



# F2 — Automaten und formale Sprachen

Matthias Jantzen

(nach und mit Folienvorlagen von Berndt Farwer)

Fachbereich Informatik

AB „Theoretische Grundlagen der Informatik“ (TGI)

Universität Hamburg

*jantzen@informatik.uni-hamburg.de*



# Die Sprachfamilie $Cf$

- **Theorem:** Folgende Aussagen sind äquivalent:
1.  $L \in Cf$
  2.  $L = L(A)$  für einen beliebigen PDA  $A$
  3.  $L = L(A)$  für einen fast-buchstabierenden PDA  $A$
  4.  $L = L(A)$  für einen buchstabierenden PDA  $A$
  5.  $L = N(A)$  für einen fast-buchstabierenden PDA  $A$



# Präfixfreiheit

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

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

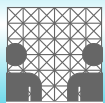


# Präfixfreiheit

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

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

- $\min(L) := \{w \in L \mid (w = uv \wedge u \in L) \rightarrow (v = \lambda)\}$   
bezeichnet den **präfixfreien Anteil** der Sprache  $L$ .



# Präfixfreiheit

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

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

- $\min(L) := \{w \in L \mid (w = uv \wedge u \in L) \rightarrow (v = \lambda)\}$  bezeichnet den **präfixfreien Anteil** der Sprache  $L$ .
- **Beispiel:** Sei  $M := \{a^m b^n \mid m, n \in \mathbb{N} : n \geq m\}$ , dann 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.



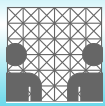
# Eigenschaften von $\det Cf$

- **Theorem:** Eine deterministische kontextfreie Sprache  $L \in \det Cf$  ist präfixfrei gdw. ein DPDA  $A$  existiert, mit  $L = N(A)$  .



# Eigenschaften von $\det Cf$

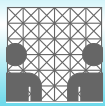
- **Theorem:** Eine deterministische kontextfreie Sprache  $L \in \det Cf$  ist präfixfrei gdw. ein DPDA  $A$  existiert, mit  $L = N(A)$  .
- **Beweisidee:**



# Eigenschaften von $\det Cf$

- **Theorem:** Eine deterministische kontextfreie Sprache  $L \in \det Cf$  ist präfixfrei gdw. ein DPDA  $A$  existiert, mit  $L = N(A)$  .
- **Beweisidee:**
  - $\Rightarrow$  DPDA für  $L$  kann kein Wort akzeptieren, nachdem ein Endzustand verlassen wurde. Hinzufügen eines neuen Zustands, der den Keller leert und genau von den vormaligen Endzuständen erreichbar ist.





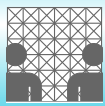
# Eigenschaften von $\det Cf$

- **Theorem:** Eine deterministische kontextfreie Sprache  $L \in \det Cf$  ist präfixfrei gdw. ein DPDA  $A$  existiert, mit  $L = N(A)$  .
- **Beweisidee:**
  - $\Rightarrow$  DPDA für  $L$  kann kein Wort akzeptieren, nachdem ein Endzustand verlassen wurde. Hinzufügen eines neuen Zustands, der den Keller leert und genau von den vormaligen Endzuständen erreichbar ist.
  - $\Leftarrow$  DPDAs können bei leerem Keller keine weiteren Bewegungen vornehmen!



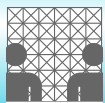
# Eigenschaften von $\det Cf$

- **Korollar** Die präfixfreien deterministisch kontextfreien Sprachen bilden eine echte Teilfamilie der Familie  $\det Cf$ .



# Eigenschaften von $\det Cf$

- **Korollar** Die präfixfreien deterministisch kontextfreien Sprachen bilden eine echte Teilfamilie der Familie  $\det Cf$ .
- **Beweis:** Die Menge  $\{a\}^*\{b\}^* = \{a^n b^m \mid m, n \in \mathbb{N}\}$  ist, da regulär, deterministisch kontextfrei, aber offensichtlich nicht präfixfrei.



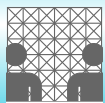
# Abschlusseigenschaften

- **Theorem:** Für  $L \in \det Cf$  ist auch  $\min(L) \in \det Cf$ .



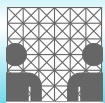
# Abschlusseigenschaften

- **Theorem:** Für  $L \in \det Cf$  ist auch  $\min(L) \in \det Cf$ .
- **Beweisidee:** Kanten aus Endzuständen entfernen.



# Abschlusseigenschaften

- **Theorem:** Für  $L \in \det Cf$  ist auch  $\min(L) \in \det Cf$ .
- **Beweisidee:** Kanten aus Endzuständen entfernen.
- **Theorem:**  $\det Cf \wedge Reg \subseteq \det Cf$ , d.h., die Familie der deterministischen kontextfreien Sprachen ist gegenüber Durchschnittsbildung mit regulären Mengen abgeschlossen.



# Abschlusseigenschaften

- **Theorem:** Für  $L \in \text{det Cf}$  ist auch  $\min(L) \in \text{det Cf}$ .
- **Beweisidee:** Kanten aus Endzuständen entfernen.
- **Theorem:**  $\text{det Cf} \wedge \text{Reg} \subseteq \text{det Cf}$ , d.h., die Familie der deterministischen kontextfreien Sprachen ist gegenüber Durchschnittsbildung mit regulären Mengen abgeschlossen.
- **Beweisidee:** Produktautomat zweier DPDAs



# Eindeutigkeit vs. Determinismus

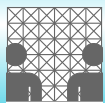
- **Theorem:** Die kontextfreie Sprache *PAL* ist eindeutig, aber nicht deterministisch.





# Eindeutigkeit vs. Determinismus

- **Theorem:** Die kontextfreie Sprache *PAL* ist eindeutig, aber nicht deterministisch.
- **Beweis:**



# Eindeutigkeit vs. Determinismus

- **Theorem:** Die kontextfreie Sprache  $PAL$  ist eindeutig, aber nicht deterministisch.
- **Beweis:**
  - Eindeutigkeit:  $S \longrightarrow aSa \mid bSb \mid a \mid b \mid \lambda$



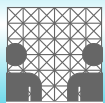
# Eindeutigkeit vs. Determinismus

- **Theorem:** Die kontextfreie Sprache  $PAL$  ist eindeutig, aber nicht deterministisch.
- **Beweis:**
  - Eindeutigkeit:  $S \longrightarrow aSa \mid bSb \mid a \mid b \mid \lambda$
  - Wäre nun  $PAL \in det Cf$ , dann wäre auch  $K := PAL \cap (ab)^+(ba)^+(ab)^+(ba)^+ \in det Cf$ .



# Eindeutigkeit vs. Determinismus

- **Theorem:** Die kontextfreie Sprache  $PAL$  ist eindeutig, aber nicht deterministisch.
- **Beweis:**
  - Eindeutigkeit:  $S \longrightarrow aSa \mid bSb \mid a \mid b \mid \lambda$
  - Wäre nun  $PAL \in det Cf$ , dann wäre auch  $K := PAL \cap (ab)^+(ba)^+(ab)^+(ba)^+ \in det Cf$ .
  - Dann ist auch  $\min(K) \in det Cf$ .



# Eindeutigkeit vs. Determinismus

- **Theorem:** Die kontextfreie Sprache  $PAL$  ist eindeutig, aber nicht deterministisch.

- **Beweis:**

- Eindeutigkeit:  $S \longrightarrow aSa \mid bSb \mid a \mid b \mid \lambda$
- Wäre nun  $PAL \in \det Cf$ , dann wäre auch  $K := PAL \cap (ab)^+(ba)^+(ab)^+(ba)^+ \in \det Cf$ .
- Dann ist auch  $\min(K) \in \det Cf$ .
- Nun ergibt sich:  
$$\min(K) = \{(ab)^m(ba)^n(ab)^n(ba)^m \mid 0 \leq n \leq m\}$$



# Eindeutigkeit vs. Determinismus

- **Theorem:** Die kontextfreie Sprache  $PAL$  ist eindeutig, aber nicht deterministisch.
- **Beweis:**
  - Eindeutigkeit:  $S \longrightarrow aSa \mid bSb \mid a \mid b \mid \lambda$
  - Wäre nun  $PAL \in \det Cf$ , dann wäre auch  $K := PAL \cap (ab)^+(ba)^+(ab)^+(ba)^+ \in \det Cf$ .
  - Dann ist auch  $\min(K) \in \det Cf$ .
  - Nun ergibt sich:
$$\min(K) = \{(ab)^m(ba)^n(ab)^n(ba)^m \mid 0 \leq n \leq m\}$$
  - Diese Sprache ist nicht einmal mehr kontextfrei, wie man mit dem  $uvwxy$ -Theorem zeigen kann.



# Eindeutigkeit vs. Determinismus

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



# Komplementbildung

- **Theorem:**  $\det Cf$  ist gegen Komplementbildung abgeschlossen, d.h.  $\forall L \in \det Cf : \overline{L} \in \det Cf$ .





# Komplementbildung

- **Theorem:**  $\det Cf$  ist gegen Komplementbildung abgeschlossen, d.h.  $\forall L \in \det Cf : \overline{L} \in \det Cf$ .
- **Beweisidee:** Endzustände mit Nicht-Endzuständen zu vertauschen reicht hier nicht aus!



# Komplementbildung

- **Theorem:**  $\det Cf$  ist gegen Komplementbildung abgeschlossen, d.h.  $\forall L \in \det Cf : \overline{L} \in \det Cf$ .
- **Beweisidee:** Endzustände mit Nicht-Endzuständen zu vertauschen reicht hier nicht aus!
- Gründe für das Nicht-Akzeptieren einer Eingabe beim DPDA:



# Komplementbildung

- **Theorem:**  $\det C_f$  ist gegen Komplementbildung abgeschlossen, d.h.  $\forall L \in \det C_f : \overline{L} \in \det C_f$ .
- **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.



# Komplementbildung

- **Theorem:**  $\det Cf$  ist gegen Komplementbildung abgeschlossen, d.h.  $\forall L \in \det Cf : \overline{L} \in \det Cf$ .
- **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.



# Komplementbildung

- **Theorem:**  $\det Cf$  ist gegen Komplementbildung abgeschlossen, d.h.  $\forall L \in \det Cf : \overline{L} \in \det Cf$ .
- **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.



# Komplementbildung

- **Theorem:**  $\det Cf$  ist gegen Komplementbildung abgeschlossen, d.h.  $\forall L \in \det Cf : \overline{L} \in \det Cf$ .
- **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.



# Beweisansatz

**Zu 1:** Neues Kellerbodenzeichen vor das alte setzen.



# Beweisansatz

**Zu 1:** Neues Kellerbodenzeichen vor das alte setzen.

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





# Beweisansatz

- 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:



# Beweisansatz

- 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  $\lambda$ -Schleife wird der Keller nie kürzer, aber ein Endzustand kommt vor. Neuer Endzustand und Schleifenanfangskante dahin „umbiegen“, Keller leeren.



# Beweisansatz

- 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  $\lambda$ -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  $\lambda$ -Schleife ist, in der niemals ein Endzustand vorkommt: Keller leeren.



# Beweisansatz

**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  $\lambda$ -Schleife wird der Keller nie kürzer, aber ein Endzustand kommt vor. Neuer Endzustand und Schleifenanfangskante dahin „umbiegen“, Keller leeren.

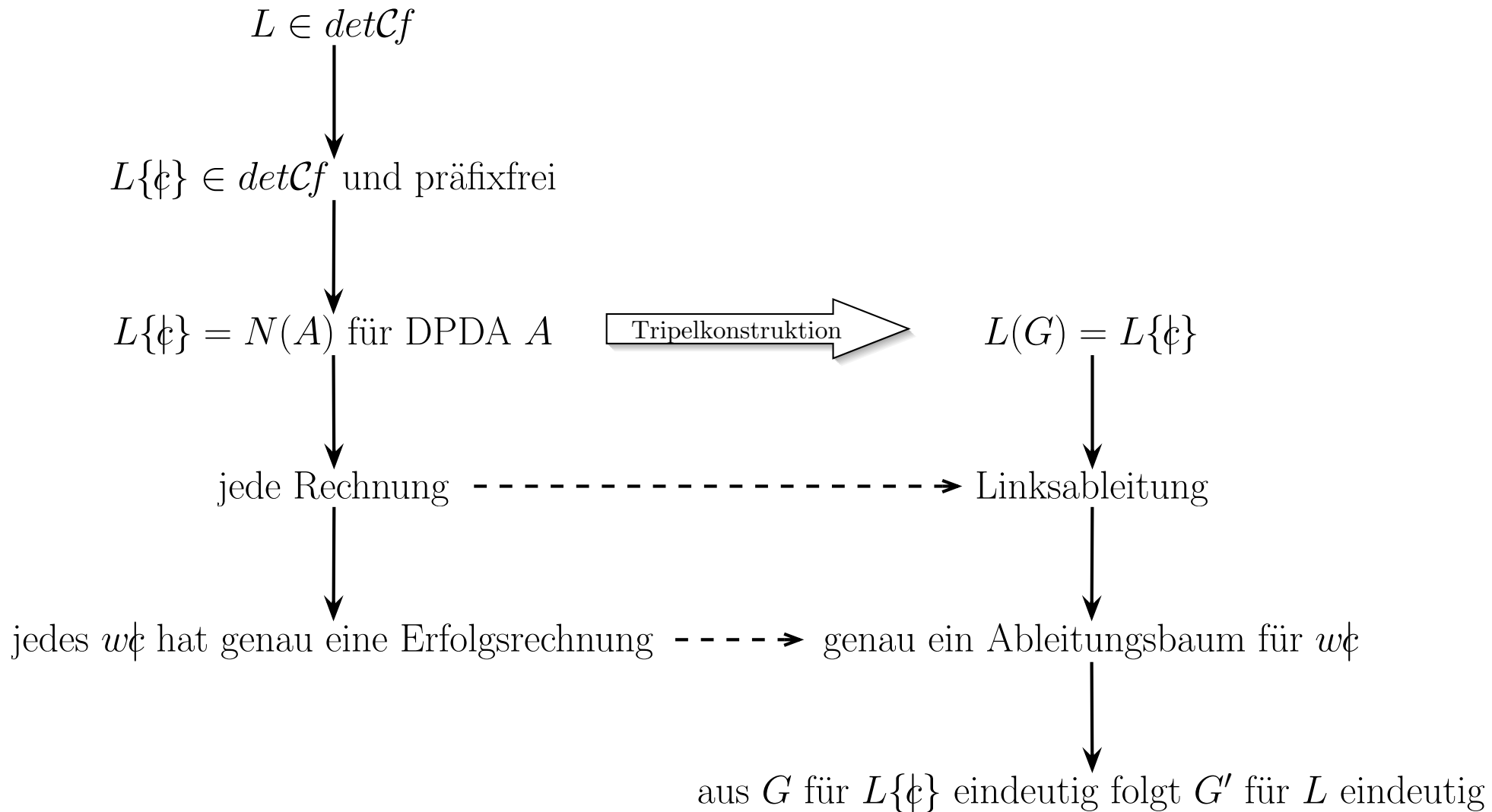
b): Falls die Konfiguration  $xq$  der Beginn einer  $\lambda$ -Schleife ist, in der niemals ein Endzustand vorkommt: Keller leeren.

**Fall 4:** Zu kompliziert ...



# Wichtig für Syntaxanalyse

● **Theorem:** Jede Sprache aus  $detCf$  ist eindeutig.





# Aufbau eines Compilers (1)

- Lexikalische Analyse



# Aufbau eines Compilers (1)

- **Lexikalische Analyse**
  - Das Quellprogramm wird analysiert:



# Aufbau eines Compilers (1)

## ● Lexikalische Analyse

- Das Quellprogramm wird analysiert:
- Symbole einlesen, Leerzeichen unterdrücken, Korrektheit sprachinterner Schlüsselworte prüfen und *Token* ersetzen.





# Aufbau eines Compilers (1)

## ● Lexikalische Analyse

- Das Quellprogramm wird analysiert:
- Symbole einlesen, Leerzeichen unterdrücken, Korrektheit sprachinterner Schlüsselworte prüfen und *Token* ersetzen.
- Pointer zu weiteren Informationen (*Wert*, *Tokentyp* und *ASCII-Wert*) anlegen



# Aufbau eines Compilers (1)

## ● Lexikalische Analyse

- Das Quellprogramm wird analysiert:
- Symbole einlesen, Leerzeichen unterdrücken, Korrektheit sprachinterner Schlüsselworte 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 (1)

## ● Lexikalische Analyse

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

| IF | *i* | <= | *n* | THEN | *a* | ( / | *i* | + | 1 | / ) | := |  $\emptyset$  | ELSE | *i* | := | *i* | + | 1 | ; |

↓  
Aktion, Pointer zu „≤“

↓  
syntaktisch, Pointer zu „]“

↓  
Identifizier, Pointer zur Adresse von „i“



# Aufbau eines Compilers (2)

- Syntaktische Analyse



# Aufbau eines Compilers (2)

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

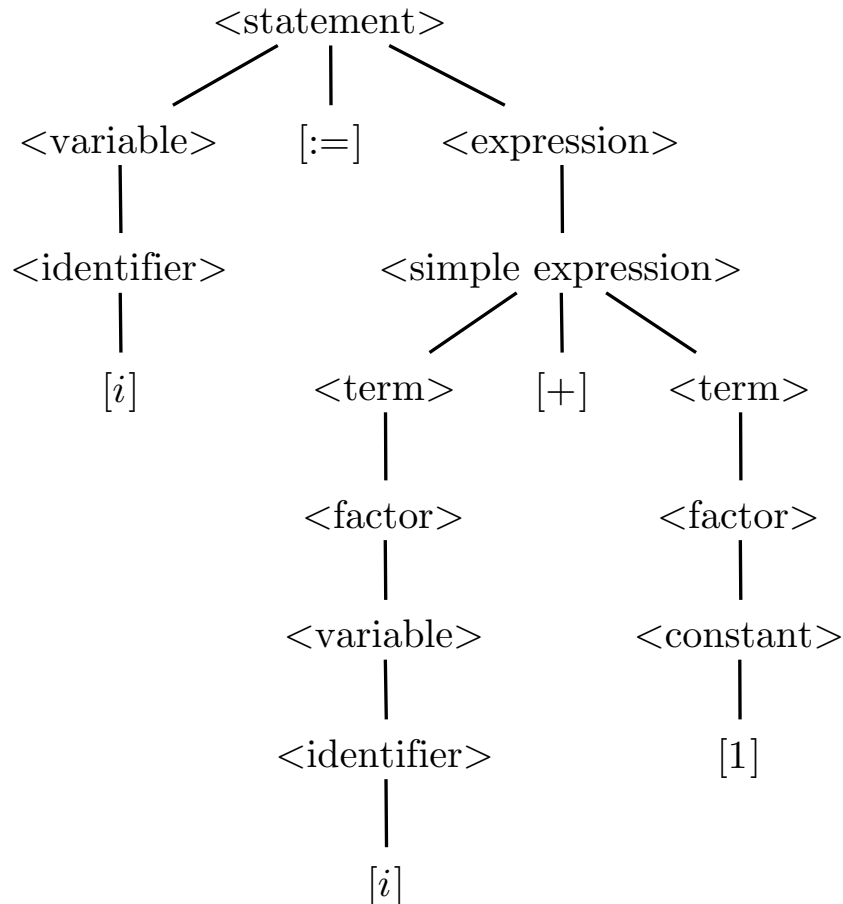


# Aufbau eines Compilers (2)

## ● Syntaktische Analyse

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

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





# Aufbau eines Compilers (3)

- **Semantische Analyse und Codegenerierung**



# Aufbau eines Compilers (3)

- **Semantische Analyse und Codegenerierung**
  - Synthese-Phase: es wird ein **abstrakter Syntaxbaum** aus dem Ableitungsbaum konstruiert





# 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



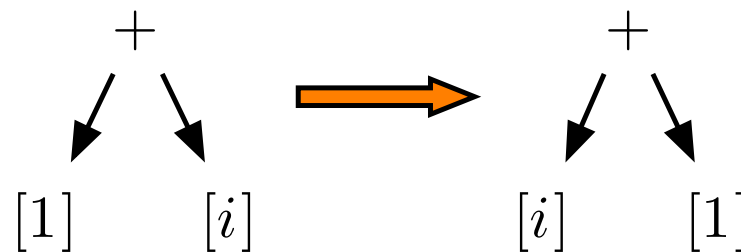
# Aufbau eines Compilers (3)

- **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 (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 (4)

## • Codeoptimierung



# Aufbau eines Compilers (4)

- **Codeoptimierung**

- Beispiel: Summe „+“ mit zwei Operanden.



# Aufbau eines Compilers (4)

## • Codeoptimierung

- Beispiel: Summe „+“ mit zwei Operanden.
- Statt „+1“ einen Befehl „increment“ verwenden.



# 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.



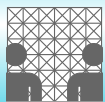
# Eine CFG

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

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

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





# LR(1)-Parsing

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>
$* T + E$	$i$		<i>shift</i>
$i * T + E$			<i>shift</i>



# Eine andere CFG

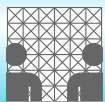
$$E \longrightarrow TE'$$

$$E' \longrightarrow +TE' \mid \lambda$$

$$T \longrightarrow FT'$$

$$T' \longrightarrow *FT' \mid \lambda$$

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



# LL(1)-Parsing

## Präfix der Satzform im Keller

Kellerboden ↓      ↓ Vorausschau

Kellerinhalt		zu lesende Eingabe	
$E$		$i + i * i$	<i>Start</i>
$E$	$i$	$+ i * i$	<i>positioning</i>
$TE'$	$i$	$+ i * i$	<i>expand</i>
$FT'E'$	$i$	$+ i * i$	<i>expand</i>
$iT'E'$	$i$	$+ i * i$	<i>expand</i>
$T'E'$		$+ i * i$	<i>reduce</i>
$T'E'$	$+$	$i * i$	<i>positioning</i>
$E'$	$+$	$i * i$	<i>expand(<math>T' \rightarrow \lambda</math>)</i>
$+TE'$	$+$	$i * i$	<i>expand</i>
$TE'$		$i * i$	<i>reduce</i>
$TE'$	$i$	$* i$	<i>positioning</i>
$FT'E'$	$i$	$* i$	<i>expand</i>



# LR( $k$ )-Grammatiken

LR-Parsing ist aus folgenden Gründen attraktiv:

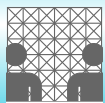
- Jede deterministische kontextfreie Sprache kann mit diesem Bottom-up Verfahren analysiert und übersetzt werden



# LR( $k$ )-Grammatiken

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.



# LR( $k$ )-Grammatiken

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.



# LR( $k$ )-Grammatiken

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.



# LR( $k$ )-Grammatiken

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 ausgebaute Verfahren, und beinahe jede Programmiersprachensyntax besitzt eine LR( $k$ )-Grammatik.





# Definition LR( $k$ )

- Sei  $G := (V_N, V_T, P, S)$  eine **reduzierte CFG**, bei der  $S$  in keiner rechten Seite einer Produktion vorkommt.
- Für alle  $k \in \mathbb{N}$  und jedes  $w \in V^*$  bezeichne  $\text{first}_k(w)$  den Präfix von  $w$  der Länge  $k$ .
- Die CFG  $G$  heißt LR( $k$ ), falls die Bedingungen 1., 2. und 3. die Aussage 4. implizieren:

$$1. \exists \alpha, \beta, \gamma \in V^* \exists A \in V_N : S \xRightarrow[\text{re}]{*} \alpha A \gamma \xRightarrow[\text{re}}{\alpha \beta \gamma}$$

$$2. \exists \alpha', \gamma', \gamma'' \in V^* \exists B \in V_N : S \xRightarrow[\text{re}]{*} \alpha' B \gamma' \xRightarrow[\text{re}}{\alpha \beta \gamma''}$$

$$3. \text{first}_k(\gamma) = \text{first}_k(\gamma'')$$

$$4. A = B, \alpha = \alpha' \text{ und } \gamma' = \gamma''$$



# Eigenschaften

- Jede  $LR(k)$ -Grammatik ist eindeutig.



# Eigenschaften

- Jede  $LR(k)$ -Grammatik ist eindeutig.
- Folgende drei Aussagen sind äquivalent:



# Eigenschaften

- Jede  $LR(k)$ -Grammatik ist eindeutig.
- Folgende drei Aussagen sind äquivalent:
  1.  $L$  ist präfixfreie, deterministisch kontextfreie Sprache



# Eigenschaften

- Jede  $LR(k)$ -Grammatik ist eindeutig.
- Folgende drei Aussagen sind äquivalent:
  1.  $L$  ist präfixfreie, deterministisch kontextfreie Sprache
  2.  $L = L(G)$  für eine  $LR(0)$ -Grammatik



# Eigenschaften

- Jede  $LR(k)$ -Grammatik ist eindeutig.
- Folgende drei Aussagen sind äquivalent:
  1.  $L$  ist präfixfreie, deterministisch kontextfreie Sprache
  2.  $L = L(G)$  für eine  $LR(0)$ -Grammatik
  3.  $L = N(A)$  für einen DPDA



# Eigenschaften

- Jede  $LR(k)$ -Grammatik ist eindeutig.
- Folgende drei Aussagen sind äquivalent:
  1.  $L$  ist präfixfreie, deterministisch kontextfreie Sprache
  2.  $L = L(G)$  für eine  $LR(0)$ -Grammatik
  3.  $L = N(A)$  für einen DPDA
- Folgende drei Aussagen sind äquivalent:



# Eigenschaften

- Jede  $LR(k)$ -Grammatik ist eindeutig.
- Folgende drei Aussagen sind äquivalent:
  1.  $L$  ist präfixfreie, deterministisch kontextfreie Sprache
  2.  $L = L(G)$  für eine  $LR(0)$ -Grammatik
  3.  $L = N(A)$  für einen DPDA
- Folgende drei Aussagen sind äquivalent:
  1.  $L$  ist deterministisch kontextfrei, d.h.  $L = L(A)$  für einen DPDA  $A$ .





# Eigenschaften

- Jede  $LR(k)$ -Grammatik ist eindeutig.
- Folgende drei Aussagen sind äquivalent:
  1.  $L$  ist präfixfreie, deterministisch kontextfreie Sprache
  2.  $L = L(G)$  für eine  $LR(0)$ -Grammatik
  3.  $L = N(A)$  für einen DPDA
- 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  $LR(1)$ -Grammatik  $G$ .



# Eigenschaften

- Jede  $LR(k)$ -Grammatik ist eindeutig.
- Folgende drei Aussagen sind äquivalent:
  1.  $L$  ist präfixfreie, deterministisch kontextfreie Sprache
  2.  $L = L(G)$  für eine  $LR(0)$ -Grammatik
  3.  $L = N(A)$  für einen DPDA
- 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  $LR(1)$ -Grammatik  $G$ .
  3.  $L = L(G)$  für eine  $LR(k)$ -Grammatik  $G$  für ein  $k \geq 1$ .



# Eigenschaften

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



# LL( $k$ )-Grammatiken

- Sei  $G := G = (V_N, V_T, P, S)$  eine kontextfreie Grammatik,  $k \in \mathbb{N}$  und  $w \in (V_N \cup V_T)^*$ .

$$\text{FIRST}_k(w) := \{u \in V_T^* \mid w \xRightarrow[\text{li}]{*} u \text{ und } |u| < k \text{ oder}$$

$$\exists v \in (V_N \cup V_T)^* : w \xRightarrow[\text{li}]{*} uv \text{ mit } |u| = k \}$$



# LL( $k$ )-Grammatiken

- Sei  $G := G = (V_N, V_T, P, S)$  eine kontextfreie Grammatik,  $k \in \mathbb{N}$  und  $w \in (V_N \cup V_T)^*$ .

$$\text{FIRST}_k(w) := \{u \in V_T^* \mid w \xRightarrow[\text{li}]{*} u \text{ und } |u| < k \text{ oder}$$

$$\exists v \in (V_N \cup V_T)^* : w \xRightarrow[\text{li}]{*} uv \text{ mit } |u| = k \}$$

- Eine reduzierte CFG  $G := G = (V_N, V_T, P, S)$  heißt LL( $k$ ) für ein  $k \in \mathbb{N}$ , falls für je zwei Linksableitungen

$$S \xRightarrow[\text{li}]{*} uA\alpha \xRightarrow[\text{li}]{} uv\alpha \xRightarrow[\text{li}]{*} uw \text{ und}$$

$$S \xRightarrow[\text{li}]{*} uA\alpha \xRightarrow[\text{li}]{} uv'\alpha \xRightarrow[\text{li}]{*} uw' \text{ mit } u, w, w' \in V_T^* \text{ und}$$

$\alpha, v, v' \in (V_N \cup V_T)^*$ , aus  $\text{FIRST}_k(w) = \text{FIRST}_k(w')$  stets  $v = v'$  folgt.



# Eigenschaften

- Keine  $LL(k)$ -Grammatik besitzt Linksrekursionen.



# Eigenschaften

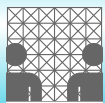
- Keine  $LL(k)$ -Grammatik besitzt Linksrekursionen.
- Jede  $LL(k)$ -Grammatik ist eindeutig und zugleich eine  $LR(k)$ -Grammatik. Folglich ist jede  $LL(k)$ -Sprache eine deterministische, kontextfreie Sprache.



# Eigenschaften

- Keine  $LL(k)$ -Grammatik besitzt Linksrekursionen.
- Jede  $LL(k)$ -Grammatik ist eindeutig und zugleich eine  $LR(k)$ -Grammatik. Folglich ist jede  $LL(k)$ -Sprache eine deterministische, kontextfreie Sprache.
- Zu festem  $k \in \mathbb{N}$  gibt es einen Algorithmus, der zu jeder beliebig vorgegebenen Grammatik  $G$  feststellt, ob dies eine  $LL(k)$ -Grammatik ist.





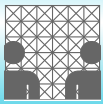
# Eigenschaften

- Keine  $LL(k)$ -Grammatik besitzt Linksrekursionen.
- Jede  $LL(k)$ -Grammatik ist eindeutig und zugleich eine  $LR(k)$ -Grammatik. Folglich ist jede  $LL(k)$ -Sprache eine deterministische, kontextfreie Sprache.
- Zu festem  $k \in \mathbb{N}$  gibt es einen Algorithmus, der zu jeder beliebig vorgegebenen Grammatik  $G$  feststellt, ob dies eine  $LL(k)$ -Grammatik ist.
- Es gibt keinen Algorithmus, der von jeder beliebig vorgelegten CFG feststellt, ob diese eine  $LL(k)$ -Sprache für irgend ein  $k \in \mathbb{N}$  generiert.



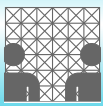
# Weitere Eigenschaften

- Es ist gibt kein Verfahren, das für jede beliebige CFG feststellt, ob diese eine LL(1)-Sprache erzeugt.



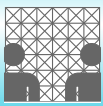
# Weitere Eigenschaften

- Es gibt kein Verfahren, das für jede beliebige CFG feststellt, ob diese eine  $LL(1)$ -Sprache erzeugt.
- Es ist unentscheidbar, ob eine beliebig gegebene  $LR(k)$ -Grammatik zugleich für irgendein  $n \in \mathbb{N}$  eine  $LL(n)$ -Grammatik ist.



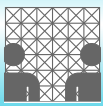
# Weitere Eigenschaften

- Es gibt kein Verfahren, das für jede beliebige CFG feststellt, ob diese eine LL(1)-Sprache erzeugt.
- Es ist unentscheidbar, ob eine beliebig gegebene LR( $k$ )-Grammatik zugleich für irgendein  $n \in \mathbb{N}$  eine LL( $n$ )-Grammatik ist.
- Die LL( $k$ )-Sprachen bilden eine echte Hierarchie: Zu jedem  $k \in \mathbb{N}$  gibt es eine LL( $k$ )-Grammatik  $G_k$ , so dass  $L(G_k) \neq L(G_{k-1})$  für jede LL( $k-1$ )-Grammatik  $G_{k-1}$ .



# Weitere Eigenschaften

- Es gibt kein Verfahren, das für jede beliebige CFG feststellt, ob diese eine LL(1)-Sprache erzeugt.
- Es ist unentscheidbar, ob eine beliebig gegebene LR( $k$ )-Grammatik zugleich für irgendein  $n \in \mathbb{N}$  eine LL( $n$ )-Grammatik ist.
- Die LL( $k$ )-Sprachen bilden eine echte Hierarchie: Zu jedem  $k \in \mathbb{N}$  gibt es eine LL( $k$ )-Grammatik  $G_k$ , so dass  $L(G_k) \neq L(G_{k-1})$  für jede LL( $k-1$ )-Grammatik  $G_{k-1}$ .
- Nicht jede deterministische, kontextfreie Sprache besitzt eine LL( $k$ )-Grammatik: Die Sprache  $\{a^n b^m \mid 1 \leq n \leq m\}$  besitzt eine LR( $k$ )-Grammatik, aber keine LL( $k$ )-Grammatik.



# Weitere Eigenschaften

- Es gibt kein Verfahren, das für jede beliebige CFG feststellt, ob diese eine LL(1)-Sprache erzeugt.
- Es ist unentscheidbar, ob eine beliebig gegebene LR( $k$ )-Grammatik zugleich für irgendein  $n \in \mathbb{N}$  eine LL( $n$ )-Grammatik ist.
- Die LL( $k$ )-Sprachen bilden eine echte Hierarchie: Zu jedem  $k \in \mathbb{N}$  gibt es eine LL( $k$ )-Grammatik  $G_k$ , so dass  $L(G_k) \neq L(G_{k-1})$  für jede LL( $k-1$ )-Grammatik  $G_{k-1}$ .
- Nicht jede deterministische, kontextfreie Sprache besitzt eine LL( $k$ )-Grammatik: Die Sprache  $\{a^n b^m \mid 1 \leq n \leq m\}$  besitzt eine LR( $k$ )-Grammatik, aber keine LL( $k$ )-Grammatik.
- Für zwei LL( $k$ )-Grammatiken  $G$  und  $G'$  ist entscheidbar, ob  $L(G) = L(G')$  gilt.