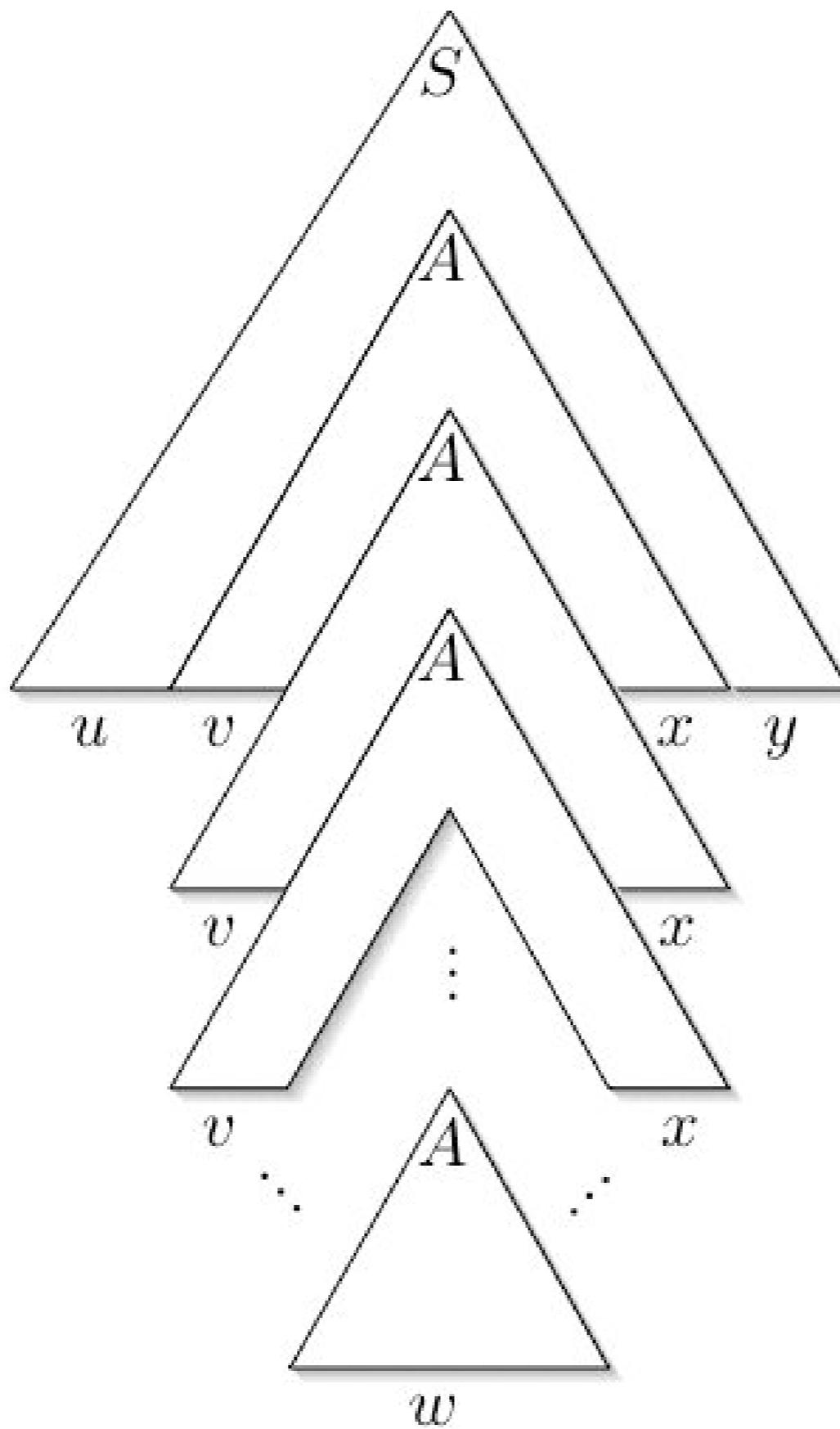
Ableitungsbaum mit Wiederholung



Wiederholung entfernen



Wiederholung öfter einsetzen



uvwxy-Theorem

Satz 16.2:

Für jede kontextfreie Sprache $L \in \mathcal{Cf}$ gibt es eine Zahl $n \in \mathbb{N}$, so dass jedes Wort $z \in L$ mit $|z| \geq n$ eine Zerlegung $z = uvwxy$ besitzt, für die folgendes gilt:

- (i) $|vx| \geq 1$
- (ii) $|vwx| \leq n$
- (iii) $\forall i \geq 0 : uv^iwx^i y \in L$

- Mit dem *uvwxy*-Theorem, kann **NICHT** gezeigt werden, dass eine Sprache kontextfrei ist!!!

Anwendung des uvwxy-Theorems

- $L := \{a^n b^n c^n \mid n \in \mathbb{N}\}$ ist nicht kontextfrei.
- Angenommen L wäre kontextfrei, dann gäbe es ein $k \in \mathbb{N}$ für das die Folgerung aus dem *uvwxy*-Theorem zutrifft.
- Wähle $z := a^k b^k c^k$.
- Da $z \in L$ und $|z| \geq k$ gelten, muss eine Aufspaltung $z = uvwxy$ mit $|vwx| \leq k$ und $|vx| \geq 1$ existieren, so dass $uv^iwx^i y \in L$ für alle $i \in \mathbb{N}$.
- Alle Aufspaltungen führen zum Widerspruch:
 - v oder x enthält a 's und b 's
 - v oder x enthält b 's und c 's
 - v oder x enthält nur a 's, nur b 's oder nur c 's

noch eine Anwendung

Beispiel: $\{a^n b^m c^n d^m \mid n, m \in \mathbb{N}\}$:

Sei k die Zahl aus dem Pumping-Lemma.

Für $z := a^k b^{k+1} c^k d^{k+1}$ gibt es keine Aufspaltung $z = uvwxy$ mit $\forall i \in \mathbb{N} : uv^i w x^i y \in L$.

- $\underbrace{a^k}_{\text{bzw. }} b^{k+1} c^k d^{k+1}$, d.h. vwx besteht nur aus a 's.
- $\underbrace{a^k b^{k+1}}_{\text{bzw. }} c^k d^{k+1}$, d.h. vwx besteht aus a 's und b 's.
- $a^k \underbrace{b^{k+1}}_{\text{bzw. }} c^k d^{k+1}$, d.h. vwx besteht nur aus b 's.
- $a^k b^{k+1} \underbrace{c^k}_{\text{bzw. }} d^{k+1}$, d.h. vwx besteht aus b 's und c 's.
- $a^k b^{k+1} \underbrace{c^k}_{\text{bzw. }} \underbrace{d^{k+1}}_{\text{bzw. }}$, d.h. vwx besteht nur aus c 's.
- $a^k b^{k+1} \underbrace{c^k d^{k+1}}_{\text{bzw. }}$, d.h. vwx besteht aus c 's und d 's.
- $a^k b^{k+1} c^k \underbrace{d^{k+1}}_{\text{bzw. }}$, d.h. vwx besteht nur aus d 's.

Entscheidbarkeit bei Cf !

Satz 16.3:

Das **spezielle Wortproblem** für kontextfreie Sprachen ist entscheidbar, d.h. es gibt ein Verfahren, das zu einer durch eine $CFG\ G$ spezifizierten Sprache $L = L(G)$ für jedes beliebige w feststellt, ob $w \in L$ gilt.

Beweisidee:

O.B.d.A. liege G in GNF vor. Da in einer Ableitung $S \xrightarrow{*} v$ in jedem Schritt ein weiteres Terminal erzeugt wird, brauchen nur die endlich vielen Ableitungen der Länge $|w|$ daraufhin überprüft zu werden, ob eine darunter ist, die das Wort w generiert.

- Das Verfahren benötigt **exponentiellen** Aufwand!
- Dieses Entscheidungsverfahren des **allgemeinen Wortproblems** erfordert zusätzlich die Konstruktion einer äquivalenten GNF .

Das **spezielle Wortproblem** für kontextfreie Sprachen kann auch mit **polynomiellem** zeitlichen Aufwand entschieden werden!
Der Algorithmus von Cocke, Younger und Kasami tut dies.

CYK-Verfahren

- Das Wortproblem für CFG in Chomsky-Normalform kann auch mit *polynomiell*em Aufwand entschieden werden
- Sei $G = (\Sigma, N, P, S)$ in CNF und $w \in \Sigma^*$ mit $w = x_1 \cdots x_n$.
- Wir definieren für alle $0 \leq i \leq j \leq n$ die Mengen $N_{ij} \subseteq N$ von Nonterminalen mit:

$$N_{i,j} := \{A \in N \mid A \xrightarrow{*} x_i \cdots x_j\}$$

- Es gilt $w \in L(G)$ gdw. $S \in N_{1n}$.
- Die Mengen $N_{i,j}$ berechnen wir mit Hilfe der *dynamischen Programmierung*: CYK-Algorithmus (Cocke-Younger-Kasami)

CYK-Verfahren

Rekursionsformel für $N_{i,j} := \{A \in N \mid A \xrightarrow{\uparrow} x_i \cdots x_j\}$:

- Für $i = j$ gilt:

$$N_{i,i} = \{A \mid \text{es gibt eine Produktion } A \rightarrow x_i\}$$

- Für $i < j$ gilt:

$$N_{i,j} = \left\{ \begin{array}{c|c} A & \begin{array}{l} \text{es gibt eine Produktion } A \rightarrow BC \\ \text{und ein } k \text{ mit } i \leq k < j, \\ \text{so dass } B \in N_{i,k} \text{ und } C \in N_{k+1,j} \end{array} \end{array} \right\}$$

Beispiel: CYK-Verfahren

Rekursionsformel für $N_{i,j} := \{A \in N \mid A \xrightarrow{*} x_i \cdots x_j\}$:

$$N_{i,j} = \begin{cases} \{A \mid \text{es gibt eine Produktion } A \rightarrow x_i\}, & i = j \\ \{A \mid \text{es gibt eine Produktion } A \rightarrow BC \wedge \exists k : i \leq k < j \wedge B \in N_{i,k} \wedge C \in N_{k+1,j}\}, & i < j \end{cases}$$

Beispiel: CYK-Verfahren

Rekursionsformel für $N_{i,j} := \{A \in N \mid A \xrightarrow{*} x_i \cdots x_j\}$:

$$N_{i,j} = \begin{cases} \{A \mid \text{es gibt eine Produktion } A \rightarrow x_i\}, & i = j \\ \{A \mid \text{es gibt eine Produktion } A \rightarrow BC \wedge \exists k : i \leq k < j \wedge B \in N_{i,k} \wedge C \in N_{k+1,j}\}, & i < j \end{cases}$$

Beispiel für $w = x_1 \cdots x_5$:

$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,3}$	$N_{2,3}$	$N_{3,3}$		
$N_{1,2}$	$N_{2,2}$			
$N_{1,1}$				

Beispiel: CYK-Verfahren

Rekursionsformel für $N_{i,j} := \{A \in N \mid A \xrightarrow{*} x_i \cdots x_j\}$:

$$N_{i,j} = \begin{cases} \{A \mid \text{es gibt eine Produktion } A \rightarrow x_i\}, & i = j \\ \{A \mid \text{es gibt eine Produktion } A \rightarrow BC \wedge \exists k : i \leq k < j \wedge B \in N_{i,k} \wedge C \in N_{k+1,j}\}, & i < j \end{cases}$$

Beispiel für $w = x_1 \cdots x_5$:

$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,3}$	$N_{2,3}$	$N_{3,3}$		
$N_{1,2}$	$N_{2,2}$			
$N_{1,1}$				

Beispiel: CYK-Verfahren

Rekursionsformel für $N_{i,j} := \{A \in N \mid A \xrightarrow{*} x_i \cdots x_j\}$:

$$N_{i,j} = \begin{cases} \{A \mid \text{es gibt eine Produktion } A \rightarrow x_i\}, & i = j \\ \{A \mid \text{es gibt eine Produktion } A \rightarrow BC \wedge \exists k : i \leq k < j \wedge B \in N_{i,k} \wedge C \in N_{k+1,j}\}, & i < j \end{cases}$$

Beispiel für $w = x_1 \cdots x_5$:

$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,3}$	$N_{2,3}$	$N_{3,3}$		
$N_{1,2}$	$N_{2,2}$			
$N_{1,1}$				

Beispiel: CYK-Verfahren

Rekursionsformel für $N_{i,j} := \{A \in N \mid A \xrightarrow{*} x_i \cdots x_j\}$:

$$N_{i,j} = \begin{cases} \{A \mid \text{es gibt eine Produktion } A \rightarrow x_i\}, & i = j \\ \{A \mid \text{es gibt eine Produktion } A \rightarrow BC \wedge \exists k : i \leq k < j \wedge B \in N_{i,k} \wedge C \in N_{k+1,j}\}, & i < j \end{cases}$$

Beispiel für $w = x_1 \cdots x_5$:

$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,3}$	$N_{2,3}$	$N_{3,3}$		
$N_{1,2}$	$N_{2,2}$			
$N_{1,1}$				

Beispiel: CYK-Verfahren

Rekursionsformel für $N_{i,j} := \{A \in N \mid A \xrightarrow{*} x_i \cdots x_j\}$:

$$N_{i,j} = \begin{cases} \{A \mid \text{es gibt eine Produktion } A \rightarrow x_i\}, & i = j \\ \{A \mid \text{es gibt eine Produktion } A \rightarrow BC \wedge \exists k : i \leq k < j \wedge B \in N_{i,k} \wedge C \in N_{k+1,j}\}, & i < j \end{cases}$$

Beispiel für $w = x_1 \cdots x_5$:

$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,3}$	$N_{2,3}$	$N_{3,3}$		
$N_{1,2}$	$N_{2,2}$			
$N_{1,1}$				

Beispiel: CYK-Verfahren

Rekursionsformel für $N_{i,j} := \{A \in N \mid A \xrightarrow{*} x_i \cdots x_j\}$:

$$N_{i,j} = \begin{cases} \{A \mid \text{es gibt eine Produktion } A \rightarrow x_i\}, & i = j \\ \{A \mid \text{es gibt eine Produktion } A \rightarrow BC \wedge \exists k : i \leq k < j \wedge B \in N_{i,k} \wedge C \in N_{k+1,j}\}, & i < j \end{cases}$$

Beispiel für $w = x_1 \cdots x_5$:

$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,3}$	$N_{2,3}$	$N_{3,3}$		
$N_{1,2}$	$N_{2,2}$			
$N_{1,1}$				

Beispiel: CYK-Verfahren

Rekursionsformel für $N_{i,j} := \{A \in N \mid A \xrightarrow{*} x_i \cdots x_j\}$:

$$N_{i,j} = \begin{cases} \{A \mid \text{es gibt eine Produktion } A \rightarrow x_i\}, & i = j \\ \{A \mid \text{es gibt eine Produktion } A \rightarrow BC \wedge \exists k : i \leq k < j \wedge B \in N_{i,k} \wedge C \in N_{k+1,j}\}, & i < j \end{cases}$$

Beispiel für $w = x_1 \cdots x_5$:

$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,3}$	$N_{2,3}$	$N_{3,3}$		
$N_{1,2}$	$N_{2,2}$			
$N_{1,1}$				

Beispiel: CYK-Verfahren

Rekursionsformel für $N_{i,j} := \{A \in N \mid A \xrightarrow{*} x_i \cdots x_j\}$:

$$N_{i,j} = \begin{cases} \{A \mid \text{es gibt eine Produktion } A \rightarrow x_i\}, & i = j \\ \{A \mid \text{es gibt eine Produktion } A \rightarrow BC \wedge \exists k : i \leq k < j \wedge B \in N_{i,k} \wedge C \in N_{k+1,j}\}, & i < j \end{cases}$$

Beispiel für $w = x_1 \cdots x_5$:

$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,3}$	$N_{2,3}$	$N_{3,3}$		
$N_{1,2}$	$N_{2,2}$			
$N_{1,1}$				

Beispiel: CYK-Verfahren

Rekursionsformel für $N_{i,j} := \{A \in N \mid A \xrightarrow{*} x_i \cdots x_j\}$:

$$N_{i,j} = \begin{cases} \{A \mid \text{es gibt eine Produktion } A \rightarrow x_i\}, & i = j \\ \{A \mid \text{es gibt eine Produktion } A \rightarrow BC \wedge \exists k : i \leq k < j \wedge B \in N_{i,k} \wedge C \in N_{k+1,j}\}, & i < j \end{cases}$$

Beispiel für $w = x_1 \cdots x_5$:

$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,3}$	$N_{2,3}$	$N_{3,3}$		
$N_{1,2}$	$N_{2,2}$			
$N_{1,1}$				

Beispiel: CYK-Verfahren

Rekursionsformel für $N_{i,j} := \{A \in N \mid A \xrightarrow{*} x_i \cdots x_j\}$:

$$N_{i,j} = \begin{cases} \{A \mid \text{es gibt eine Produktion } A \rightarrow x_i\}, & i = j \\ \{A \mid \text{es gibt eine Produktion } A \rightarrow BC \wedge \exists k : i \leq k < j \wedge B \in N_{i,k} \wedge C \in N_{k+1,j}\}, & i < j \end{cases}$$

Beispiel für $w = x_1 \cdots x_5$:

$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,3}$	$N_{2,3}$	$N_{3,3}$		
$N_{1,2}$	$N_{2,2}$			
$N_{1,1}$				

Beispiel: CYK-Verfahren

Rekursionsformel für $N_{i,j} := \{A \in N \mid A \xrightarrow{*} x_i \cdots x_j\}$:

$$N_{i,j} = \begin{cases} \{A \mid \text{es gibt eine Produktion } A \rightarrow x_i\}, & i = j \\ \{A \mid \text{es gibt eine Produktion } A \rightarrow BC \wedge \exists k : i \leq k < j \wedge B \in N_{i,k} \wedge C \in N_{k+1,j}\}, & i < j \end{cases}$$

Beispiel für $w = x_1 \cdots x_5$:

$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,3}$	$N_{2,3}$	$N_{3,3}$		
$N_{1,2}$	$N_{2,2}$			
$N_{1,1}$				

Beispiel: CYK-Verfahren

Rekursionsformel für $N_{i,j} := \{A \in N \mid A \xrightarrow{*} x_i \cdots x_j\}$:

$$N_{i,j} = \begin{cases} \{A \mid \text{es gibt eine Produktion } A \rightarrow x_i\}, & i = j \\ \{A \mid \text{es gibt eine Produktion } A \rightarrow BC \wedge \exists k : i \leq k < j \wedge B \in N_{i,k} \wedge C \in N_{k+1,j}\}, & i < j \end{cases}$$

Beispiel für $w = x_1 \cdots x_5$:

$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,3}$	$N_{2,3}$	$N_{3,3}$		
$N_{1,2}$	$N_{2,2}$			
$N_{1,1}$				

Beispiel: CYK-Verfahren

Rekursionsformel für $N_{i,j} := \{A \in N \mid A \xrightarrow{*} x_i \cdots x_j\}$:

$$N_{i,j} = \begin{cases} \{A \mid \text{es gibt eine Produktion } A \rightarrow x_i\}, & i = j \\ \{A \mid \text{es gibt eine Produktion } A \rightarrow BC \wedge \exists k : i \leq k < j \wedge B \in N_{i,k} \wedge C \in N_{k+1,j}\}, & i < j \end{cases}$$

Beispiel für $w = x_1 \cdots x_5$:

$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,3}$	$N_{2,3}$	$N_{3,3}$		
$N_{1,2}$	$N_{2,2}$			
$N_{1,1}$				

Beispiel: CYK-Verfahren

Rekursionsformel für $N_{i,j} := \{A \in N \mid A \xrightarrow{*} x_i \cdots x_j\}$:

$$N_{i,j} = \begin{cases} \{A \mid \text{es gibt eine Produktion } A \rightarrow x_i\}, & i = j \\ \{A \mid \text{es gibt eine Produktion } A \rightarrow BC \wedge \exists k : i \leq k < j \wedge B \in N_{i,k} \wedge C \in N_{k+1,j}\}, & i < j \end{cases}$$

Beispiel für $w = x_1 \cdots x_5$:

$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,3}$	$N_{2,3}$	$N_{3,3}$		
$N_{1,2}$	$N_{2,2}$			
$N_{1,1}$				

Beispiel: CYK-Verfahren

Rekursionsformel für $N_{i,j} := \{A \in N \mid A \xrightarrow{*} x_i \cdots x_j\}$:

$$N_{i,j} = \begin{cases} \{A \mid \text{es gibt eine Produktion } A \rightarrow x_i\}, & i = j \\ \{A \mid \text{es gibt eine Produktion } A \rightarrow BC \wedge \exists k : i \leq k < j \wedge B \in N_{i,k} \wedge C \in N_{k+1,j}\}, & i < j \end{cases}$$

Beispiel für $w = x_1 \cdots x_5$:

$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,3}$	$N_{2,3}$	$N_{3,3}$		
$N_{1,2}$	$N_{2,2}$			
$N_{1,1}$				

Beispiel: CYK-Verfahren

Rekursionsformel für $N_{i,j} := \{A \in N \mid A \xrightarrow{*} x_i \cdots x_j\}$:

$$N_{i,j} = \begin{cases} \{A \mid \text{es gibt eine Produktion } A \rightarrow x_i\}, & i = j \\ \{A \mid \text{es gibt eine Produktion } A \rightarrow BC \wedge \exists k : i \leq k < j \wedge B \in N_{i,k} \wedge C \in N_{k+1,j}\}, & i < j \end{cases}$$

Beispiel für $w = x_1 \cdots x_5$:

$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,3}$	$N_{2,3}$	$N_{3,3}$		
$N_{1,2}$	$N_{2,2}$			
$N_{1,1}$				

Beispiel: CYK-Verfahren

Rekursionsformel für $N_{i,j} := \{A \in N \mid A \xrightarrow{*} x_i \cdots x_j\}$:

$$N_{i,j} = \begin{cases} \{A \mid \text{es gibt eine Produktion } A \rightarrow x_i\}, & i = j \\ \{A \mid \text{es gibt eine Produktion } A \rightarrow BC \wedge \exists k : i \leq k < j \wedge B \in N_{i,k} \wedge C \in N_{k+1,j}\}, & i < j \end{cases}$$

Beispiel für $w = x_1 \cdots x_5$:

$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,3}$	$N_{2,3}$	$N_{3,3}$		
$N_{1,2}$	$N_{2,2}$			
$N_{1,1}$				

Beispiel: CYK-Verfahren

Rekursionsformel für $N_{i,j} := \{A \in N \mid A \xrightarrow{*} x_i \cdots x_j\}$:

$$N_{i,j} = \begin{cases} \{A \mid \text{es gibt eine Produktion } A \rightarrow x_i\}, & i = j \\ \{A \mid \text{es gibt eine Produktion } A \rightarrow BC \wedge \exists k : i \leq k < j \wedge B \in N_{i,k} \wedge C \in N_{k+1,j}\}, & i < j \end{cases}$$

Beispiel für $w = x_1 \cdots x_5$:

$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,3}$	$N_{2,3}$	$N_{3,3}$		
$N_{1,2}$	$N_{2,2}$			
$N_{1,1}$				

Beispiel: CYK-Verfahren

Rekursionsformel für $N_{i,j} := \{A \in N \mid A \xrightarrow{*} x_i \cdots x_j\}$:

$$N_{i,j} = \begin{cases} \{A \mid \text{es gibt eine Produktion } A \rightarrow x_i\}, & i = j \\ \{A \mid \text{es gibt eine Produktion } A \rightarrow BC \wedge \exists k : i \leq k < j \wedge B \in N_{i,k} \wedge C \in N_{k+1,j}\}, & i < j \end{cases}$$

Beispiel für $w = x_1 \cdots x_5$:

$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,3}$	$N_{2,3}$	$N_{3,3}$		
$N_{1,2}$	$N_{2,2}$			
$N_{1,1}$				

A diagram illustrating the CYK algorithm. It shows a 5x5 grid of cells labeled $N_{i,j}$. A blue arrow points from the bottom-left cell ($N_{1,1}$) up to the cell ($N_{1,4}$). Another blue arrow points horizontally from the cell ($N_{1,4}$) across the row to the cell ($N_{5,5}$). The cell ($N_{1,5}$) is circled in red.

Beispiel: CYK-Verfahren

Rekursionsformel für $N_{i,j} := \{A \in N \mid A \xrightarrow{*} x_i \cdots x_j\}$:

$$N_{i,j} = \begin{cases} \{A \mid \text{es gibt eine Produktion } A \rightarrow x_i\}, & i = j \\ \{A \mid \text{es gibt eine Produktion } A \rightarrow BC \wedge \exists k : i \leq k < j \wedge B \in N_{i,k} \wedge C \in N_{k+1,j}\}, & i < j \end{cases}$$

Beispiel für $w = x_1 \cdots x_5$:

$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,3}$	$N_{2,3}$	$N_{3,3}$		
$N_{1,2}$	$N_{2,2}$			
$N_{1,1}$				

Beispiel: CYK-Verfahren

Rekursionsformel für $N_{i,j} := \{A \in N \mid A \xrightarrow{*} x_i \cdots x_j\}$:

$$N_{i,j} = \begin{cases} \{A \mid \text{es gibt eine Produktion } A \rightarrow x_i\}, & i = j \\ \{A \mid \text{es gibt eine Produktion } A \rightarrow BC \wedge \exists k : i \leq k < j \wedge B \in N_{i,k} \wedge C \in N_{k+1,j}\}, & i < j \end{cases}$$

Beispiel für $w = x_1 \cdots x_5$:

$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,3}$	$N_{2,3}$	$N_{3,3}$		
$N_{1,2}$	$N_{2,2}$			
$N_{1,1}$				

Beispiel: CYK-Verfahren

Rekursionsformel für $N_{i,j} := \{A \in N \mid A \xrightarrow{*} x_i \cdots x_j\}$:

$$N_{i,j} = \begin{cases} \{A \mid \text{es gibt eine Produktion } A \rightarrow x_i\}, & i = j \\ \{A \mid \text{es gibt eine Produktion } A \rightarrow BC \wedge \exists k : i \leq k < j \wedge B \in N_{i,k} \wedge C \in N_{k+1,j}\}, & i < j \end{cases}$$

Beispiel für $w = x_1 \cdots x_5$:

$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,3}$	$N_{2,3}$	$N_{2,3}$		
$N_{1,2}$	$N_{2,2}$			
$N_{1,1}$				

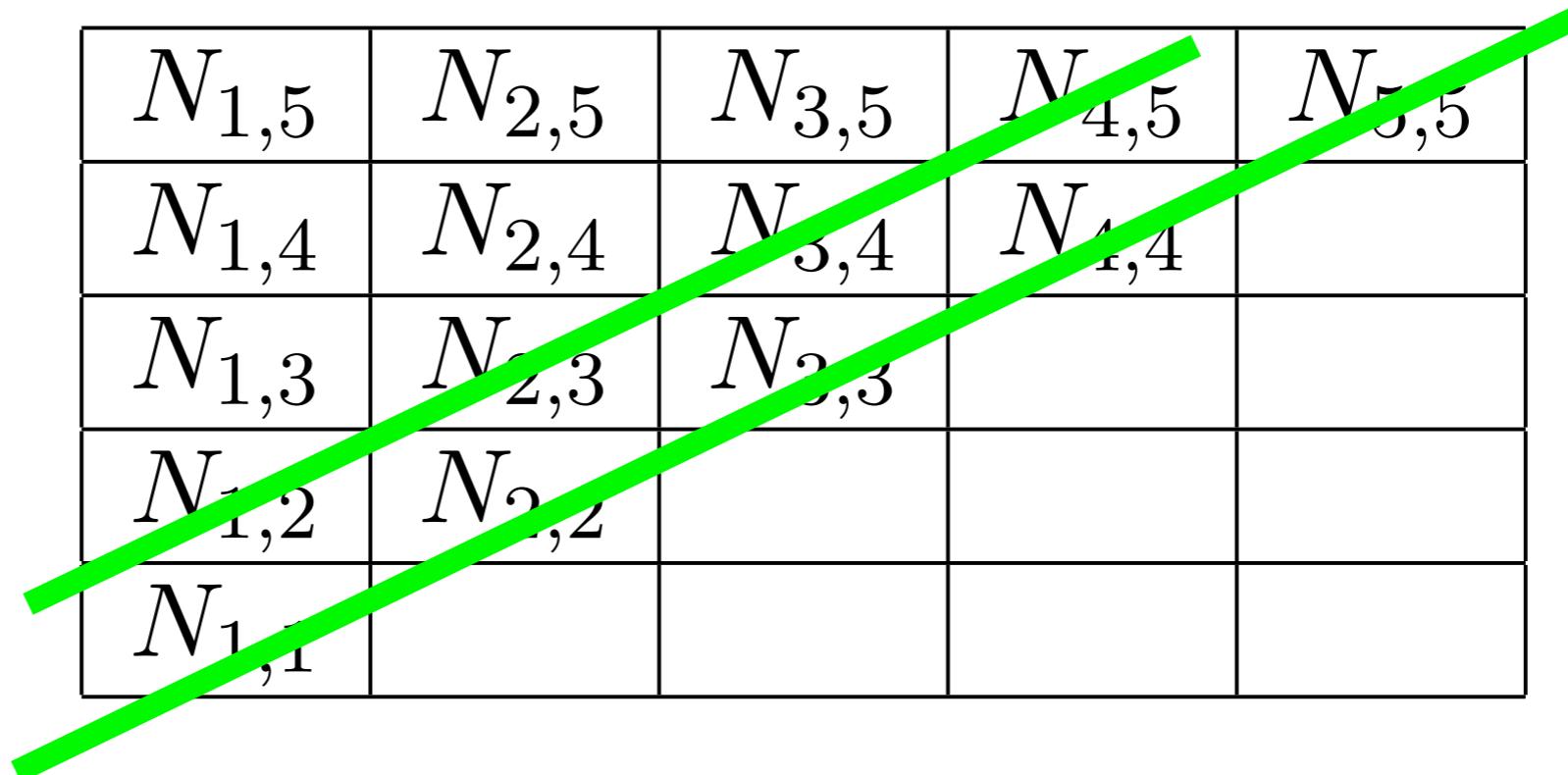
Beispiel: CYK-Verfahren

Rekursionsformel für $N_{i,j} := \{A \in N \mid A \xrightarrow{*} x_i \cdots x_j\}$:

$$N_{i,j} = \begin{cases} \{A \mid \text{es gibt eine Produktion } A \rightarrow x_i\}, & i = j \\ \{A \mid \text{es gibt eine Produktion } A \rightarrow BC \wedge \exists k : i \leq k < j \wedge B \in N_{i,k} \wedge C \in N_{k+1,j}\}, & i < j \end{cases}$$

Beispiel für $w = x_1 \cdots x_5$:

$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,3}$	$N_{2,3}$	$N_{2,3}$		
$N_{1,2}$	$N_{2,2}$			
$N_{1,1}$				



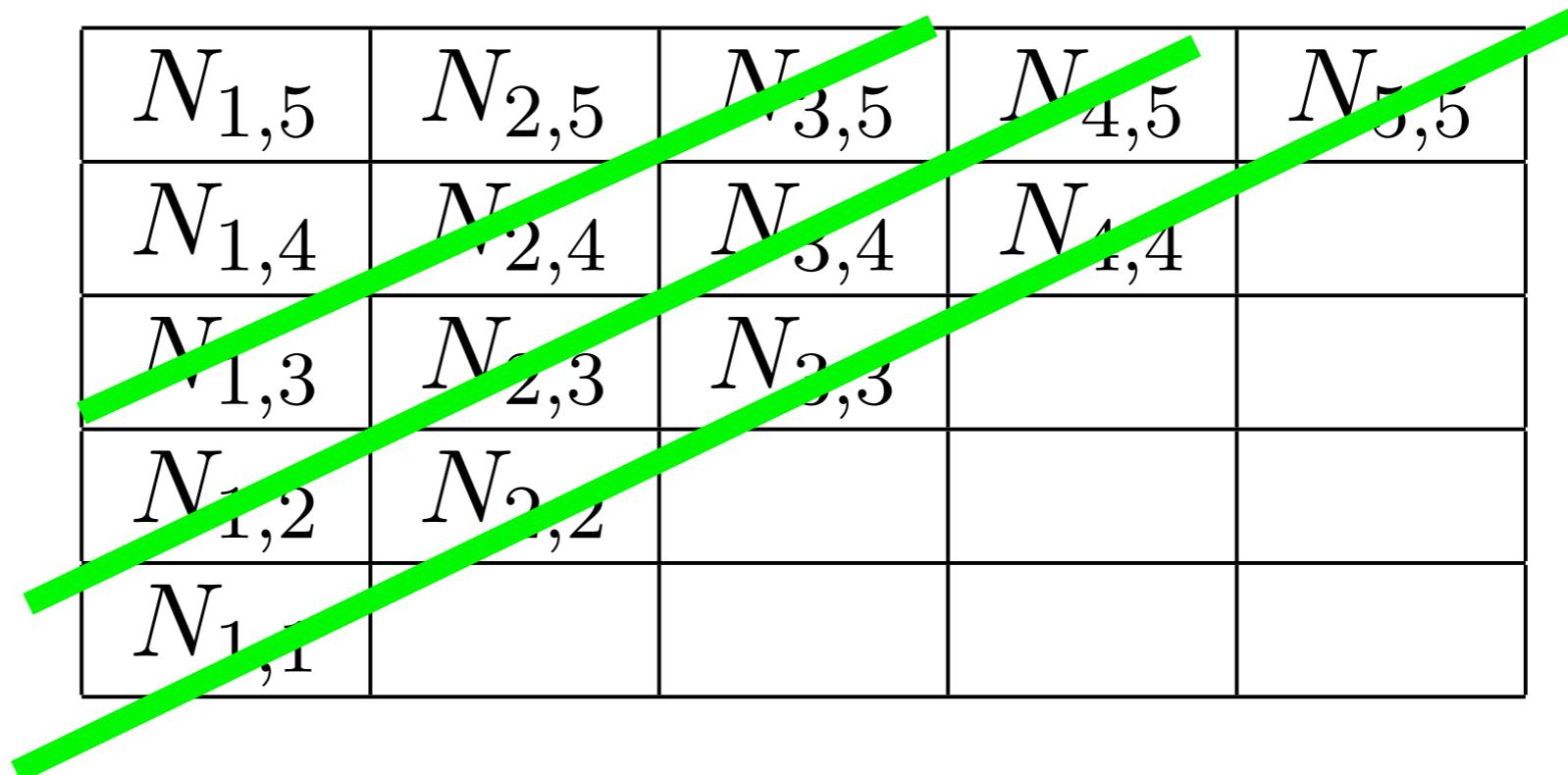
Beispiel: CYK-Verfahren

Rekursionsformel für $N_{i,j} := \{A \in N \mid A \xrightarrow{*} x_i \cdots x_j\}$:

$$N_{i,j} = \begin{cases} \{A \mid \text{es gibt eine Produktion } A \rightarrow x_i\}, & i = j \\ \{A \mid \text{es gibt eine Produktion } A \rightarrow BC \wedge \exists k : i \leq k < j \wedge B \in N_{i,k} \wedge C \in N_{k+1,j}\}, & i < j \end{cases}$$

Beispiel für $w = x_1 \cdots x_5$:

$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,3}$	$N_{2,3}$	$N_{3,3}$		
$N_{1,2}$	$N_{2,2}$			
$N_{1,1}$				



Beispiel: CYK-Verfahren

Rekursionsformel für $N_{i,j} := \{A \in N \mid A \xrightarrow{*} x_i \cdots x_j\}$:

$$N_{i,j} = \begin{cases} \{A \mid \text{es gibt eine Produktion } A \rightarrow x_i\}, & i = j \\ \{A \mid \text{es gibt eine Produktion } A \rightarrow BC \wedge \exists k : i \leq k < j \wedge B \in N_{i,k} \wedge C \in N_{k+1,j}\}, & i < j \end{cases}$$

Beispiel für $w = x_1 \cdots x_5$:

$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,3}$	$N_{2,3}$	$N_{3,3}$		
$N_{1,2}$	$N_{2,2}$			
$N_{1,1}$				

Komplexität: CYK-Verfahren

Beispiel für $w = x_1 \cdots x_5$:

$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,3}$	$N_{2,3}$	$N_{3,3}$		
$N_{1,2}$	$N_{2,2}$			
$N_{1,1}$				

Komplexitätsabschätzung:

- Wir müssen also $\simeq n^2/2$ viele Tabelleneinträge berechnen.
- Jede Berechnung iteriert über alle k , wobei $1 \leq k \leq n$ und alle Produktionen $A \rightarrow BC$.
- Insgesamt also: $n^2/2 \cdot k \cdot |P|$ viele Tests, wobei $|P|$ für eine feste Grammatik eine Konstante ist.
- Also Laufzeit = $const \cdot n^3$

Komplexität: CYK-Verfahren

Beispiel für $w = x_1 \cdots x_5$:

$N_{1,5}$	$N_{2,5}$	$N_{3,5}$	$N_{4,5}$	$N_{5,5}$
$N_{1,4}$	$N_{2,4}$	$N_{3,4}$	$N_{4,4}$	
$N_{1,3}$	$N_{2,3}$	$N_{3,3}$		
$N_{1,2}$	$N_{2,2}$			
$N_{1,1}$				

Komplexitätsabschätzung:

- Wir müssen also $\simeq n^2/2$ viele Tabelleneinträge berechnen.
- Jede Berechnung iteriert über alle k , wobei $1 \leq k \leq n$ und alle Produktionen $A \rightarrow BC$.
- Insgesamt also: $n^2/2 \cdot k \cdot |P|$ viele Tests, wobei $|P|$ für eine feste Grammatik eine Konstante ist.
- Also Laufzeit = $const \cdot n^3$

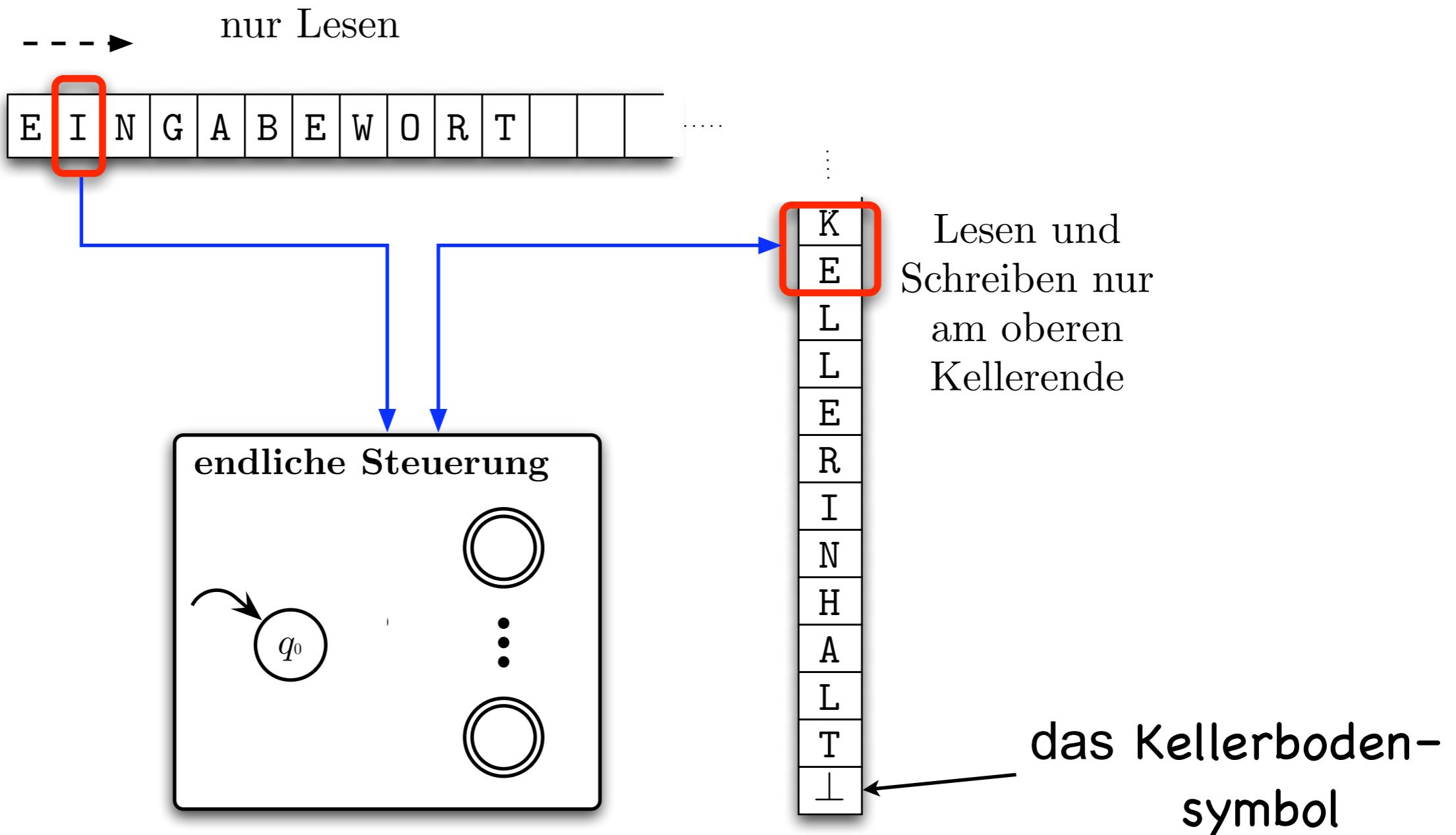
FGI 1

Automaten, Formale Sprachen,
Berechenbarkeit

Kap. 16 Kellerautomat

Michael Köhler-Bußmeier

Kellerautomat (informal)



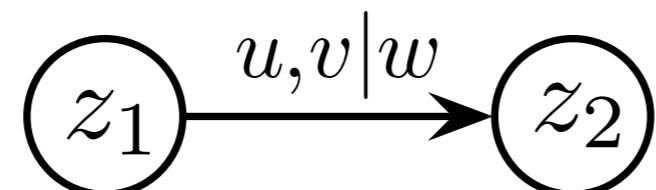
Kellerautomat (formal)

Definition 16.1:

- Ein **nichtdeterministischer Kellerautomat** (*PDA* für *push down automaton*) ist ein Tupel $A := (Q, \Sigma, \Gamma, \delta, q_0, \perp, F)$, wobei gilt:
 - Q ist endliche Menge von **Zuständen**.
 - Σ ist endliches **Eingabealphabet**.
 - Γ ist endliches **Kelleralphabet**.
 - $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow 2^{Q \times \Gamma^*}$ ist die endliche **Zustandsüberführungsfunktion**.
 - $q_0 \in Q$ ist der **Startzustand**.
 - $\perp \in \Gamma$ ist das **Kellerbodenzeichen** oder **Kellerstartsymbol**.
 - $F \subseteq Q$ ist die Menge der **Endzustände**.

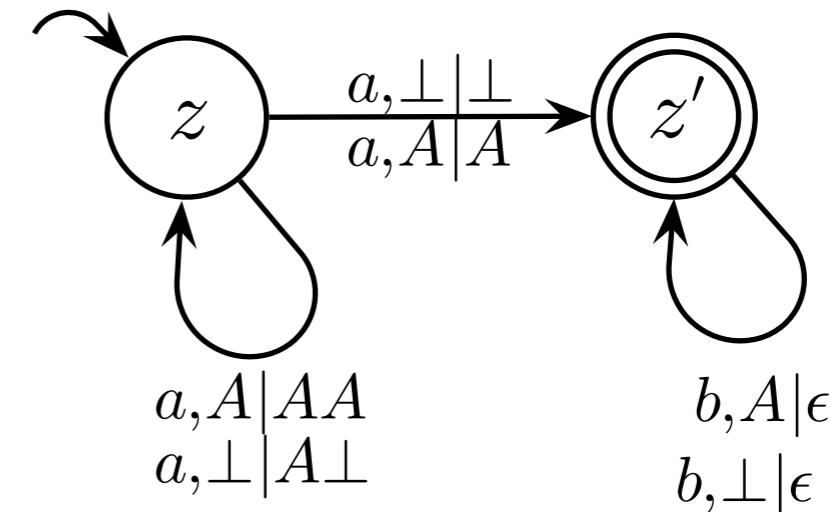
Konfigurationen

- **Wichtige Notation:** Der Kellerinhalt wird durch ein Wort $v \in \Gamma^*$ so beschrieben, dass das oberste Zeichen des Kelleinhaltes in v ganz am Anfang, d.h. links, steht.
- **Beispiel:** $abab\perp$ bedeutet, dass a das oberste und \perp das unterste Zeichen im Keller ist.
- **Konfigurationen** sind Elemente aus $Q \times \Sigma^* \times \Gamma^*$.
- Zunächst eine weitere **Notation:** Zustandsüberführungen werden als beschriftete Kanten im Zustands(übergangs)diagramm gezeichnet. Für $(z_2, w) \in \delta(z_1, u, v)$:



Übergangsrelation zwischen Konfigurationen

Beispiel PDA:



- Offensichtliche Frage:

- Was ist die akzeptierte Sprache eines *PDA*?
- ... dazu benötigen wir folgende Begriffe:
 - * Rechnung/Erfolgsrechnung eines *PDA*
 - * Konfiguration eines *PDA*
 - * Überführungs- oder Transitionsrelation eines *PDA*

Konfiguration eines PDA

Ähnlich wie bei endlichen Automaten erklären wir eine Relation für Konfigurationen und deren Übergänge.

Definition 16.2:

Für einen *PDA* K heißt $w \in Q \times \Sigma^* \times \Gamma^*$ **Konfiguration** gdw.

- $w = (p, u, v)$ mit $u \in \Sigma^*, v \in \Gamma^*$ und $p \in Q$:
- K befindet sich im Zustand p , die noch zu verarbeitende Eingabe ist u und v ist der Kellerinhalt.
- $v = \epsilon$ bedeutet, dass gar nichts, nicht einmal das Kellerbodenzeichen, im Keller steht!
- Falls $v \neq \epsilon$, dann ist $v \in \Gamma \cdot \Gamma^*$, d.h., ganz links in v steht das oberste Kellersymbol.
- Entsprechend bedeutet $u = \epsilon$, dass die Eingabe vollständig gelesen wurde.

Die Überführungsrelation beim PDA

Definition 16.3:

Seien $u, u' \in \Gamma^*$, $w, w' \in \Sigma^*$ und $z, z' \in Q$.

Zwischen Konfigurationen eines Kellerautomaten A wird die **Überführungsrelation** (Rechenschritt-, Transitionsrelation)

$$\vdash \subseteq (Q \times \Sigma^* \times \Gamma^*) \times (Q \times \Sigma^* \times \Gamma^*)$$

definiert durch:

$$(z, xu, yv') \vdash (z', u, y'v') \text{ gdw. } (z', y') \in \delta(z, x, y)$$

Wie üblich bezeichnet \vdash^* die reflexive, transitive Hülle von \vdash und wir schreiben \vdash_A , wenn andernfalls unklar ist, zu welchem *PDA* diese Überführungsrelation gehört.

Akzeptierten von Wörtern mit PDA

- Verwendet werden i.A. Bedingungen an die Konfiguration eines Automaten.
- **Zur Erinnerung:** Bei **endlichen Automaten** musste ein *Endzustand* erreicht werden.
- Beim **Kellerautomaten** haben wir zwei Möglichkeiten:
 - Zustand der Steuerungskomponente
 - Speicherzustand
- Dies führt zur **Akzeptierung mit Endzustand** und zur **Akzeptierung mit leerem Keller**.

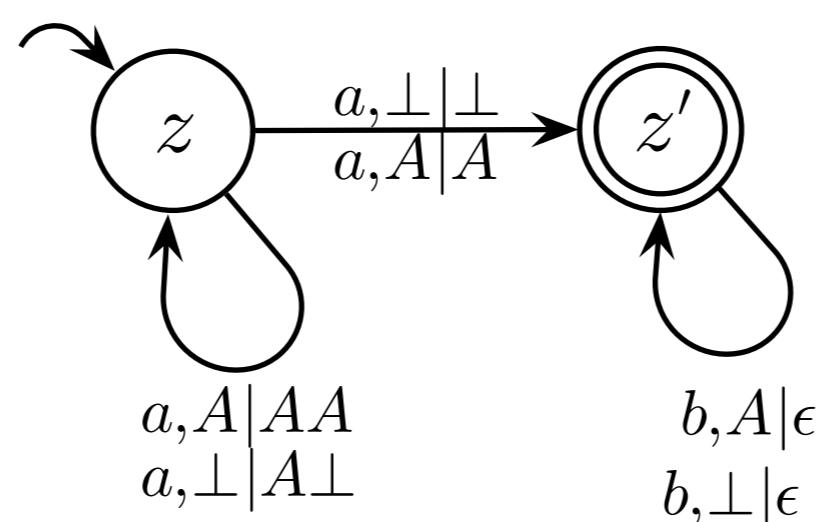
mit Endzustand akzeptierte Sprache

Definition 16.4:

Die vom dem *PDA* $A := (Q, \Sigma, \Gamma, \delta, q_0, \perp, F)$ mit Endzustand akzeptierte Sprache $L(A)$ (Bei Vossen/Witt mit $L_F(A)$ bezeichnet) ist

$$L(A) := \left\{ w \in \Sigma^* \mid \exists p \in F : \exists \gamma \in \Gamma^* : (q_0, w, \perp) \vdash^* (p, \epsilon, \gamma) \right\}$$

Beispiel: $L(A) = \{a^n b^m \mid n, m \in \mathbb{N} \wedge n \geq 1 \wedge n \geq m\}$



... für jedes gelesene b muss vorher ein a gelesen worden sein!

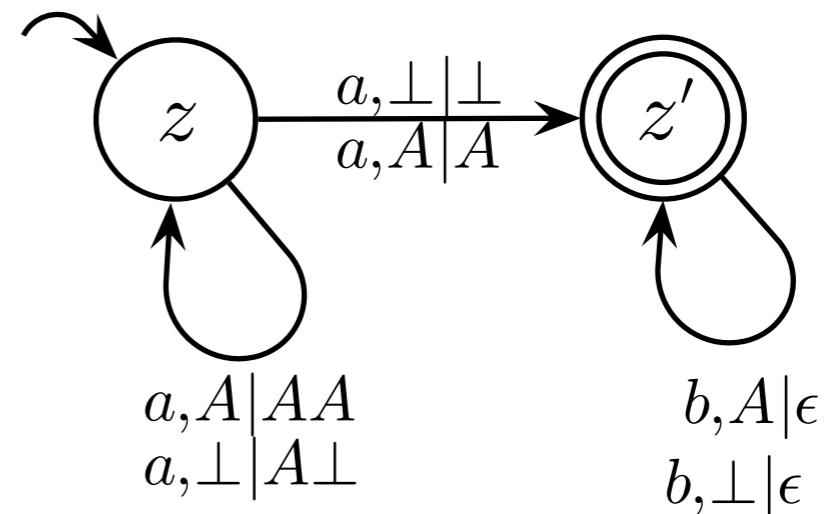
Beispielrechnung

Start $(z, aaabb, \perp)$

$\vdash (z, aabb, A\perp)$

$\vdash (z, abb, AA\perp)$

$\vdash (z, bb, AAA\perp)$



- und hier blockiert der PDA.
- Eine weitere Rechnung für $aaabb$:
 $(z, aaabb, \perp) \vdash (z, aabb, A\perp) \vdash (z, abb, AA\perp) \vdash (z', bb, AAA\perp) \vdash (z', b, A\perp) \vdash (z', \epsilon, \perp)$

Zwar ist der Keller *nicht* leer (er enthält noch \perp), aber es ist ein Endzustand erreicht! Folglich akzeptieren!

- Für $aabbb$, jedoch, existiert *keine* Erfolgsrechnung.

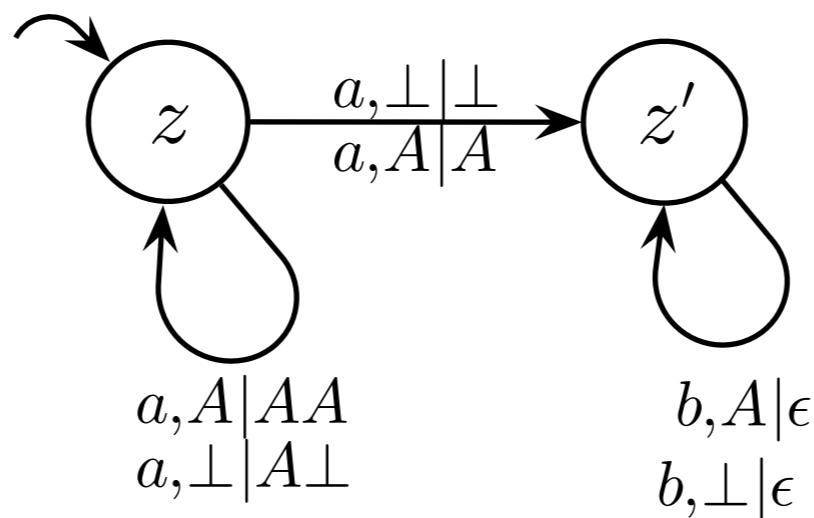
mit leerem Keller akzeptierte Sprache

Definition 16.5:

Die vom nichtdeterministischen *PDA* A mit leerem Keller akzeptierte Sprache $L_\epsilon(A)$ ist:

$$L_\epsilon(A) := \left\{ w \in \Sigma^* \mid \exists p \in Q : (q_0, w, \perp) \vdash^* (p, \epsilon, \epsilon) \right\}$$

Beispiel: $L_\epsilon(A) = \{a^n b^n \mid n \in \mathbb{N}, n \geq 1\} \dots$ für jedes gelesene



b muss vorher ein a gelesen worden sein und für jedes a muss später ein b folgen!

Äquivalenz (wie üblich)

Definition 16.6:

Zwei Kellerautomaten A und B heißen **äquivalent** genau dann, wenn ihre mit Endzustand akzeptierten Sprachen gleich sind, d.h. $L(A) = L(B)$ gilt.

- **Merke:** Dieser Äquivalenzbegriff vergleicht nur *PDAs*, die im Endzustand akzeptieren!
- Eine Verallgemeinerung wäre aber natürlich möglich:
 - bezüglich der mit leerem Keller akzeptierten Sprachen ($L_\epsilon(A) = L_\epsilon(B)$)
 - „interdisziplinär“ ($L(A) = L_\epsilon(B)$)

Akzeptieren mit Endzustand oder mit leerem Keller?

Satz 16.4:

Es existiert zu jedem PDA $A := (Q, \Sigma, \Gamma, \delta, q_0, \perp, F)$ ein PDA $B := (Q', \Sigma, \Gamma', \delta', q_{\text{start}}, \#, F')$ mit $L(A) = L_\epsilon(B) = L(B)$.

Beweis:

Konstruiere PDA $B := (Q', \Sigma, \Gamma', \delta', q_{\text{start}}, \#, \{q_{\text{accept}}\})$ mit

- neuem Kellersymbol $\#$, also $\Gamma' := \Gamma \uplus \{\#\}$;
- neuen Zuständen $Q' := Q \uplus \{q_{\text{start}}, q_{\text{empty}}, q_{\text{accept}}\}$
- $(\perp \#, q_0) \in \delta'(q_{\text{start}}, \epsilon, \#)$
„Initialisierung“: Daruntersetzen des neuen Kellerboden $\#$.
- $(\epsilon, q_{\text{empty}}) \in \delta'(q_e, \epsilon, \epsilon)$ für alle $q_e \in F$ „Übergänge zu dem, den Keller leerenden, Zustand q_{empty} !“
- $(\epsilon, q_{\text{empty}}) \in \delta'(q_{\text{empty}}, \epsilon, a)$ für alle $a \in \Gamma$ „Keller leeren bis auf $\#$.“
- $(\epsilon, q_{\text{accept}}) \in \delta'(q_{\text{empty}}, \epsilon, \#)$ „Endzustand und leerer Keller!“

Akzeptieren mit Endzustand oder mit leerem Keller?

Satz 16.1 Zu jedem PDA $A = (Q, \Sigma, \Gamma, \delta, q_0, \perp, F)$ existiert ein PDA $B = (Q', \Sigma, \Gamma', \delta', p_0, \sharp, F')$ mit $L(B) = L_\epsilon(A)$.

Beweis: PDA $B = (Q \uplus \{p_0, p_f\}, \Sigma, \Gamma \uplus \{\sharp\}, \delta', p_0, \sharp, \{p_f\})$.

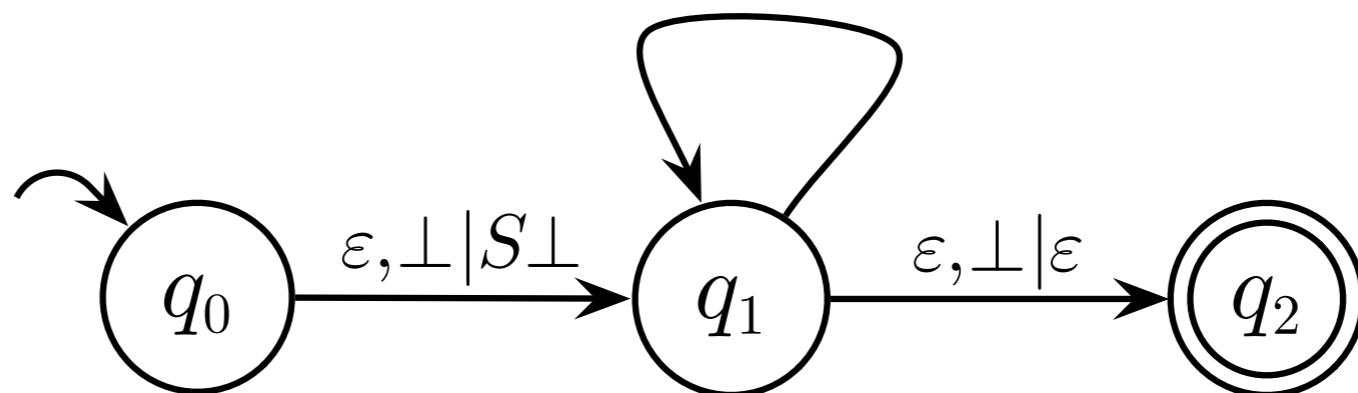
- Ergänze neuen Startzustand p_0 und neuen Endzustand p_f .
- Definiere neues Kellerbodensymbol \sharp
- Setze initial \perp oben auf den Keller: $\delta'(p_0, \epsilon, \sharp) = (q_0, \perp \sharp)$
- Simuliere A wie zuvor.
- Leert A den Keller, so erreicht B den Keller \sharp .
- Jeder Zustand q führt bei „leerem“ Keller zu p_f , d.h. $\delta'(q, \epsilon, \sharp) = (p_f, \epsilon)$ für alle $q \in Q$.

von der CFG zum PDA

Satz 16.5:

Zu jeder kontextfreien Grammatik $G = (\Sigma, N, P, S)$ kann effektiv ein Kellerautomat $A_G := (Q, \Sigma, \Gamma, \delta, q_0, \perp, F)$ konstruiert werden, für den $L_\epsilon(A_G) = L(A_G) = L(G)$ gilt.

- **Beweis:** Sei $G = (\Sigma, N, P, S)$ eine *CFG*.
- Konstruiere $A_G := (Q, \Sigma, \Gamma, \delta, q_0, \perp, F)$ mit drei Zuständen: $Q := \{q_0, q_1, q_2\}$
- $\Gamma := N \cup \Sigma \cup \{\perp\}$
- und Kanten:
 - $a, a | \epsilon$ für jedes $a \in \Sigma$
 - $\epsilon, A | w$ für alle $A \rightarrow w \in P$



vom PDA zur CFG

Satz 16.6 Zu jedem PDA A ist $L_\epsilon(A)$ eine kontextfreie Sprache.

Beweis: Zu einem gegebenen PDA A , der mit leerem Keller akzeptiert, konstruieren wir mittels der Tripelkonstruktion eine Grammatik G , die die Berechnungen von A simuliert.

- Grundidee: Wird ein Symbol X auf den Keller gepackt, dann betrachten wir solche Zeichenketten w , die von der Eingabe gelesen werden, bevor X wieder vom Keller genommen wird.

$$(q, w, X) \vdash^* (p, \epsilon, \epsilon)$$

- Wir definieren Nonterminale $[qXp]$, aus denen man alle Worte w ableiten kann, die der PDA A in q gestartet lesen kann, bis X vom Keller genommen wird:

$$[qXp] \xrightarrow{*} w \quad \text{gdw.} \quad (q, w, X) \vdash^* (p, \epsilon, \epsilon)$$

vom PDA zur CFG

Rekursion: Sei $(p_1, X_1 \cdots X_k) \in \delta(q, y, X)$ im PDA.

Angenommen wir wissen bereits für alle $i \in \{1, \dots, k\}$:

$$(p_i, w_i, X_i) \vdash^* (p_{i+1}, \epsilon, \epsilon),$$

dann können wir dies weiter zusammensetzen:

$$(q, yw_1 \cdots w_k, X) \vdash^* (p_k, \epsilon, \epsilon)$$

Dies folgt aus der folgenden Rechnung:

$$\begin{aligned} (q, yw_1 \cdots w_k, X) &\vdash (p_1, w_1 \cdots w_k, X_1 \cdots X_k) \\ &\vdash^* (p_2, w_2 \cdots w_k, X_2 \cdots X_k) \\ &\vdash^* \dots \\ &\vdash^* (p_k, w_k, X_k) \\ &\vdash^* (p_{k+1}, \epsilon, \epsilon) \end{aligned}$$

vom PDA zur CFG

Sei $(p_1, X_1 \cdots X_k) \in \delta(q, y, X)$ im PDA.

Die obige Rechnung

$$(q, yw_1 \cdots w_k, X) \vdash (p_1, w_1 \cdots w_k, X_1 \cdots X_k) \vdash^* (p_{k+1}, \epsilon, \epsilon)$$

wird in der Grammatik durch folgende Produktion simuliert:

$$[qXp_{k+1}] \rightarrow y[p_1X_1p_2] \cdots [p_kX_kp_{k+1}]$$

für alle möglichen $p_1, \dots, p_{k+1} \in Q$.

Insgesamt gilt dann die Simulation:

$$S \xrightarrow{*} w \quad \text{gdw.} \quad (q_0, w, \perp) \vdash^* (p, \epsilon, \epsilon) \quad \text{gdw.} \quad w \in L_\epsilon(A)$$

Charakterisierung kontextfreier Sprachen

Insgesamt haben wir also gezeigt:

Satz 16.2 *Folgende Aussagen sind äquivalent:*

- L ist kontextfrei.
- $L = L(A)$ für einen PDA A .
- $L = L_\epsilon(B)$ für einen PDA B .
- $L = L(G)$ für eine kontextfreie Grammatik G .

FGI 1

Automaten, Formale Sprachen,
Berechenbarkeit

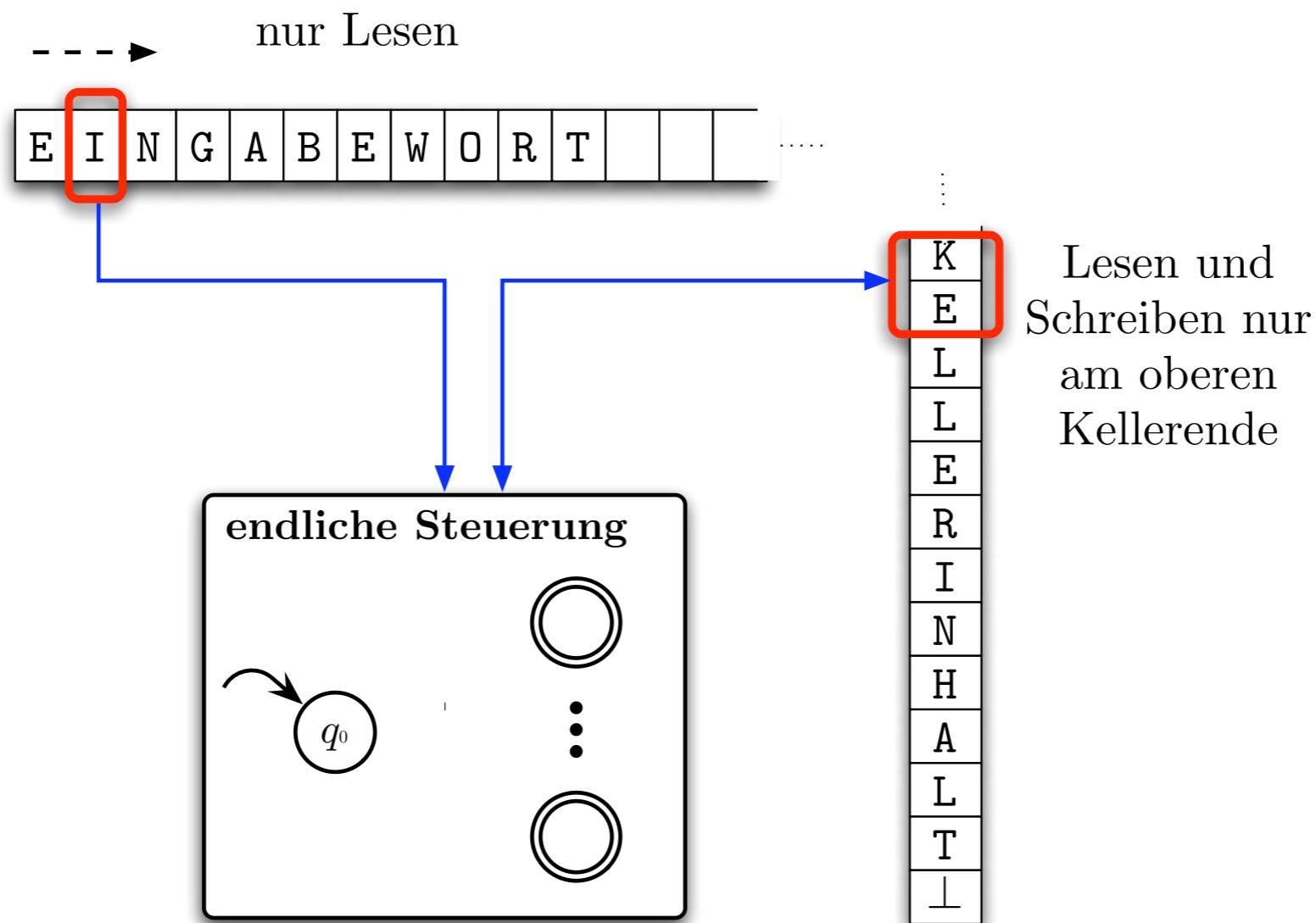
Kap. 17

Deterministischer Kellerautomat,
andere Chomsky Grammatiken

Michael Köhler-Bußmeier

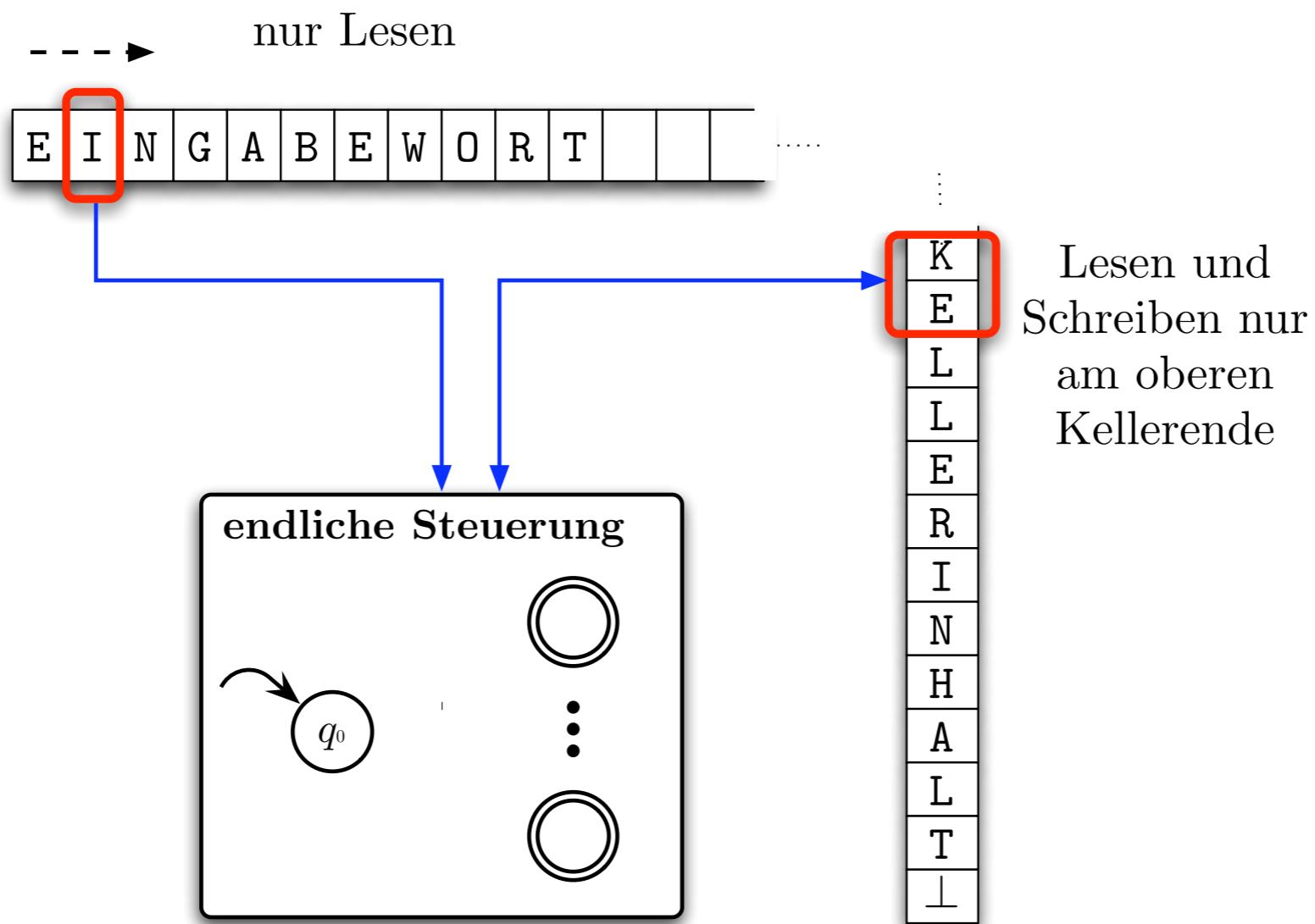
Kellerautomat (informal)

Zur Wiederholung:

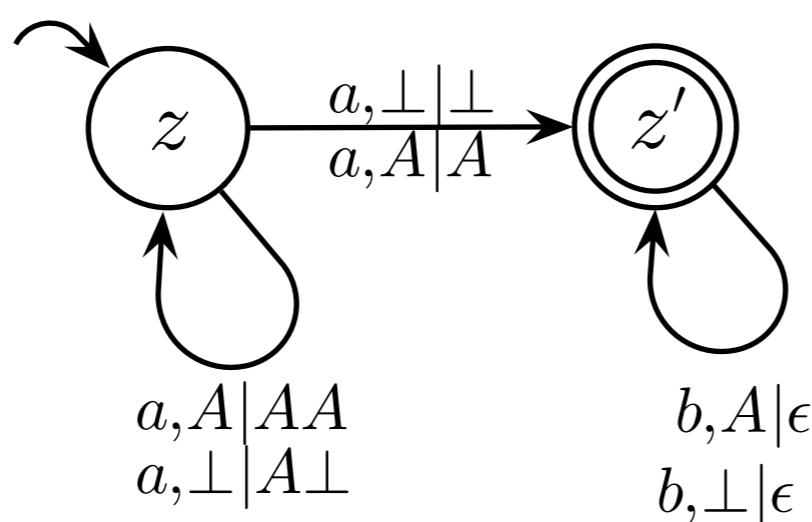


Kellerautomat (informal)

Zur Wiederholung:



Beispiel PDA:



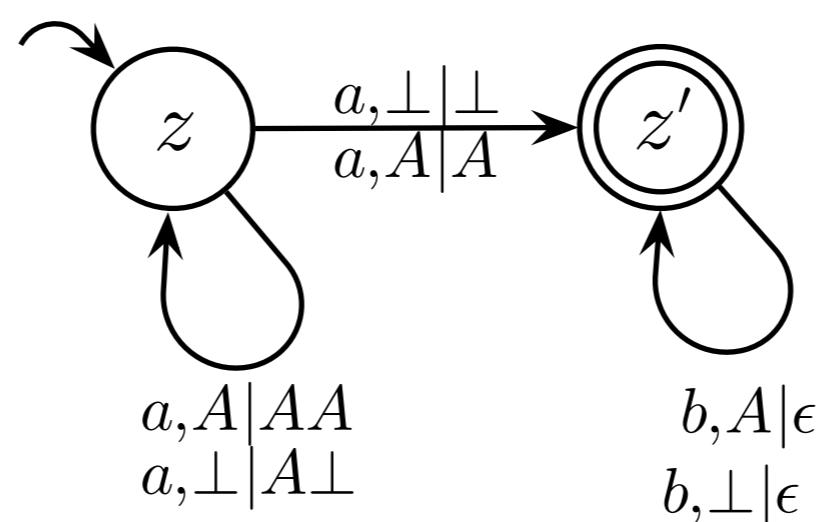
mit Endzustand akzeptierte Sprache

Definition 16.4:

Die vom dem *PDA* $A := (Q, \Sigma, \Gamma, \delta, q_0, \perp, F)$ mit Endzustand akzeptierte Sprache $L(A)$ (Bei Vossen/Witt mit $L_F(A)$ bezeichnet) ist

$$L(A) := \left\{ w \in \Sigma^* \mid \exists p \in F : \exists \gamma \in \Gamma^* : (q_0, w, \perp) \vdash^* (p, \epsilon, \gamma) \right\}$$

Beispiel: $L(A) = \{a^n b^m \mid n, m \in \mathbb{N} \wedge n \geq 1 \wedge n \geq m\}$



... für jedes gelesene b muss vorher ein a gelesen worden sein!

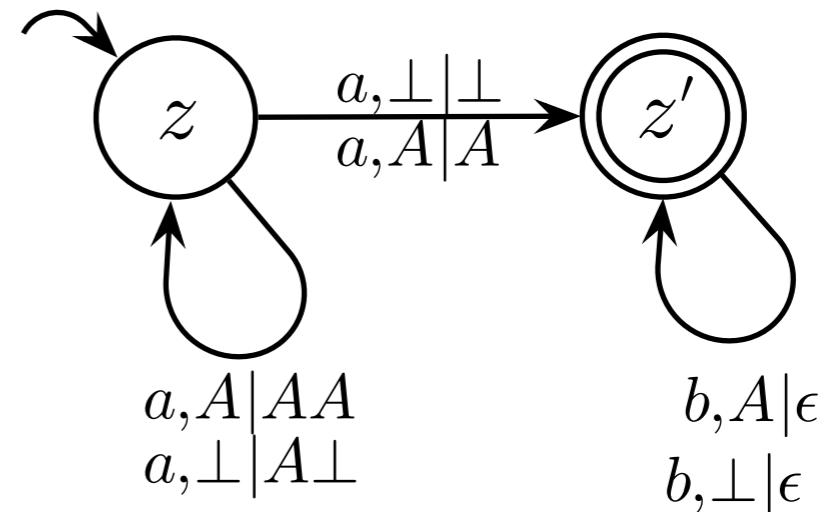
Beispielrechnung

Start $(z, aaabb, \perp)$

$\vdash (z, aabb, A\perp)$

$\vdash (z, abb, AA\perp)$

$\vdash (z, bb, AAA\perp)$



- und hier blockiert der PDA.
- Eine weitere Rechnung für $aaabb$:
 $(z, aaabb, \perp) \vdash (z, aabb, A\perp) \vdash (z, abb, AA\perp) \vdash (z', bb, AAA\perp) \vdash (z', b, A\perp) \vdash (z', \epsilon, \perp)$

Zwar ist der Keller *nicht* leer (er enthält noch \perp), aber es ist ein Endzustand erreicht! Folglich akzeptieren!

- Für $aabbb$, jedoch, existiert *keine* Erfolgsrechnung.

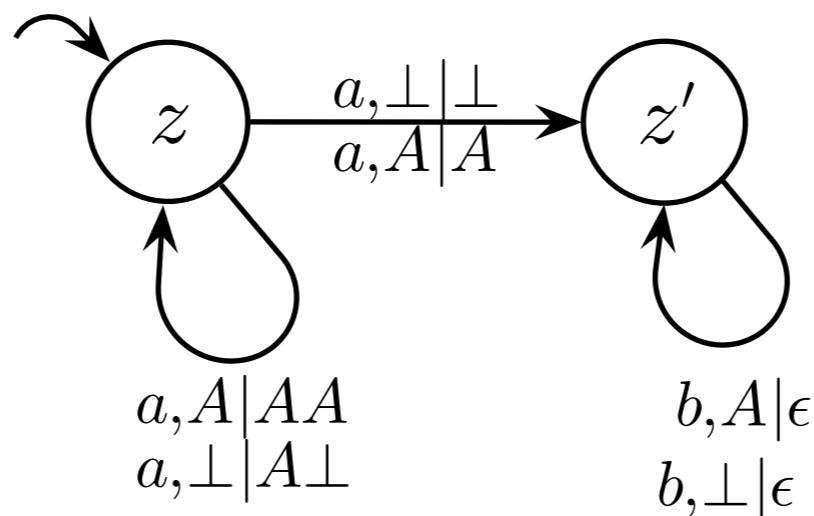
mit leerem Keller akzeptierte Sprache

Definition 16.5:

Die vom nichtdeterministischen *PDA* A mit leerem Keller akzeptierte Sprache $L_\epsilon(A)$ ist:

$$L_\epsilon(A) := \left\{ w \in \Sigma^* \mid \exists p \in Q : (q_0, w, \perp) \vdash^* (p, \epsilon, \epsilon) \right\}$$

Beispiel: $L_\epsilon(A) = \{a^n b^n \mid n \in \mathbb{N}, n \geq 1\} \dots$ für jedes gelesene



b muss vorher ein a gelesen worden sein und für jedes a muss später ein b folgen!

Sprachfamilie

Definition 17.1:

- Eine Klasse \mathcal{L} von formalen Sprachen wird **Sprachfamilie** genannt, wenn folgende Bedingungen erfüllt sind:
 1. $\mathcal{L} \neq \emptyset$.
 2. Es existiert ein *abzählbar unendliches* Alphabet Γ , so dass für jede Sprache $L \in \mathcal{L}$ ein *endliches* Alphabet $\Sigma \subseteq \Gamma$ existiert mit $L \subseteq \Sigma^*$.
 3. $\exists L \in \mathcal{L} : L \neq \emptyset$.

Die Klasse *aller* Wortmengen ist **keine** Sprachfamilie, da es für diese unendlich vielen Sprachen kein gemeinsames endliches Alphabet Σ gibt.

die Sprachfamilie Cf

Aus den vorangegangenen Ergebnissen wissen wir:

Die Sprachfamilie Cf :

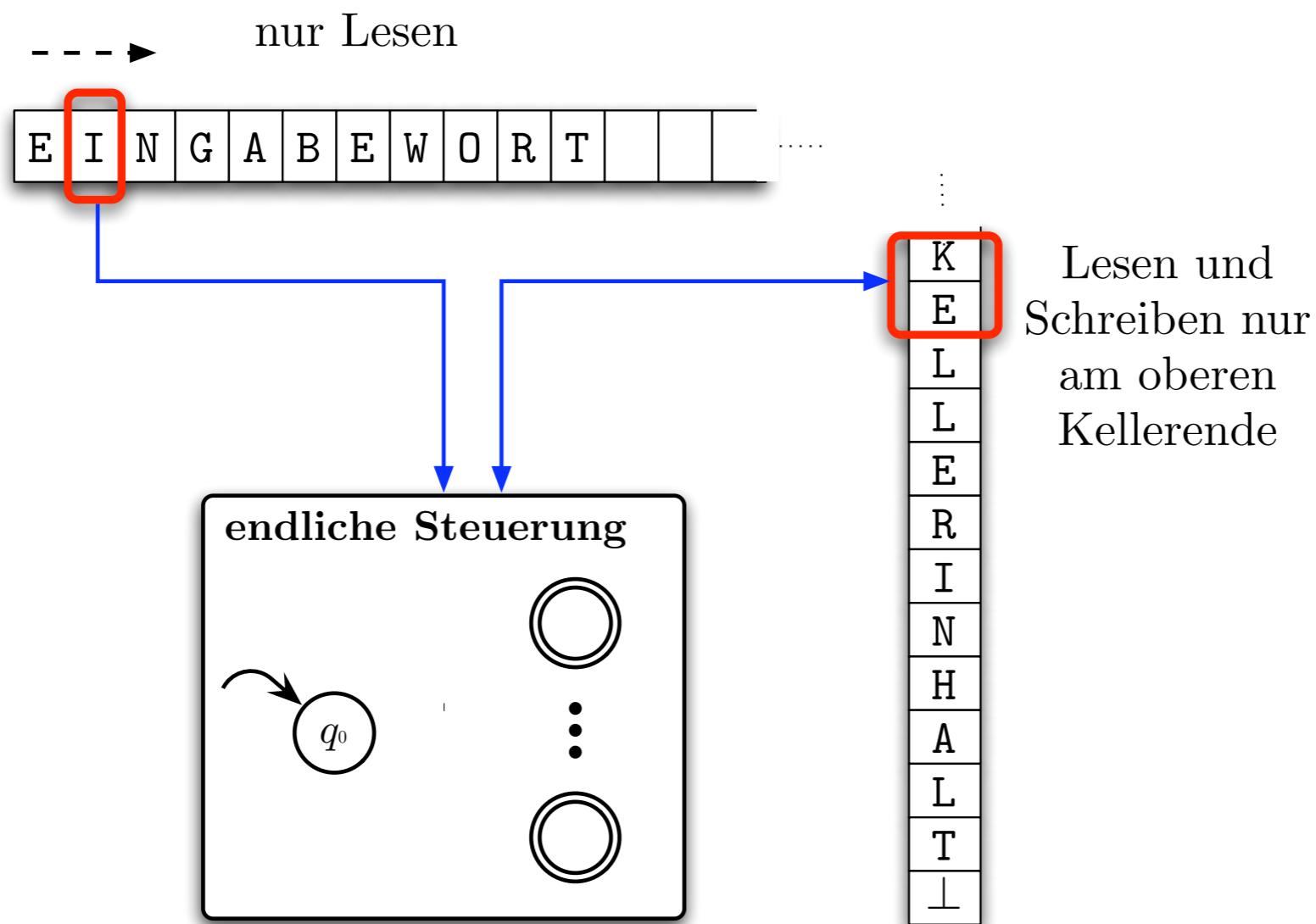
- Korollar:

Folgende Aussagen sind äquivalent:

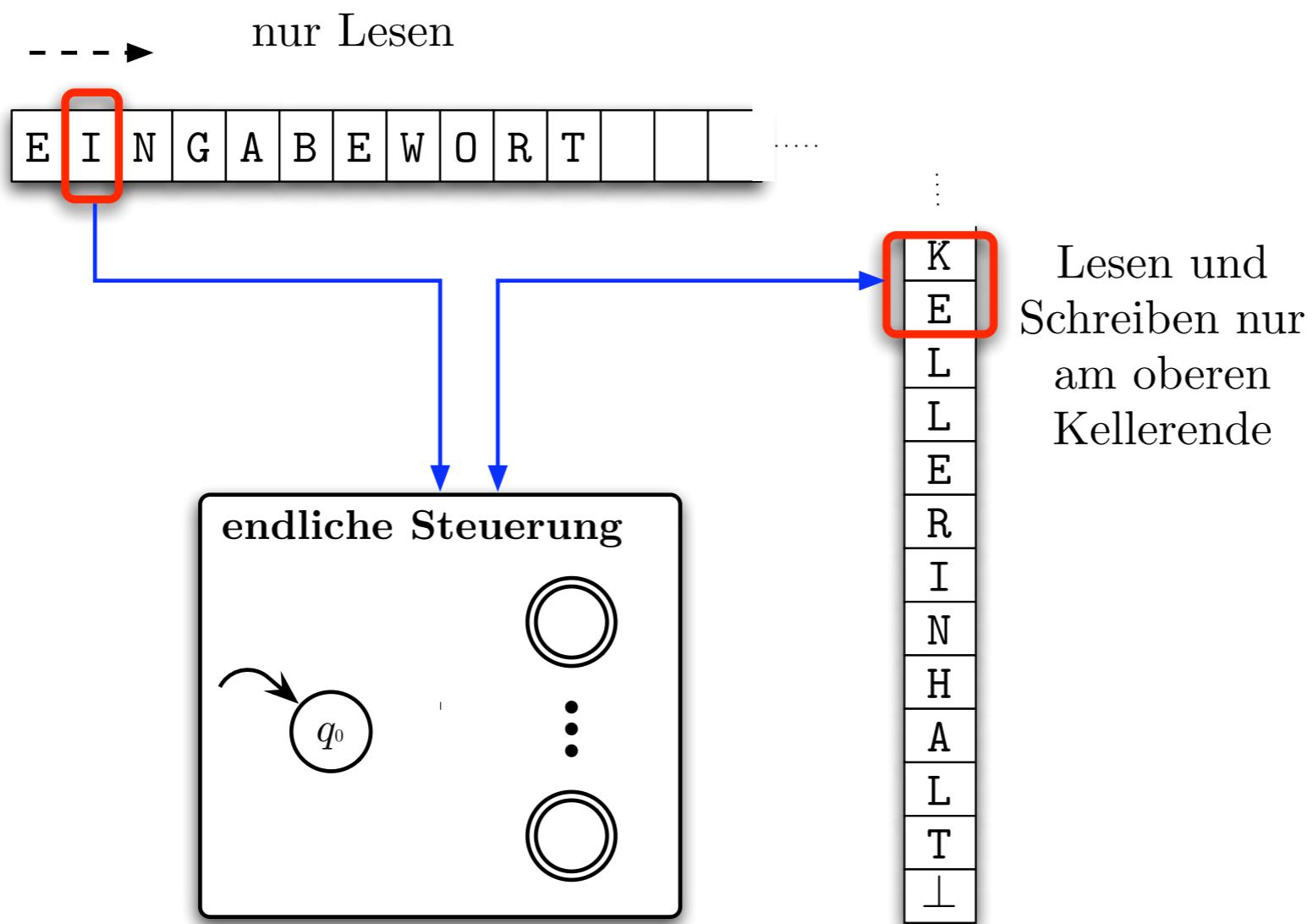
1. $L \in Cf$
2. $L = L(A)$ für einen PDA A
3. $L = L_\epsilon(A)$ für einen PDA A

- Was nun? Eigenschaften deterministischer Kellerautomaten...

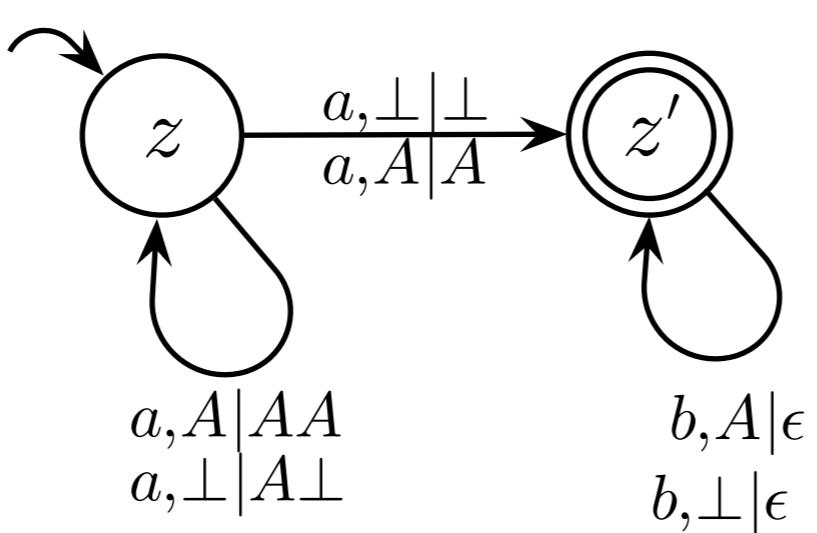
Kellerautomat (informal)



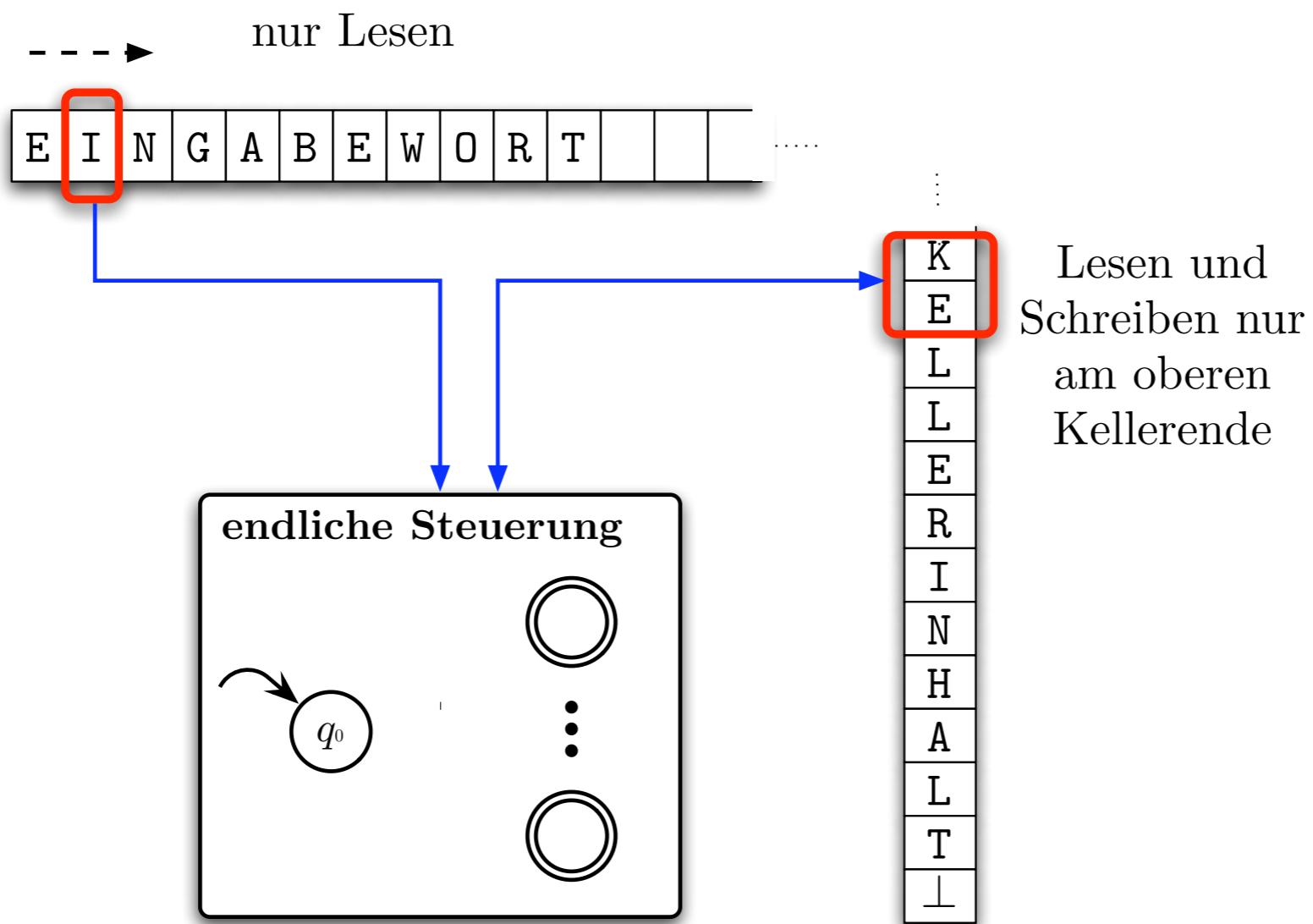
Kellerautomat (informal)



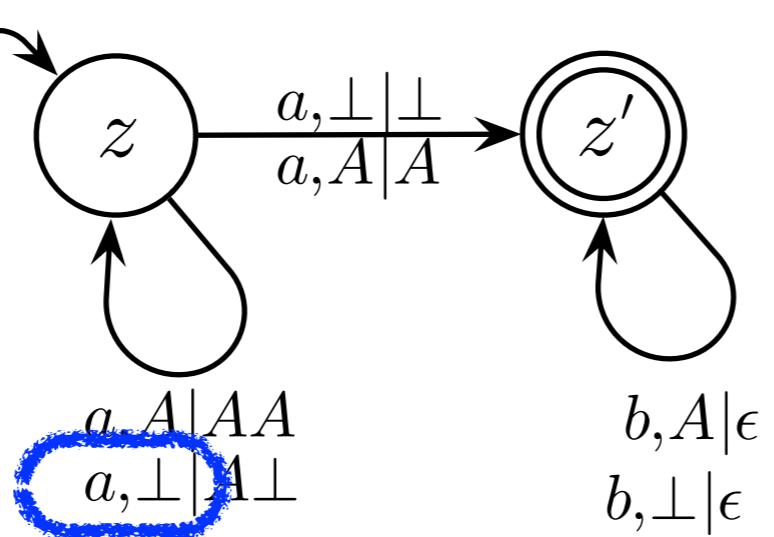
Beispiel PDA:



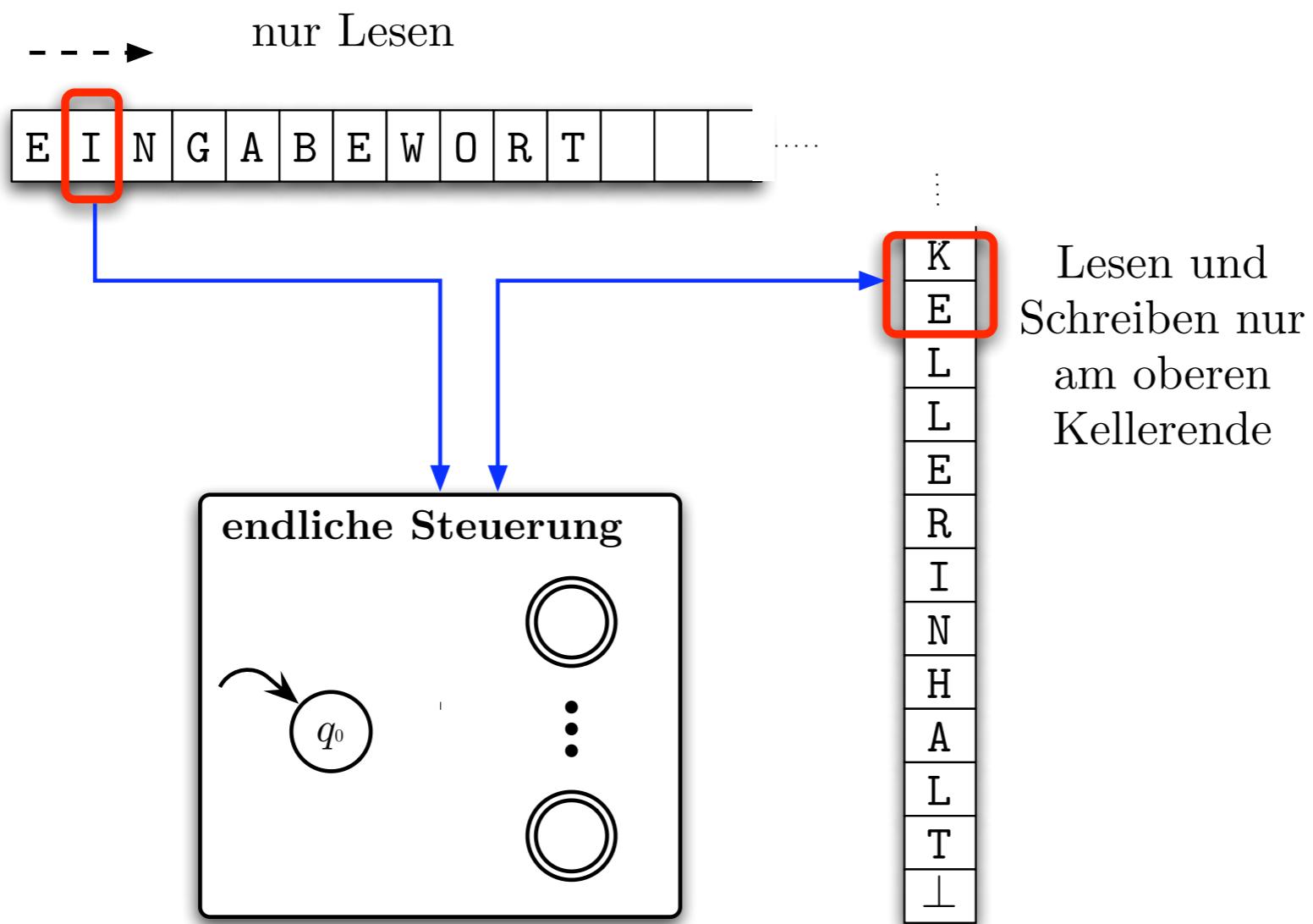
Kellerautomat (informal)



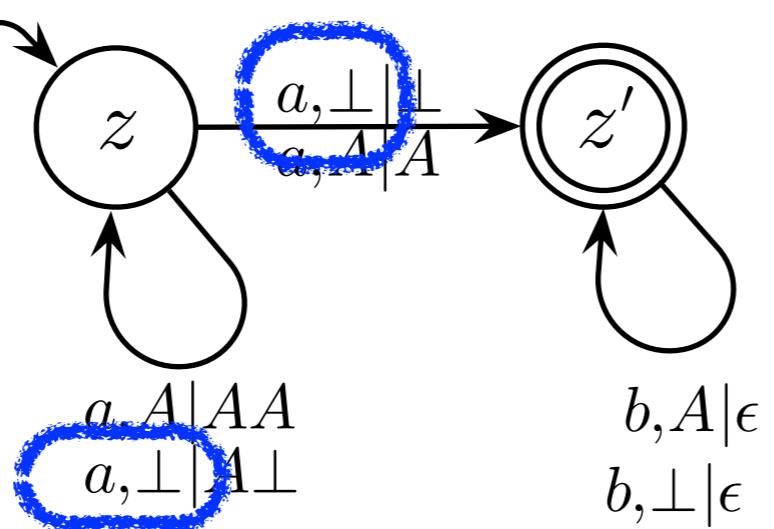
Beispiel PDA:



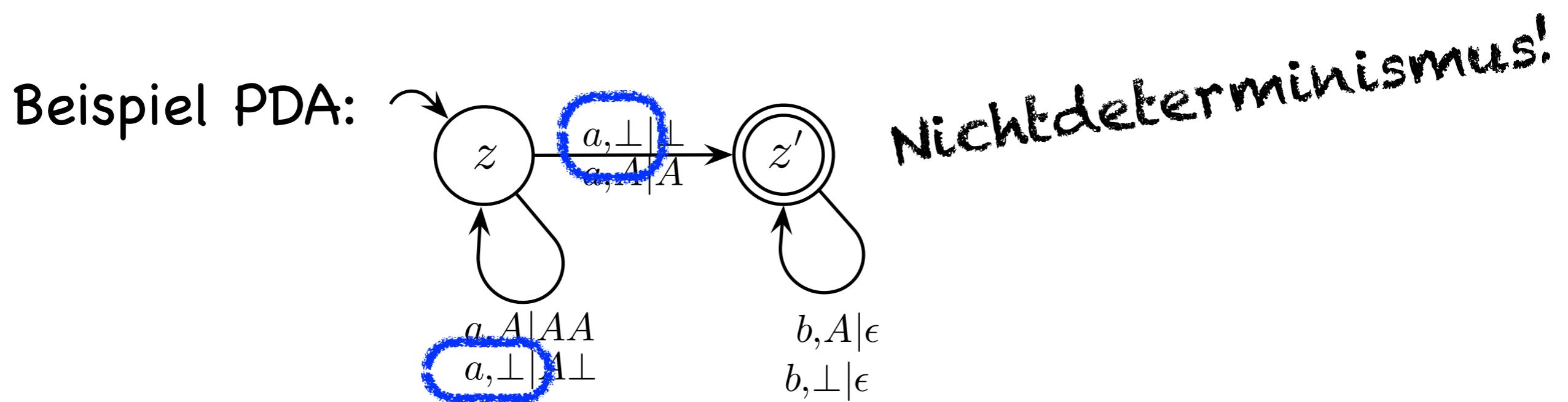
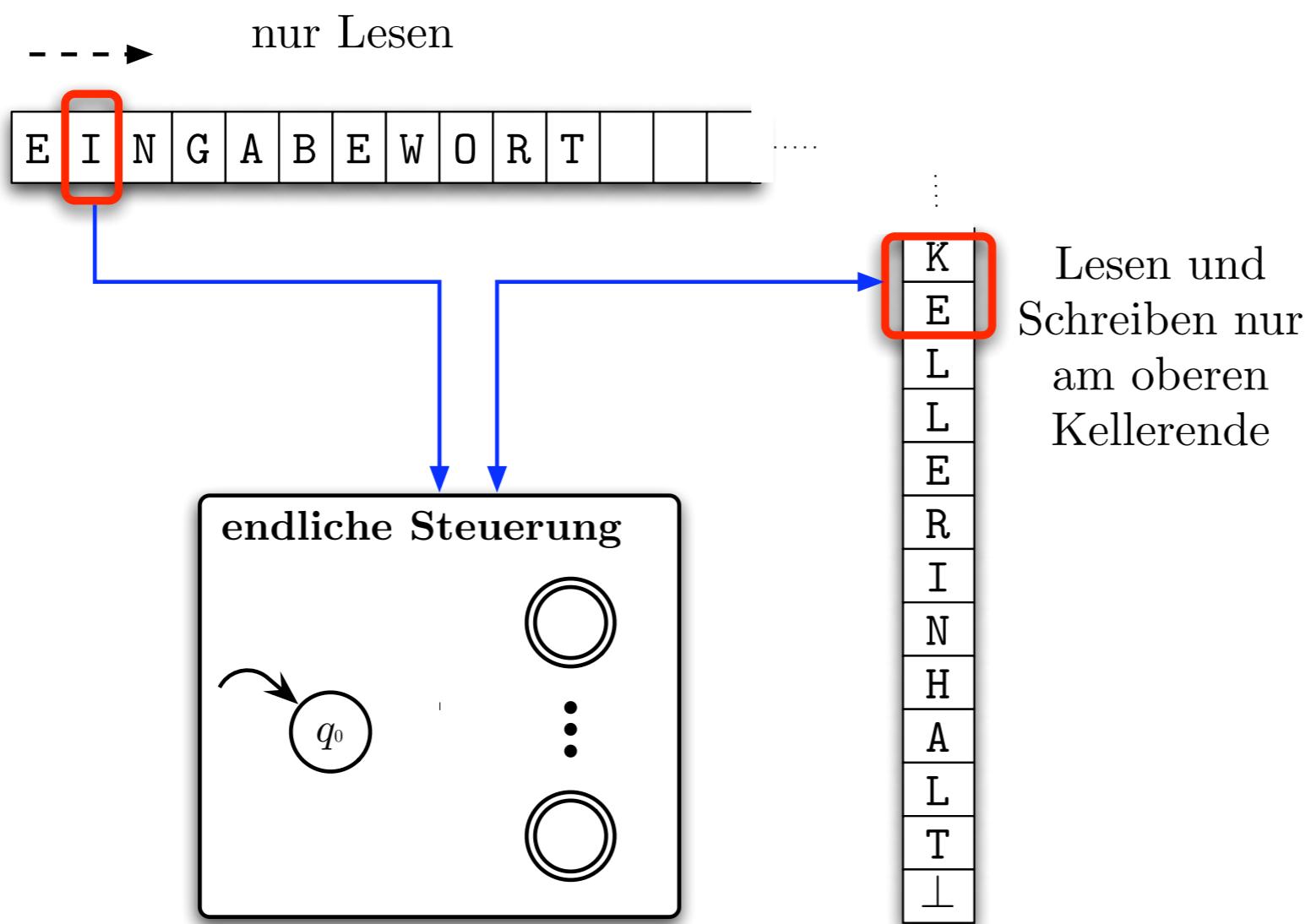
Kellerautomat (informal)



Beispiel PDA:



Kellerautomat (informal)



Deterministischer Kellerautomat (DPDA)

Definition 17.2:

Ein Kellerautomat $A = (Q, \Sigma, \Gamma, \delta, q_0, \perp, F)$ heißt

- **deterministisch**, (Abk.: *DPDA* für *deterministic push-down automaton*, auch: DKA) genau dann, wenn die folgenden Bedingungen erfüllt sind:

1. $|\delta(z, x, y)| \leq 1$ für jedes $(z, x, y) \in Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$.
2. Falls $\delta(z, \epsilon, y) \neq \emptyset$,
so gilt $\forall x \in \Sigma : \delta(z, x, y) = \emptyset$.

Eine Sprache L heißt deterministisch kontextfrei, wenn es einen *DPDA* A gibt mit $L = L(A)$.

Mit $\text{detCf} := \{L(A) \mid A \text{ ein DPDA}\}$ werde die Familie der deterministisch kontextfreien Sprachen bezeichnet.

Eigenschaften deterministisch kontextfreier Sprachen

Definition 17.3:

Eine Sprache $L \subseteq \Sigma^*$ heißt **präfixfrei** gdw.

$$\forall w \in L : \forall v \in \Sigma^* : (wv \in L) \Rightarrow (v = \epsilon).$$

Definition 17.4:

$$\min(L) := \{w \in L \mid (w = uv \wedge u \in L) \Rightarrow (v = \epsilon)\}$$

bezeichnet den **präfixfreien Anteil** der Sprache L .

Beispiel:

Für $M := \{a^m b^n \mid m, n \in \mathbb{N} : n \geq m\}$ ist $\min(M) = \{a^n b^n \mid n \in \mathbb{N}\}$, denn es gibt für $n > m$ mindestens einen echten Präfix eines Wortes $a^m b^n$, der wieder als Wort der Menge M vorkommt. Also muss $n = m$ gelten.

Präfixfreiheit bei $\det\mathcal{Cf}$

Satz 17.1:

Eine deterministisch kontextfreie Sprache $L \in \det\mathcal{Cf}$ ist präfixfrei gdw. ein DPDA A existiert, mit $L = L_\epsilon(A)$.

Beweisidee:

- ⇒ Ein DPDA für L kann kein Wort akzeptieren, nachdem ein Endzustand verlassen wurde, weil L präfixfrei ist. Also: Hinzufügen eines neuen Zustands, der den Keller leert und mit ϵ -Übergang genau von den vormaligen Endzuständen erreichbar ist.
- ⇐ DPDAs können bei leerem Keller (auch \perp ist dort nicht!) keine weiteren Bewegungen vornehmen!

Korollar Die präfixfreien deterministisch kontextfreien Sprachen bilden eine echte Teilfamilie der Familie $\det\mathcal{Cf}$.

Beweis: Die Menge $\{a\}^*\{b\}^* = \{a^n b^m \mid m, n \in \mathbb{N}\}$ ist deterministisch kontextfrei, aber offensichtlich nicht präfixfrei.

Präfixfreiheit bei $\det\mathcal{Cf}$

Satz 17.1:

Eine deterministisch kontextfreie Sprache $L \in \det\mathcal{Cf}$ ist präfixfrei gdw. ein DPDA A existiert, mit $L = L_\epsilon(A)$.

Beweisidee:

- ⇒ Ein DPDA für L kann kein Wort akzeptieren, nachdem ein Endzustand verlassen wurde, weil L präfixfrei ist. Also: Hinzufügen eines neuen Zustands, der den Keller leert und mit ϵ -Übergang genau von den vormaligen Endzuständen erreichbar ist.
- ⇐ DPDAs können bei leerem Keller (auch \perp ist dort nicht!) keine weiteren Bewegungen vornehmen!

Korollar Die präfixfreien deterministisch kontextfreien Sprachen bilden eine echte Teilfamilie der Familie $\det\mathcal{Cf}$.

Beweis: Die Menge $\{a\}^*\{b\}^* = \{a^n b^m \mid m, n \in \mathbb{N}\}$ ist deterministisch kontextfrei, aber offensichtlich nicht präfixfrei.

Präfixfreiheit bei $\det\mathcal{Cf}$

Satz 17.1:

Eine deterministisch kontextfreie Sprache $L \in \det\mathcal{Cf}$ ist präfixfrei gdw. ein DPDA A existiert, mit $L = L_\epsilon(A)$.

Beweisidee:

- ⇒ Ein DPDA für L kann kein Wort akzeptieren, nachdem ein Endzustand verlassen wurde, weil L präfixfrei ist. Also: Hinzufügen eines neuen Zustands, der den Keller leert und mit ϵ -Übergang genau von den vormaligen Endzuständen erreichbar ist.
- ⇐ DPDAs können bei leerem Keller (auch \perp ist dort nicht!) keine weiteren Bewegungen vornehmen!

Korollar Die präfixfreien deterministisch kontextfreien Sprachen bilden eine echte Teilstammfamilie der Familie $\det\mathcal{Cf}$.

Beweis: Die Menge $\{a\}^*\{b\}^* = \{a^n b^m \mid m, n \in \mathbb{N}\}$ ist deterministisch kontextfrei, aber offensichtlich nicht präfixfrei.

Kurz: „ $L_\epsilon(\text{DPDA}) \neq L(\text{DPDA})$ “

wichtige Eigenschaft der Familie $\det \mathcal{Cf}$

Satz 17.2:

Für $L \in \det \mathcal{Cf}$ ist auch $\min(L) \in \det \mathcal{Cf}$.

Beweisidee: Kanten aus Endzuständen entfernen.

Satz 17.3:

$\det \mathcal{Cf} \wedge \mathcal{R} \subseteq \det \mathcal{Cf}$, d.h., die Familie der deterministisch kontextfreien Sprachen ist gegenüber Durchschnittsbildung mit regulären Mengen abgeschlossen.

Beweisidee: sog. Produktautomat des DFA 's für R mit endlicher Kontrolle des $DPDA$'s

\wedge wird oben als Operator zwischen Sprachfamilien benutzt. Die Bedeutung ist dann:

$$\mathcal{L} \wedge \mathcal{K} := \{L \cap K \mid L \in \mathcal{L} \text{ und } K \in \mathcal{K}\}.$$

wichtige Eigenschaft der Familie $\det \mathcal{Cf}$

Satz 17.2:

Für $L \in \det \mathcal{Cf}$ ist auch $\min(L) \in \det \mathcal{Cf}$.

Beweisidee: Kanten aus Endzuständen entfernen.

Satz 17.3:

$\det \mathcal{Cf} \wedge \mathcal{R}eg \subseteq \det \mathcal{Cf}$, d.h., die Familie der deterministisch kontextfreien Sprachen ist gegenüber Durchschnittsbildung mit regulären Mengen abgeschlossen.

Beweisidee: sog. Produktautomat des DFA 's für R mit endlicher Kontrolle des $DPDA$'s

\wedge wird oben als Operator zwischen Sprachfamilien benutzt. Die Bedeutung ist dann:

$$\mathcal{L} \wedge \mathcal{K} := \{L \cap K \mid L \in \mathcal{L} \text{ und } K \in \mathcal{K}\}.$$

eine kontextfreie Sprache, die nicht deterministisch ist

Satz 17.4:

Die kontextfreie Sprache $PAL := \{ww^{\text{rev}} \mid w \in \{a,b\}^*\}$ ist kontextfrei und eindeutig, aber nicht deterministisch.

Beweis:

- Eindeutigkeit: $S \longrightarrow aSa \mid bSb \mid \lambda$
- Wäre nun $PAL \in \text{det Cf}$, dann wäre auch
$$K := PAL \cap \left[(ab)^+ (ba)^* (ab)^* (ba)^+ \cup (ab)^* (ba)^+ (ab)^+ (ba)^* \right] \in \text{det Cf}.$$
- Dann ist auch $\min(K)$ deterministisch kontextfrei.
- Nun ist:
$$\min(K) = \{(ab)^m (ba)^n (ab)^n (ba)^m \mid 0 \leq n \leq m\}$$
- Diese Sprache ist nicht einmal mehr kontextfrei, wie man mit dem $uvwxy$ -Theorem zeigen kann.
- Also kann PAL selbst nicht deterministisch kontextfrei gewesen sein.

eine Abschlusseigenschaft von $\det \mathcal{C}f$

Satz 17.5:

$\det \mathcal{C}f$ ist gegen Komplementbildung abgeschlossen, d.h.:
 $\forall L \subseteq \Sigma^*, L \in \det \mathcal{C}f : \overline{L} := \Sigma^* \setminus L \in \det \mathcal{C}f.$

Beweisidee: Endzustände mit Nicht-Endzuständen zu vertauschen reicht hier nicht aus!

Gründe für das Nicht-Akzeptieren einer Eingabe beim *DPDA*:

1. Die Rechnung bricht ab, weil der Keller leer ist.
2. Das Wort w wird mangels Folgekonfiguration nicht vollständig gelesen.
3. Der *DPDA* gerät in eine Endlosschleife, ohne weitere Symbole von der Eingabe zu lesen.
4. Erfolgsrechnung, bei der der *DPDA* ohne weiteres Lesen der Eingabe abwechselnd in End- und Nicht-Endzustände gerät.

Beweisideen

Zu 1: Neues Kellerbodenzeichen vor das alte setzen.

Zu 2: Einführen einer Senke, in die bisher in anderen Zuständen nicht definierte Folgekonfigurationen führen.

Zu 3: Zwei Unter-Fälle:

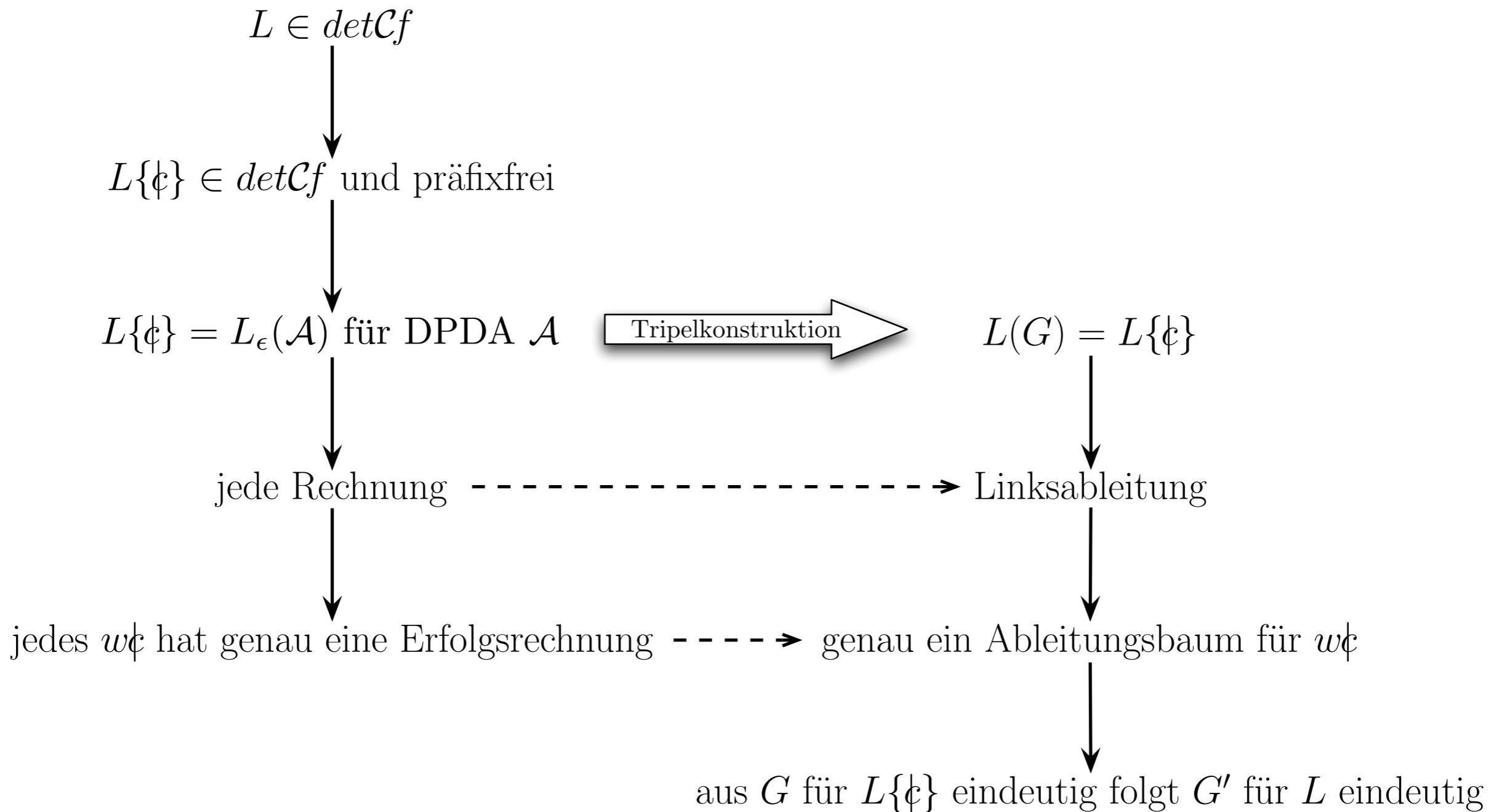
- a): In der ϵ -Schleife wird der Keller nie kürzer, aber ein Endzustand kommt vor. Neuer Endzustand und Schleifenanfangskante dahin „umbiegen“, Keller leeren.
- b): Falls die Konfiguration xq der Beginn einer ϵ -Schleife ist, in der niemals ein Endzustand vorkommt: Keller leeren.

Fall 4: Zu kompliziert ... **für die Vorlesung ...**

wichtig für Syntaxanalyse

Satz 17.6:

Jede Sprache aus $\det \mathcal{C}f$ ist eindeutig.



Eindeutige Grammatiken

Eindeutige Grammatiken

Wenn wir eine Sprache mit einem DPDA erkennen können, dann ist sie eindeutig. (Satz 17.6)

Eindeutige Grammatiken

Wenn wir eine Sprache mit einem DPDA erkennen können, dann ist sie eindeutig. (Satz 17.6)

Typischerweise wird eine Programmiersprache aber anhand einer **Grammatik** definiert.

Eindeutige Grammatiken

Wenn wir eine Sprache mit einem DPDA erkennen können, dann ist sie eindeutig. (Satz 17.6)

Typischerweise wird eine Programmiersprache aber anhand einer **Grammatik** definiert.

Frage: Gibt es einen Typ von Grammatiken, die stets eindeutig sind und zu denen man stets einen parsenden DPDA konstruieren kann?

Eindeutige Grammatiken

Wenn wir eine Sprache mit einem DPDA erkennen können, dann ist sie eindeutig. (Satz 17.6)

Typischerweise wird eine Programmiersprache aber anhand einer **Grammatik** definiert.

Frage: Gibt es einen Typ von Grammatiken, die stets eindeutig sind und zu denen man stets einen parsenden DPDA konstruieren kann?

Ja!

Eindeutige Grammatiken

Wenn wir eine Sprache mit einem DPDA erkennen können, dann ist sie eindeutig. (Satz 17.6)

Typischerweise wird eine Programmiersprache aber anhand einer **Grammatik** definiert.

Frage: Gibt es einen Typ von Grammatiken, die stets eindeutig sind und zu denen man stets einen parsenden DPDA konstruieren kann?

Ja!

Das sind die LR(k) Grammatiken.

Was tut ein Compiler?

```
| IF | i |<=| n | THEN | a | (/ | i | + | 1 | /) |:=| Ø | ELSE | i |:=| i | + | 1 | ; |
```

↓ ↓
Aktion, Pointer zu „≤“ syntaktisch, Pointer zu „]“
↓
Identifier, Pointer zur Adresse von „i“

Aufbau eines Compilers (1): (Lexikalische Analyse)

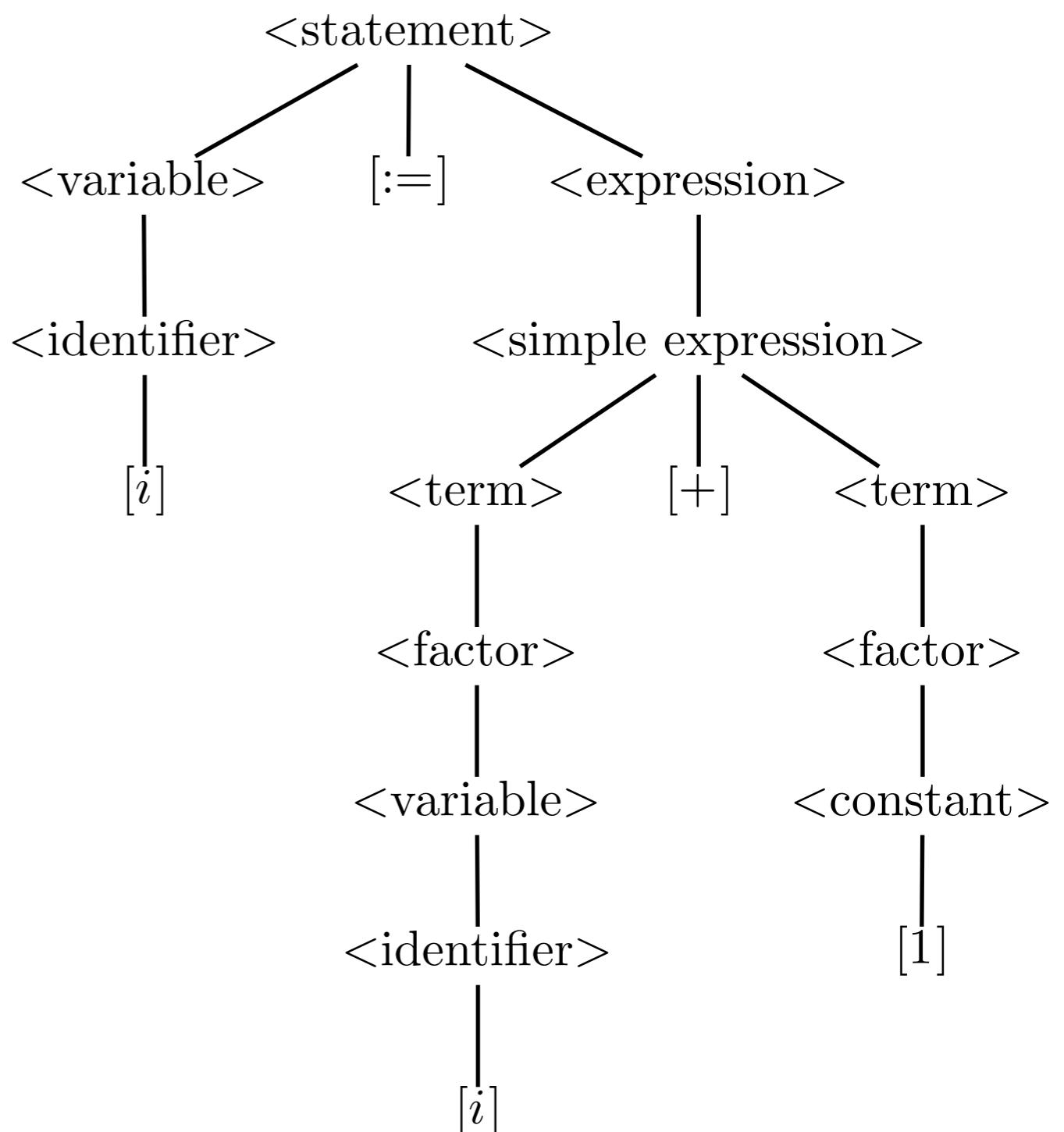
- Das Quellprogramm wird analysiert:
- Symbole einlesen, Leerzeichen unterdrücken, Korrektheit sprachinterner Schlüsselwörter prüfen und *Token* ersetzen.
- Pointer zu weiteren Informationen (*Wert*, *Tokentyp* und *ASCII-Wert*) anlegen
- kontextabhängige (z.B. *nicht deklarierte Variablen*) und syntaktische Fehler suchen.

Aufbau eines Compilers

Aufbau eines Compilers (2): (Syntaktische Analyse)

Einen Ableitungsbaum zu der von der lexikalischen Analyse erstellten Token-Kette erzeugen.

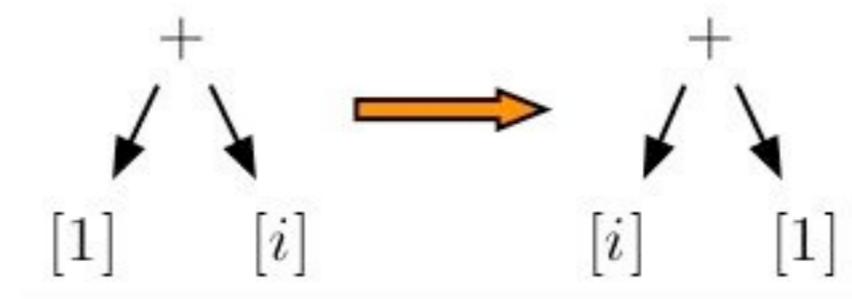
Bsp.: $|i| := |i| + |1|$



Aufbau eines Compilers

Aufbau eines Compilers (3): (Semantische Analyse und Codegenerierung)

- Synthese-Phase: es wird ein **abstrakter Syntaxbaum** aus dem Ableitungsbaum konstruiert
- ähnlich der Baumdarstellung eines arithmetischen Ausdrucks
- „Normalisierung“



Aufbau eines Compilers

Aufbau eines Compilers (4): (Codeoptimierung)

- Beispiel: Summe „+“ mit zwei Operanden.
- Statt „,+1“ einen Befehl „increment“ verwenden.
- Vereinfachungen von einfachen Laufschleifen durch Splitten, Zusammenfassen, Abwickeln oder Herausziehen von Anweisungen können vorgenommen werden.

Eine einfache CFG für „Expression“:

$$\begin{array}{rcl} E & \longrightarrow & E + T \mid T \\ T & \longrightarrow & T * F \mid F \\ F & \longrightarrow & (E) \mid i \end{array}$$

LR(1)-Analyse mit Kellerautomat

$$E \longrightarrow E + T \mid T$$

$$T \longrightarrow T * F \mid F$$

$$F \longrightarrow (E) \mid i$$

Kellerboden ↓ ↓ Vorausschau

Kellerinhalt		zu lesende Eingabe	
		<i>i</i> + <i>i</i> * <i>i</i>	<i>Start</i>
	<i>i</i>	+ <i>i</i> * <i>i</i>	<i>shift</i>
	<i>F</i>	+ <i>i</i> * <i>i</i>	<i>reduce</i>
	<i>T</i>	+ <i>i</i> * <i>i</i>	<i>reduce</i>
	<i>E</i>	+ <i>i</i> * <i>i</i>	<i>reduce</i>
	+ <i>E</i>	<i>i</i> * <i>i</i>	<i>shift</i>
	<i>i</i> + <i>E</i>	* <i>i</i>	<i>shift</i>
	<i>F</i> + <i>E</i>	* <i>i</i>	<i>reduce</i>
	<i>T</i> + <i>E</i>	* <i>i</i>	<i>reduce</i> *)
	* <i>T</i> + <i>E</i>	<i>i</i>	<i>shift</i>
	<i>i</i> * <i>T</i> + <i>E</i>		<i>shift</i>
	<i>F</i> * <i>T</i> + <i>E</i>		<i>reduce</i>
	<i>T</i> + <i>E</i>		<i>reduce</i>
	<i>E</i>		<i>reduce</i>
			<i>accept</i>

LR(1)-Analyse mit Kellerautomat

$$E \longrightarrow E + T \mid T$$

$$T \longrightarrow T * F \mid F$$

$$F \longrightarrow (E) \mid i$$

Kellerboden ↓ ↓ Vorausschau

Kellerinhalt		zu lesende Eingabe	
	<i>i</i>	<i>i</i> + <i>i</i> * <i>i</i> + <i>i</i> * <i>i</i>	<i>Start shift</i>
<i>i</i>	+	<i>i</i> * <i>i</i>	<i>shift</i>
<i>F</i>	+	<i>i</i> * <i>i</i>	<i>reduce</i>
<i>T</i>	+	<i>i</i> * <i>i</i>	<i>reduce</i>
<i>E</i>	+	<i>i</i> * <i>i</i>	<i>reduce</i>
+ <i>E</i>	<i>i</i>	* <i>i</i>	<i>shift</i>
<i>i</i> + <i>E</i>	*	<i>i</i>	<i>shift</i>
<i>F</i> + <i>E</i>	*	<i>i</i>	<i>reduce</i>
<i>T</i> + <i>E</i>	*	<i>i</i>	<i>reduce</i> *)
* <i>T</i> + <i>E</i>	<i>i</i>		<i>shift</i>
<i>i</i> * <i>T</i> + <i>E</i>			<i>shift</i>
<i>F</i> * <i>T</i> + <i>E</i>			<i>reduce</i>
<i>T</i> + <i>E</i>			<i>reduce</i>
<i>E</i>			<i>reduce</i>
			<i>accept</i>

LR(1)-Analyse mit Kellerautomat

$$E \longrightarrow E + T \mid T$$

$$T \longrightarrow T * F \mid F$$

$$F \longrightarrow (E) \mid i$$

Kellerboden ↓ ↓ Vorausschau

Kellerinhalt		zu lesende Eingabe	
		$i + i * i$	<i>Start</i>
i	i	$+ i * i$	<i>shift</i>
i	$+$	$i * i$	<i>shift</i>
F	$+$	$i * i$	<i>reduce</i>
T	$+$	$i * i$	<i>reduce</i>
E	$+$	$i * i$	<i>reduce</i>
$+ E$	i	$* i$	<i>shift</i>
$i + E$	$*$	i	<i>shift</i>
$F + E$	$*$	i	<i>reduce</i>
$T + E$	$*$	i	<i>reduce *)</i>
$* T + E$	i		<i>shift</i>
$i * T + E$			<i>shift</i>
$F * T + E$			<i>reduce</i>
$T + E$			<i>reduce</i>
E			<i>reduce</i>
			<i>accept</i>

LR(1)-Analyse mit Kellerautomat

$$E \longrightarrow E + T \mid T$$

$$T \longrightarrow T * F \mid F$$

$$F \longrightarrow (E) \mid i$$

Kellerboden ↓ ↓ Vorausschau

Kellerinhalt		zu lesende Eingabe	
		$i + i * i$	<i>Start</i>
i	i	$+ i * i$	<i>shift</i>
i	$+$	$i * i$	<i>shift</i>
F	F	$i * i$	<i>reduce</i>
E	$+$	$i * i$	<i>reduce</i>
$+ E$	i	$* i$	<i>shift</i>
$i + E$	$*$	i	<i>shift</i>
$F + E$	$*$	i	<i>reduce</i>
$T + E$	$*$	i	<i>reduce *)</i>
$* T + E$	i		<i>shift</i>
$i * T + E$			<i>shift</i>
$F * T + E$			<i>reduce</i>
$T + E$			<i>reduce</i>
E			<i>reduce</i>
			<i>accept</i>

LR(1)-Analyse mit Kellerautomat

$$E \longrightarrow E + T \mid T$$

$$T \longrightarrow T * F \mid F$$

$$F \longrightarrow (E) \mid i$$

Kellerboden ↓ ↓ Vorausschau

Kellerinhalt		zu lesende Eingabe	
		$i + i * i$	<i>Start</i>
	i	$+ i * i$	<i>shift</i>
	i	$i * i$	<i>shift</i>
F	$+$	$i * i$	<i>reduce</i>
T	$+$	$i * i$	<i>reduce</i>
	E	$i * i$	<i>reduce</i>
	$+ E$	$i * i$	<i>shift</i>
i	$+ E$	i	<i>shift</i>
F	$+ E$	i	<i>reduce</i>
T	$+ E$	i	<i>reduce *)</i>
	$* T$	i	<i>shift</i>
i	$* T$	i	<i>shift</i>
F	$* T$	i	<i>reduce</i>
	T	i	<i>reduce</i>
	E		<i>reduce</i>
			<i>accept</i>

LR(1)-Analyse mit Kellerautomat

$$E \longrightarrow E + T \mid T$$

$$T \longrightarrow T * F \mid F$$

$$F \longrightarrow (E) \mid i$$

Kellerboden ↓ ↓ Vorausschau

Kellerinhalt		zu lesende Eingabe	
	<i>i</i>	<i>i</i> + <i>i</i> * <i>i</i>	<i>Start</i>
	<i>i</i>	+ <i>i</i> * <i>i</i>	<i>shift</i>
<i>i</i>	+	<i>i</i> * <i>i</i>	<i>shift</i>
<i>F</i>	+	<i>i</i> * <i>i</i>	<i>reduce</i>
<i>T</i>	+	<i>i</i> * <i>i</i>	<i>reduce</i>
<i>E</i>	+	<i>i</i> * <i>i</i>	<i>reduce</i>
+ <i>E</i>	<i>i</i>	* <i>i</i>	<i>shift</i>
<i>i</i> + <i>E</i>	*	<i>i</i>	<i>shift</i>
<i>F</i> + <i>E</i>	*	<i>i</i>	<i>reduce</i>
<i>T</i> + <i>E</i>	*	<i>i</i>	<i>reduce</i> *)
* <i>T</i> + <i>E</i>	<i>i</i>		<i>shift</i>
<i>i</i> * <i>T</i> + <i>E</i>			<i>shift</i>
<i>F</i> * <i>T</i> + <i>E</i>			<i>reduce</i>
<i>T</i> + <i>E</i>			<i>reduce</i>
<i>E</i>			<i>reduce</i>
			<i>accept</i>

LR(1)-Analyse mit Kellerautomat

$E \longrightarrow E + T \mid T$
 $T \longrightarrow T * F \mid F$
 $F \longrightarrow (E) \mid i$

Kellerboden ↓ ↓ Vorausschau

Kellerinhalt		zu lesende Eingabe	
		$i + i * i$	<i>Start</i>
	i	$+ i * i$	<i>shift</i>
i	$+$	$i * i$	<i>shift</i>
F	$+$	$i * i$	<i>reduce</i>
T	$+$	$i * i$	<i>reduce</i>
<hr/>			
	E	$i * i$	<i>reduce</i>
	$+ E$	$i * i$	<i>shift</i>
i	$+ E$	i	<i>shift</i>
F	$+ E$	i	<i>reduce</i>
T	$+ E$	i	<i>reduce</i> $*)$
	$* T + E$	i	<i>shift</i>
i	$* T + E$	i	<i>shift</i>
F	$* T + E$	i	<i>reduce</i>
	$T + E$	i	<i>reduce</i>
	E		<i>reduce</i>
			<i>accept</i>

LR(1)-Analyse mit Kellerautomat

Kellerboden ↓ ↓ Vorausschau

$$\begin{array}{lcl}
 E & \longrightarrow & E + T \mid T \\
 T & \longrightarrow & T * F \mid F \\
 F & \longrightarrow & (E) \mid i
 \end{array}$$

Kellerinhalt		zu lesende Eingabe	
		$i + i * i$	<i>Start</i>
	i	$+ i * i$	<i>shift</i>
	$i +$	$i * i$	<i>shift</i>
	$F +$	$i * i$	<i>reduce</i>
	$T +$	$i * i$	<i>reduce</i>
	$E +$	$i * i$	<i>reduce</i>
$+ E$	i	$* i$	<i>shift</i>
$i + E$	$*$	i	<i>shift</i>
$F + E$	$*$	i	<i>reduce</i>
$T + E$	$*$	i	<i>reduce *)</i>
$* T + E$	i		<i>shift</i>
$i * T + E$			<i>shift</i>
$F * T + E$			<i>reduce</i>
$T + E$			<i>reduce</i>
E			<i>reduce</i>
			<i>accept</i>

Warum LR-Parsing?

LR-Parsing ist aus folgenden Gründen attraktiv:

- Jede deterministische kontextfreie Sprache kann mit diesem Bottom-up Verfahren analysiert und übersetzt werden
- Verfahren kann effizient implementiert werden.
- Syntaktische Fehler können so früh wie möglich erkannt werden.
- Viele existierende Compiler-Generatoren bieten Hilfen an, wenn eine Grammatik keine LR-Grammatik ist, Mehrdeutigkeiten aufweist und daher geändert werden muss.
- Die LR-Parsingmethode ist das am besten ausgebauten Verfahren, und beinahe jede Programmiersprachensyntax besitzt eine $LR(k)$ -Grammatik.

Ergebnisse zu deterministischen CFG's

- Jede $\text{LR}(k)$ -Grammatik ist eindeutig.
- Folgende drei Aussagen sind äquivalent:
 1. L ist präfixfreie, deterministisch kontextfreie Sprache
 2. $L = L(G)$ für eine $\text{LR}(0)$ -Grammatik
 3. $L = L_\epsilon(A)$ für einen DPDA A
- Folgende drei Aussagen sind äquivalent:
 1. L ist deterministisch kontextfrei, d.h. $L = L(A)$ für einen DPDA A .
 2. $L = L(G)$ für eine $\text{LR}(1)$ -Grammatik G .
 3. $L = L(G)$ für eine $\text{LR}(k)$ -Grammatik G für ein $k \geq 1$.
- Zu jedem $k \geq 1$ gibt es eine kontextfreie Grammatik die $\text{LR}(k)$ aber nicht $\text{LR}(k - 1)$ ist.

FGI 1

Automaten, Formale Sprachen,
Berechenbarkeit

Kap. 17

andere Chomsky Grammatiken

Michael Köhler-Bußmeier

Typ-0 Grammatik nach Chomsky

Definition 17.5:

Eine **Typ 0 Grammatik** wird beschrieben durch ein Quadrupel $G = (\Sigma, N, P, S)$ mit:

Σ ist ein endliches Alphabet von **Terminalsymbolen** (auch: Terminalen).

N ist ein mit Σ disjunktes endliches Alphabet von **Nonterminalen** (auch: Nichtterminalen, Variablen, metalinguistischen Variablen, Hilfszeichen oder syntaktischen Kategorien). $N \cap \Sigma = \emptyset$.

P ist eine endliche Menge von **Produktionen** oder **Regeln**. Es gilt

$$P \subseteq (N \cup \Sigma)^* \cdot N \cdot (N \cup \Sigma)^* \times (N \cup \Sigma)^*.$$

S ist das **Startsymbol** und es gilt $S \in N$.

Typ-0 Grammatik nach Chomsky

Definition 17.5:

Eine **Typ 0 Grammatik** wird beschrieben durch ein Quadrupel $G = (\Sigma, N, P, S)$ mit:

Σ ist ein endliches Alphabet von **Terminalsymbolen** (auch: Terminalen).

N ist ein mit Σ disjunktes endliches Alphabet von **Nonterminalen** (auch: Nichtterminalen, Variablen, metalinguistischen Variablen, Hilfszeichen oder syntaktischen Kategorien). $N \cap \Sigma = \emptyset$.

P ist eine endliche Menge von **Produktionen** oder **Regeln**. Es gilt

$$P \subseteq (N \cup \Sigma)^* \cdot \underline{N} \cdot (N \cup \Sigma)^* \times (N \cup \Sigma)^*.$$

S ist das **Startsymbol** und es gilt $S \in N$.

Konvention, wie bei CFG's

- Als **Nonterminale** werden meist *Großbuchstaben* verwendet (A, B, C, \dots).
- Als **Terminale** werden meist *Kleinbuchstaben* verwendet (a, b, c, \dots).
- $(v, w) \in P$ wird üblicherweise als $v \longrightarrow w$ geschrieben.
- Alternativen werden mit $|$ abgekürzt.
- Die Mengen N und Σ können von den Regeln, d.h. der Menge P abgelesen werden. Es reicht bei Einhalten dieser Konventionen deshalb meist, die **Regelmenge** und das **Startsymbol** anzugeben.

Ableiten bei Typ-0 Grammatik

Definition 17.6:

Die **einschrittige Ableitung** eines Wortes v aus einem Wort u mittels der Produktionen einer Typ-0 Grammatik G wird notiert als:

$$u \xrightarrow{G} v$$

- Die Relation $\xrightarrow{G} \subseteq (N \cup \Sigma)^* \times (N \cup \Sigma)^*$ ist definiert durch:

$$u \xrightarrow{G} v \quad \text{gdw. } \exists u_1, u_2, w_l, w_r \in (N \cup \Sigma)^*: \\ \exists [w_l \longrightarrow w_r] \in P : u = u_1 [w_l] u_2 \wedge v = u_1 [w_r] u_2$$

- Als **Ableitungsrelation** wird die reflexive, transitive Hülle der Relation \xrightarrow{G} bezeichnet, die als
$$\xrightarrow{G}^* \subseteq (V \cup \Sigma)^* \times (V \cup \Sigma)^*$$
 geschrieben wird.

erzeugte Sprache

Definition 17.7:

Sei $G = (\Sigma, N, P, S)$ eine Typ-0 Grammatik. Die von G generierte oder **erzeugte Sprache** ist:

$$L(G) := \{w \in \Sigma^* \mid S \xrightarrow[G]{*} w\}.$$

Mit Typ-0 Grammatiken können Sprachen erzeugt werden,
die nicht mehr kontextfrei sind.

Spezielle Varianten von Typ-0 Grammatiken sind zwar
ebenfalls mächtiger als CFG's, aber weniger
ausdrucksstark als kontextfreie Grammatiken!

Typ-1 Grammatiken

Definition 17.8:

Sei $G = (\Sigma, N, P, S)$ eine Typ-0 Grammatik. G heißt: kontext-sensitiv, monoton, oder vom Typ-1, wenn

1. $S \rightarrow \epsilon$ in P die einzige löschende Produktion ist,
2. $|w_l| \leq |w_r|$ für jede Regel $(w_l, w_r) \in P \setminus \{(S, \epsilon)\}$ gilt,
3. in keiner Regel $(w_l, w_r) \in P$ kommt S in w_r vor.

Typ-1 Grammatiken

Definition 17.8:

Sei $G = (\Sigma, N, P, S)$ eine Typ-0 Grammatik. G heißt: kontextsensitiv, monoton, oder vom Typ-1, wenn

1. $S \rightarrow \epsilon$ in P die einzige löschende Produktion ist,
2. $|w_l| \leq |w_r|$ für jede Regel $(w_l, w_r) \in P \setminus \{(S, \epsilon)\}$ gilt,
3. in keiner Regel $(w_l, w_r) \in P$ kommt S in w_r vor.

Satz: Jede kontextfreie Sprache ist immer auch kontextsensitiv.

Typ-1 Grammatiken

Definition 17.8:

Sei $G = (\Sigma, N, P, S)$ eine Typ-0 Grammatik. G heißt: kontextsensitiv, monoton, oder vom Typ-1, wenn

1. $S \rightarrow \epsilon$ in P die einzige löschende Produktion ist,
2. $|w_l| \leq |w_r|$ für jede Regel $(w_l, w_r) \in P \setminus \{(S, \epsilon)\}$ gilt,
3. in keiner Regel $(w_l, w_r) \in P$ kommt S in w_r vor.

Satz: Jede kontextfreie Sprache ist immer auch kontextsensitiv.

Beweis: Zu jeder kontextfreien Grammatik G existiert eine äquivalente Grammatik G' , die auch kontextsensitiv ist. (Wieso?)

Beispiele

Seien $G_i := (\{a, b, c\}, \{S_i, A, B, C\}, P_i, S_i)$ für $i \in \{1, 2\}$ Grammatiken mit:

$P_1 :$

$$\begin{array}{l}
 S_1 \longrightarrow aS_1BC \quad | \quad aBC \\
 CB \longrightarrow BC \\
 aB \longrightarrow ab \\
 bB \longrightarrow bb \\
 bC \longrightarrow bc \\
 cC \longrightarrow cc
 \end{array}$$

$P_2 :$

$$\begin{array}{l}
 S_2 \longrightarrow AS_2B \quad | \quad \epsilon \\
 A \longrightarrow a \\
 B \longrightarrow b
 \end{array}$$

S₂ kommt auch rechts vor und erzeugt ε:
für Typ-2 erlaubt,
für Typ-1 nicht erlaubt!

- Sind G_1 und G_2 kontextsensitiv?
- Falls nicht, sind $L(G_1)$ und $L(G_2)$ kontextsensitiv?
- $L(G_1) = \{a^n b^n c^n \mid n \in \mathbb{N}, n \geq 1\}$ und $L(G_2) = ?$

Bezeichnungen

Partiell wird hier wiederholt:

Definition 17.9:

- Die Familie der kontextfreien Sprachen wird mit $\mathcal{C}f$ bezeichnet.
- Die Familie der kontextsensitiven Sprachen wird mit $\mathcal{C}s$ bezeichnet.
- Die Familie der entscheidbaren Mengen wird mit $\mathcal{R}ec$ bezeichnet.
wird später definiert!
- Die Familie der aufzählbaren Mengen wird mit $\mathcal{R}e$ bezeichnet.

Folgerung

Folgende Beziehungen sind nun bewiesen:

Familie der **regulären Mengen**

- \subsetneq Familie der **kontextfreien Mengen**
- \subsetneq Familie der **kontextsensitiven Mengen**
- \subsetneq Familie der **entscheidbaren Mengen**
- \subsetneq Familie der **aufzählbaren Mengen**
- \subsetneq Familie der **abzählbaren Mengen**
- \neq Familie der **überabzählbaren Mengen**

Folgerung

Folgende Beziehungen sind nun bewiesen:

Familie der **regulären Mengen**

- ⊈ Familie der **kontextfreien Mengen**
- ⊈ Familie der **kontextsensitiven Mengen**
- ⊈ Familie der **entscheidbaren Mengen**
- ⊈ Familie der **aufzählbaren Mengen**
- ⊈ Familie der **abzählbaren Mengen**
- ≠✓ Familie der **überabzählbaren Mengen**

Folgerung

Folgende Beziehungen sind nun bewiesen:

Familie der **regulären Mengen**

- Familie der **kontextfreien Mengen**
- Familie der **kontextsensitiven Mengen**
- Familie der **entscheidbaren Mengen**
- Familie der **aufzählbaren Mengen**
- Familie der **abzählbaren Mengen**
- Familie der **überabzählbaren Mengen**

Folgerung

Folgende Beziehungen sind nun bewiesen:

Familie der **regulären Mengen**

$\subseteq \checkmark$ Familie der **kontextfreien Mengen**

$\subseteq \checkmark$ Familie der **kontextsensitiven Mengen**

\subseteq Familie der **entscheidbaren Mengen**

\subseteq Familie der **aufzählbaren Mengen**

\subseteq Familie der **abzählbaren Mengen**

$\neq \checkmark$ Familie der **überabzählbaren Mengen**

Folgerung

Folgende Beziehungen sind nun bewiesen:

Familie der **regulären Mengen**

$\subseteq \checkmark$ Familie der **kontextfreien Mengen**

$\subseteq \checkmark$ Familie der **kontextsensitiven Mengen**

\subseteq Familie der **entscheidbaren Mengen**

\subseteq Familie der **aufzählbaren Mengen**

\subseteq Familie der **abzählbaren Mengen**

$\neq \checkmark$ Familie der **überabzählbaren Mengen**