# Kapitel:
# Kontextfreie Sprachen

# Die Syntax von Programmiersprachen

Wie lässt sich die Syntax einer Programmiersprache definieren,
so dass die nachfolgende Syntaxanalyse effizient durchführbar ist?

Wir definieren die Syntax durch eine Grammatik.

**Beispiele:**

- Wie definiert man arithmetische Ausdrücke?

  Wenn V alle gültigen Variablennamen definiert, dann wird ein arithmetischer Ausdruck $A$ durch

  $$A \rightarrow A + A \mid A - A \mid A * A \mid (A) \mid V$$

  definiert.

- Die Menge der gültigen Variablennamen lässt sich einfach durch eine reguläre Grammatik definieren, arithmetische Ausdrücke aber nicht.

# Kontextfreie Sprachen

Eine Grammatik $G = (\Sigma, V, S, P)$ mit Produktionen der Form

$$X \to u \quad \text{mit } X \in V \text{ und } u \in (V \cup \Sigma)^*$$

heißt kontextfrei.
Eine Sprache $L$ heißt kontextfrei, wenn es eine kontextfreie Grammatik $G$ gibt, die $L$ erzeugt, d.h. wenn

$$L(G) = L.$$

**Beachte:**

- Nur Variablen $X$ dürfen ersetzt werden:

  der Kontext von $X$ spielt keine Rolle.

- Kontextfreie Grammatiken sind mächtig, weil rekursive Definitionen ausgedrückt werden können.

  Die Produktionen $A \to A + A \mid A - A \mid A * A \mid (A) \mid V$ stellen eine rekursive Definition arithmetischer Ausdrücke dar.

# Beispiele kontextfreier Sprachen

- $\{a^n b^n \mid n \in \mathbb{N}\}$ ist kontextfrei:
  - Wir arbeiten nur mit dem Startsymbol $S$ und
  - den Produktionen $S \to aSb \mid \epsilon$.
  - Die erste Anwendung von $S \to aSb$ erzeugt das linkeste $a$ und das rechteste $b$.

- Die Dyck-Sprache $D_k$ aller Menge wohlgeformten Ausdrücke mit $k$ Klammertypen $(_1, )_1, \ldots, (_k, )_k$ ist kontextfrei:
  - Auch diesmal arbeiten wir nur mit dem Startsymbol $S$.
  - Die Produktionen: $S \to (_1\, S\, )_1 \mid (_2\, S\, )_2 \mid \cdots \mid (_k\, S\, )_k \mid SS \mid \epsilon$.
  - Was modellieren Dyck-Sprachen?
    - ⋆ Die Klammerstruktur in arithmetischen oder Booleschen Ausdrücken,
    - ⋆ die „Klammerstruktur“ von Codeblöcken (z.B. geschweifte Klammern in C oder `begin` und `end` in Pascal),
    - ⋆ die Syntax von HTML oder XML.

# Eine KFG für $L := \{w \in \{0,1\}^+ \ : \ |w|_0 = |w|_1\}$

Wir arbeiten mit den drei Variablen S, Null, Eins und

den Produktionen

$$\begin{array}{rcl} S & \to & 0\,\text{Eins} \mid 1\,\text{Null} \\ \text{Eins} & \to & 1 \mid 1\,S \mid 0\,\text{Eins}\,\text{Eins} \\ \text{Null} & \to & 0 \mid 0\,S \mid 1\,\text{Null}\,\text{Null} \end{array}$$

**Idee:**

- Ein Wort $w \in \{0,1\}^*$ ist genau dann aus Eins ableitbar, wenn $w$ genau eine Eins mehr als Nullen hat.
- $w$ ist genau dann aus Null ableitbar, wenn $w$ genau eine Null mehr als Einsen hat.
- $w$ ist genau dann $S$ ableitbar, wenn $w$ genau so viele Nullen wie Einsen hat.

**Behauptung:** $L(G) = L$.

**Beweis:** siehe Tafel!

# Ein Fragment von Pascal

Wir beschreiben einen (allerdings sehr kleinen) Ausschnitt von Pascal durch eine kontextfreie Grammatik.

- Wir benutzen das Alphabet $\Sigma = \{\texttt{a}, \ldots, \texttt{z}, \texttt{;}, :=, \texttt{begin}, \texttt{end}, \texttt{while}, \texttt{do}\}$ und
- die Variablen *S*, statements, statement, assign-statement, while-statement, variable, boolean, expression.
- variable, boolean und expression sind im Folgenden nicht weiter ausgeführt.

$$
\begin{aligned}
S &\to \texttt{begin}\ \text{statements}\ \texttt{end} \\
\text{statements} &\to \text{statement} \mid \text{statement}\ \texttt{;}\ \text{statements} \\
\text{statement} &\to \text{assign-statement} \mid \text{while-statement} \\
\text{assign-statement} &\to \text{variable} := \text{expression} \\
\text{while-statement} &\to \texttt{while}\ \text{boolean}\ \texttt{do}\ \text{statements}
\end{aligned}
$$

# Programmiersprachen und kontextfreie Sprachen

Lassen sich die syntaktisch korrekten Programme einer modernen Programmiersprache durch eine kontextfreie Sprache definieren?

- 1. Antwort: Nein. In Pascal muss zum Beispiel sichergestellt werden, dass Anzahl und Typen der formalen und aktuellen Parameter übereinstimmen.
  - Die Sprache $\{ww \mid w \in \Sigma^*\}$ wird sich als nicht kontextfrei herausstellen.
- 2. Antwort: Im Wesentlichen ja, wenn man „Details“ wie Typ-Deklarationen und Typ-Überprüfungen ausklammert:
  - Man beschreibt die Syntax durch eine kontextfreie Grammatik, die alle syntaktisch korrekten Programme erzeugt.
  - Allerdings werden auch syntaktisch inkorrekte Progamme (z.B. aufgrund von Typ-Inkonsistenzen) erzeugt.
  - Die nicht-kontextfreien Syntax-Vorschriften können nach Erstellung des Ableitungsbaums überprüft werden.

# Die Backus-Naur Normalform

Die Backus-Naur Normalform (BNF) wird zur Formalisierung der Syntax von Programmiersprachen genutzt.

- Sie ist ein “Dialekt” der Kontextfreien Grammatiken.

  Produktionen der Form

  $$X \to aYb$$

  (mit $X, Y \in V$ und $a, b \in \Sigma$) werden in BNF notiert als

  ```
  <X> ::= a <Y> b
  ```

- **Beispiel:** Eine Beschreibung der Syntax von Java in einer Variante der BNF auf `http://docs.oracle.com/javase/specs/jls/se7/html/index.html`

  (zuletzt besucht am 23.05.2012)

Eine effiziente Syntaxanalyse ist möglich.

**Frage:** Was ist eine Syntaxanalyse?

**Antwort:** Die Bestimmung einer Ableitung bzw. eines Ableitungsbaums.

Und was ist ein Ableitungsbaum?

# Ableitungsbäume und eindeutige Grammatiken

# Ableitungsbäume

$G = (\Sigma, V, S, P)$ sei eine kontextfreie Grammatik.

Ein Baum $B$ heißt Ableitungsbaum für das Wort $w$, falls Folgendes gilt:

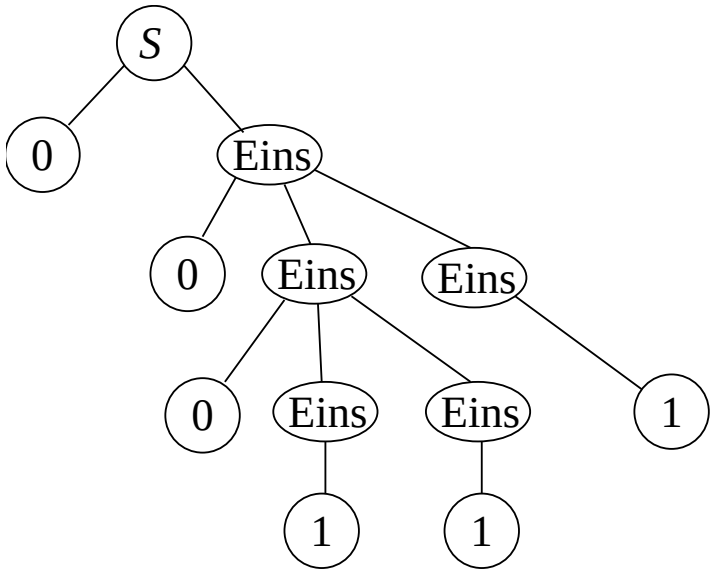
- $w$ ist die links-nach-rechts Konkatenation der Blätter von $B$.
- Die Wurzel von $B$ ist mit $S$ markiert.
- Innere Knoten sind mit Variablen markiert.
  Blätter sind mit Buchstaben aus $\Sigma$ oder dem Symbol $\varepsilon$ markiert.
- Für jeden Knoten $v$ gilt:
  - Wenn $v$ mit dem Symbol $\varepsilon$ markiert ist, so ist $v$ das einzige Kind seines Elternknotens $w$, und $w$ ist mit einer Variablen $A$ markiert, so dass $A \to \varepsilon$ eine Produktion von $G$ ist.
  - Wenn $v$ mit einer Variablen $A$ markiert ist und die Kinder von $v$ die Markierungen (von links nach rechts) $v_1, \ldots, v_s \in \Sigma \cup V$ tragen, dann ist $A \to v_1 \cdots v_s$ eine Produktion von $G$.

# Ein Ableitungsbaum für $w = 000111$

Die Produktionen

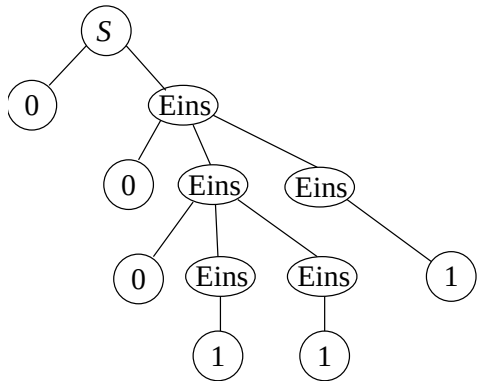
$$
\begin{aligned}
S &\to 0\,\text{Eins} \mid 1\,\text{Null} \\
\text{Eins} &\to 1 \mid 1\,S \mid 0\,\text{Eins}\,\text{Eins} \\
\text{Null} &\to 0 \mid 0\,S \mid 1\,\text{Null}\,\text{Null}
\end{aligned}
$$

$$
\begin{aligned}
S &\Rightarrow 0\,\text{Eins} \Rightarrow 00\,\text{Eins}\,\text{Eins} \Rightarrow 000\,\text{Eins}\,\text{Eins}\,\text{Eins} \\
&\Rightarrow 0001\,\text{Eins}\,\text{Eins} \Rightarrow 00011\,\text{Eins} \Rightarrow 000111 \quad \text{mit dem Baum}
\end{aligned}
$$



# Links- und Rechtsableitungen

Ein Ableitungsbaum „produziert“ Linksableitungen (ersetze jeweils die linkeste Variable) und Rechtsableitungen (ersetze rechteste Variable).




- hat die Linksableitung

$$\begin{aligned} S &\Rightarrow 0\,\text{Eins} \Rightarrow 00\,\text{Eins}\,\text{Eins} \Rightarrow 000\,\text{Eins}\,\text{Eins}\,\text{Eins} \\ &\Rightarrow 0001\,\text{Eins}\,\text{Eins} \Rightarrow 00011\,\text{Eins} \Rightarrow 000111 \end{aligned}$$

- und die Rechtsableitung

$$\begin{aligned} S &\Rightarrow 0\,\text{Eins} \Rightarrow 00\,\text{Eins}\,\text{Eins} \Rightarrow 00\,\text{Eins}\,1 \\ &\Rightarrow 000\,\text{Eins}\,\text{Eins}\,1 \Rightarrow 000\,\text{Eins}\,11 \Rightarrow 000111. \end{aligned}$$

# Eindeutigkeit und Mehrdeutigkeit

Angenommen, ein Programm hat unterschiedliche Ableitungsbäume $B_1, B_2$. Dann ist die Semantik des Programms unklar, wenn $B_1$ und $B_2$ unterschiedliche Bedeutung besitzen!

- Wir nennen eine Grammatik mehrdeutig, wenn ein Wort zwei oder mehrere Ableitungsbäume besitzt.
- Eine Grammatik ist eindeutig, wenn jedes Wort höchstens einen Ableitungsbaum besitzt.
- Ein Sprache $L$ ist eindeutig, wenn $L = L(G)$ für eine eindeutige kontextfreie Grammatik $G$ gilt. Ansonsten heißt $L$ inhärent mehrdeutig.

**Beispiel (hier ohne Beweis)**

Die folgende kontextfreie Sprache ist inhärent mehrdeutig:

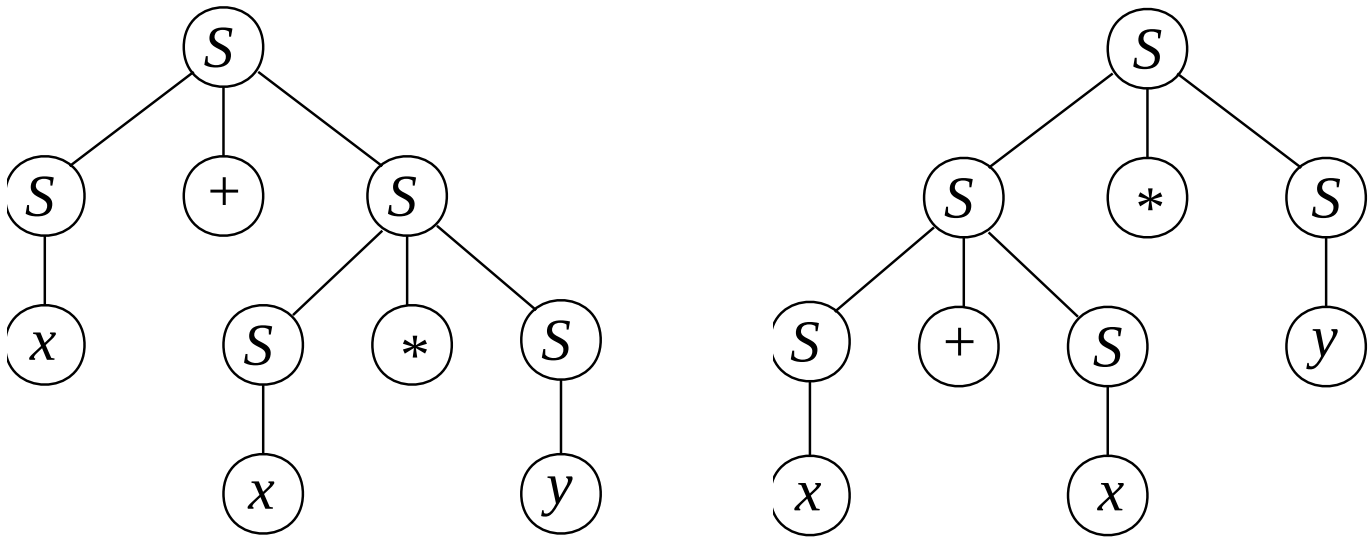
$$\{ a^j b^k c^\ell \ : \ j, k, \ell \in \mathbb{N} \ \text{ mit } \ j = k \ \text{ oder } \ k = \ell \}$$

(Ein Beweis findet sich in Kapitel 6.7 des Buchs von Ingo Wegener.)

# Beispiel: Arithmetische Ausdrücke (1/4)

Die Produktionen $S \rightarrow S + S \mid S * S \mid (S) \mid x \mid y$
definieren arithmetische Ausdrücke auf mehrdeutige Art und Weise.

**Denn:** Das Wort $x + x * y$ hat die beiden Ableitungsbäume:




Der erste Baum führt zur Auswertung $x + (x * y)$, der zweite zu $(x + x) * y$.

Wir brauchen eine eindeutige Grammatik!

# Beispiel: Arithmetische Ausdrücke (2/4)
## Eine eindeutige Grammatik

Wir definieren eine neue Grammatik $G$, die festlegt, dass die Multiplikation stärker "bindet" als die Addition.

- Wir arbeiten mit dem Startsymbol $S$ und den Variablen $T$ (generiert Terme) und $F$ (generiert Faktoren).
- Die Produktionen von $G$ haben die Form

$$\begin{aligned} S &\to S + T \mid T \\ T &\to T * F \mid F \\ F &\to (S) \mid x \mid y. \end{aligned}$$

**Frage:** Warum ist diese Grammatik eindeutig?

# Beispiel: Arithmetische Ausdrücke (3/4)
## Nachweis der Eindeutigkeit: Induktionsanfang

Die Produktionen von $G$:

$$\begin{aligned} S &\to S + T \mid T \\ T &\to T * F \mid F \\ F &\to (S) \mid x \mid y. \end{aligned}$$

Wir führen einen Beweis über die Länge eines arithmetischen Ausdrucks $A$. D.h. wir zeigen für jedes $n \geqslant 1$ und jeden arithmetischen Ausdruck $A$ der Länge $n$, dass $A$ höchstens einen Ableitungsbaum besitzt.

**Induktionsanfang:** $n = 1$

- Die Länge von $A$ ist 1.
- Dann ist $A = x$ oder $A = y$.
- Nur die Ableitung $S \Rightarrow T \Rightarrow F \Rightarrow x$ bzw. $S \Rightarrow T \Rightarrow F \Rightarrow y$ ist möglich. Insbesondere hat $A$ genau einen Ableitungsbaum.

# Beispiel: Arithmetische Ausdrücke (4/4)
## Nachweis der Eindeutigkeit: Induktionsschritt

Die Produktionen von $G$:

$$\begin{array}{lcl} S & \to & S + T \mid T \\ T & \to & T * F \mid F \\ F & \to & (S) \mid x \mid y. \end{array}$$

**Induktionsschritt:** $n \to n+1$

Wir müssen die folgenden 3 Fälle betrachten: $A = A_1 + A_2$, $A = A_1 * A_2$, $A = (A_1)$, wobei $A_1$ und $A_2$ arithmetische Ausdrücke sind.

**Fall 1:** $A = A_1 + A_2$. Dann gilt:

Der Buchstabe $+$ kann nur durch Anwenden der Produktion $S \to S + T$ eingeführt werden.

Also muss jeder Ableitungsbaum $B$ von $A$ mit der Produktion $S \to S + T$ beginnen **(Warum?)**, und die mit $S$ und $T$ beschrifteten Kinder der Wurzel von $B$ erzeugen arithmetische Ausdrücke $A_1'$ und $A_2'$, so dass $A = A_1' + A_2'$.

Wegen $|A_1'|, |A_2'| < |A|$ sind gemäß Induktionsannahme ihre Ableitungsbäume eindeutig. Also besitzt auch $A$ höchstens einen Ableitungsbaum.

**Fall 2:** $A = A_1 * A_2$ **bzw. Fall 3:** $A = (A_1)$**:** analog (Übung!). □

# Die Syntax von if-then-else

Die Syntax für geschachtelte If-Then-Else Anweisungen bei optionaler Else-Verzweigung:

$$S \rightarrow \text{anweisung} \mid \texttt{if}\ \text{bedingung}\ \texttt{then}\ S \mid \texttt{if}\ \text{bedingung}\ \texttt{then}\ S\ \texttt{else}\ S$$

- S ist eine Variable, anweisung und bedingung sind Buchstaben.
- Und wie, bitte schön, ist

  `if` bedingung `then if` bedingung `then` anweisung `else` anweisung

  zu verstehen? Worauf bezieht sich das letzte `else`?
- Diese Grammatik ist mehrdeutig.

**Übungsaufgabe:**
Finde eine neue, eindeutige Grammatik, die dieselbe Sprache erzeugt!

# Das Pumping Lemma für kontextfreie Sprachen

# Das Pumping Lemma für kontextfreie Sprachen

## Das Pumping Lemma für Kontextfreie Sprachen

Sei $L$ eine kontextfreie Sprache.

- Dann gibt es eine **Pumpingkonstante** $n \geqslant 1$, so dass
- jedes Wort $z \in L$ der Länge $|z| \geqslant n$
- eine Zerlegung mit den folgenden Eigenschaften besitzt:
  - $z = uvwxy$, $|vwx| \leqslant n$, $|vx| \geqslant 1$ und
  - $uv^i wx^i y \in L$ für jedes $i \geqslant 0$.

**Beweis:** Später, als direkte Folgerung aus Ogden's Lemma. □

**Folgerung:**

- Wenn es für **jede** Pumpingkonstante $n$ **irgendein** $z \in L$ der Länge $\geqslant n$ gibt, so dass Ab- oder Aufpumpen ($i = 0$ oder $i \geqslant 2$, für irgendein $i$) aus der Sprache hinausführt (d.h. $uv^i wx^i y \notin L$),
  dann ist die Sprache **NICHT** kontextfrei.
- Allerdings: Weder Pumpingkonstante noch Zerlegung sind bekannt.

# Anwendung des Pumping Lemmas: Die Spielregeln

Wie kann man das Pumping Lemma nutzen, um zu zeigen, dass eine Sprache $L$ nicht kontextfrei ist?

(1) Der **"Gegner"** wählt eine Pumpingkonstante $n \geqslant 1$.

(2) Wir wählen ein Wort $z \in L$ mit $|z| \geqslant n$.

(3) Der Gegner zerlegt $z$ in $z = uvwxy$, so dass gilt:
$|vwx| \leqslant n$ und $|vx| \geqslant 1$.

(4) Wir pumpen ab oder auf, d.h. wählen $i = 0$ oder $i \geqslant 2$.

(5) Wir haben gewonnen, wenn $uv^i wx^i y \notin L$; ansonsten hat der Gegner gewonnen.

Aus dem Pumping Lemma folgt:
Wenn wir eine Gewinnstrategie in diesem Spiel haben, dann ist die Sprache $L$ nicht kontextfrei.

**Beachte:** Der "Gegner" kontrolliert die **Pumpingkonstante** $n$ vollständig und die **Zerlegung** $z = uvwxy$ teilweise, denn er muss $|vwx| \leqslant n$ und $|vx| \geqslant 1$ garantieren.

# Das Pumping Lemma: Ein Anwendungsbeispiel (1/2)

## Beispiel

Die Sprache $L := \{a^m b^m c^m : m \geqslant 0\}$ ist nicht kontextfrei.

**Beweisidee:**

(1) Der Gegner wählt die uns vorher nicht bekannte Pumpingkonstante $n$.

(2) Wir wählen $z := a^n b^n c^n$.

(3) Der Gegner zerlegt $z$ in $z = uvwxy$, so dass gilt:
$|vwx| \leqslant n$ und $|vx| \geqslant 1$.

(4) Wir sehen: Wegen $|vwx| \leqslant n$ kann $vwx$ nicht alle drei Buchstaben $a, b, c$ enthalten. Wegen $|vx| \geqslant 1$ ist $vx$ ein nicht-leeres Wort, das höchstens 2 verschiedene Buchstaben enthält. Wenn wir mit $i := 2$ aufpumpen, kann das Wort $uv^2wx^2y$ nicht von allen drei Buchstaben $a, b, c$ gleich viele enthalten.

(5) Also ist $uv^2wx^2y \notin L$. Wir haben also gewonnen.

# Das Pumping Lemma: Ein Anwendungsbeispiel (2/2)

## Beispiel

Die Sprache $L := \{a^m b^m c^m : m \geqslant 0\}$ ist nicht kontextfrei.

**Beweis:** Angenommen, $L$ ist doch kontextfrei.
Gemäß Pumping Lemma gibt es dann eine Pumpingkonstante $n \geqslant 1$, so dass jedes Wort $z \in L$ mit $|z| \geqslant n$ eine Zerlegung in $z = uvwxy$ besitzt, so dass gilt:

$$(*): \qquad |vwx| \leqslant n, \quad |vx| \geqslant 1 \quad \text{und} \quad uv^i wx^i y \in L \text{ für alle } i \geqslant 0.$$

Betrachte insbesondere das Wort $z := a^n b^n c^n$. Es gilt: $z \in L$ und $|z| \geqslant n$.
Gemäß Pumping Lemma gibt es also eine Zerlegung $z = uvwxy$, so dass $(*)$ gilt.
Wegen $uvwxy = a^n b^n c^n$ und $|vwx| \leqslant n$, kann $vwx$ nicht alle drei Buchstaben $a, b, c$ enthalten.
Wegen $|vx| \geqslant 1$, ist $vx$ also ein nicht-leeres Wort, das höchstens zwei verschiedene Buchstaben enthält.
Aber dann kann das Wort $uv^2 wx^2 y$ nicht von allen drei Buchstaben $a, b, c$ gleich viele enthalten. Somit ist $uv^2 wx^2 y \notin L$. WIDERSPRUCH! □

# Das Pumping Lemma: Noch ein Anwendungsbeispiel

## Beispiel

Die Sprache $L := \{w\#w : w \in \{a, b\}^*\}$ ist nicht kontextfrei.

**Beweisidee:**

(1) Der Gegner wählt die uns vorher nicht bekannte Pumpingkonstante $n$.

(2) Wir wählen $z := a^n b^n \# a^n b^n$.

(3) Der Gegner gibt uns eine Zerlegung $z = uvwxy$ mit $|vwx| \leqslant n$ und $|vx| \geqslant 1$.

(4) Wir pumpen mit $i := 2$ auf.

**Behauptung:** $uv^2wx^2y \notin L$

Details: siehe Tafel

# Grenzen des Pumping Lemmas

Es gibt Sprachen, die nicht kontextfrei sind, deren Nicht-Kontextfreiheit mit dem Pumping Lemma aber nicht nachgewiesen werden kann.

**Beispiel:** Die Sprache $L := \{a^j b^k c^\ell d^m : j = 0 \text{ oder } k = \ell = m\}$

ist nicht kontextfrei, erfüllt aber die Aussage des Pumping Lemmas. D.h.:

Es gibt eine Pumpingkonstante $n \geqslant 1$ (nämlich $n = 1$), so dass

- jedes Wort $z \in L$ mit $|z| \geqslant n$
- eine Zerlegung mit den folgenden Eigenschaften besitzt:
  - $z = uvwxy$, $|vwx| \leqslant n$, $|vx| \geqslant 1$, und
  - $uv^i wx^i y \in L$ für jedes $i \geqslant 0$.

**Beweis:**

Die Nicht-Kontextfreiheit von $L$ werden wir mit Ogden's Lemma beweisen (später!).

Rest: Siehe Tafel!

# Ogden's Lemma

## Ogden's Lemma

Sei $L$ eine kontextfreie Sprache.

- Dann gibt es eine Pumpingkonstante $n \geqslant 1$, so dass
- jedes Wort $z \in L$ und jede Markierung von mindestens $n$ Positionen in $z$
- eine Zerlegung mit folgenden Eigenschaften besitzt:
  - $z = uvwxy$, $vwx$ enthält höchstens $n$ markierte Positionen, $vx$ enthält mindestens eine markierte Position und
  - $uv^i wx^i y \in L$ für jedes $i \geqslant 0$.

**Beachte:** Im Spiel gegen den Gegner hat unsere Kraft zugenommen:

- Wir wählen $z \in L$ **und** markieren $\geqslant n$ Positionen.
- Der Gegner kann immer noch die Zerlegung $z = uvwxy$ wählen, muss aber gewährleisten, dass
  - $vwx$ höchstens $n$ markierte Positionen und
  - $vx$ **mindestens eine markierte Position** besitzt!

Das **Pumping Lemma folgt direkt aus Ogden's Lemma**, indem wir **alle** Buchstaben von $z$ markieren.

# Ogden's Lemma: Ein Anwendungsbeispiel (1/2)

## Beispiel

Die Sprache $L := \{a^j b^k c^\ell d^m : j = 0 \text{ oder } k = \ell = m\}$ ist nicht kontextfrei.

**Beweisidee:**

(1) Der Gegner wählt die uns vorher nicht bekannte Pumpingkonstante $n$.

(2) Wir wählen $z := a\,b^n c^n d^n$ und markieren $b^n c^n d^n$.

(3) Für die vom Gegner gewählte Zerlegung $z = uvwxy$ gilt dann:

$vwx$ besitzt kein $b$ oder kein $d$,
denn $vwx$ besitzt nur $\leqslant n$ markierte Positionen.

Somit ist $vx$ ein Wort, das kein $b$ oder kein $d$, aber mindestens einen der Buchstaben $b, c, d$ enthält (da $vx$ mindestens eine markierte Position enthält).

(4) Wir pumpen mit $i := 2$ auf. Das dadurch entstehende Wort $uv^2wx^2y$ besitzt zu wenige $b$'s oder zu wenige $d$'s. Also ist $uv^2wx^2y \notin L$.

# Ogden's Lemma: Ein Anwendungsbeispiel (2/2)

## Beispiel

Die Sprache $L := \{a^j b^k c^\ell d^m : j = 0 \text{ oder } k = \ell = m\}$ ist nicht kontextfrei.

**Beweis:** Angenommen, $L$ ist doch kontextfrei.
Gemäß Ogden's Lemma gibt es dann eine Pumpingkonstante $n \geqslant 1$, so dass jedes Wort $z \in L$ und jede Markierung von mindestens $n$ Positionen in $z$ eine Zerlegung in $z = uvwxy$ besitzt, so dass gilt:

$(*)$: $vwx$ besitzt höchstens $n$ markierte Positionen,
$vx$ besitzt mindestens eine markierte Position und
$uv^i wx^i y \in L$ für alle $i \geqslant 0$.

Betrachte insbesondere das Wort $z := a\,b^n c^n d^n$ und markiere die Positionen $b^n c^n d^n$.
Es gilt: $z \in L$, und es sind mindestens $n$ Positionen von $z$ markiert.
Gemäß Ogden's Lemma gibt es also eine Zerlegung $z = uvwxy$, so dass $(*)$ gilt.
Wegen $z = uvwxy = a\,b^n c^n d^n$ besitzt $vwx$ kein $b$ oder kein $d$, da $vwx$ nur höchstens $n$ markierte Positionen enthält und wir die Positionen $b^n c^n d^n$ markiert haben.
Somit ist $vx$ ein Wort, das kein $b$ oder kein $d$, aber mindestens einen der Buchstaben $b, c, d$ enthält (da $vx$ mindestens eine markierte Position enthält).
Daher enthält das mit $i = 2$ aufgepumpte Wort $uv^2 wx^2 y$ zu wenige $b$'s oder zu wenige $d$'s und liegt somit nicht in $L$. WIDERSPRUCH! □

## Definition (Chomsky Normalform)

Eine Grammatik $G = (\Sigma, V, S, P)$ ist in Chomsky-Normalform, wenn alle Produktionen die folgende Form haben:

$$A \to BC \quad \text{oder} \quad A \to a, \qquad \text{mit } A, B, C \in V \text{ und } a \in \Sigma.$$

## Satz (hier ohne Beweis):

Jede kontextfreie Sprache $L$ mit $\varepsilon \notin L$ wird durch eine Grammatik in Chomsky Normalform erzeugt.

Es gibt einen Algorithmus, der bei Eingabe einer kontextfreien Grammatik $G$ in quadratischer Zeit eine Grammatik $G'$ in Chomsky Normalform erzeugt mit $L(G') = L(G) \setminus \{\varepsilon\}$.

**Beispiel:** Finde eine Grammatik in Chomsky Normalform für die Sprache

$$L = \{a^n b^n \ : \ n \in \mathbb{N}_{>0}\}.$$

**Antwort:** $S \to AR \mid AB, \quad A \to a, \quad B \to b, \quad R \to SB.$

# Ogden's Lemma: Der Beweis (1/5)

Wir betrachten zuerst die „Rahmenbedingungen“:

- Laut dem auf der vorhergehenden Folie zitierten Satz gibt es zur kontextfreien Sprache $L$ eine Grammatik $G = (\Sigma, V, S, P)$ in Chomsky Normalform mit $L(G) = L \setminus \{\varepsilon\}$.
- Wir wählen $n := 2^{|V|+1}$ als Pumpingkonstante (klar: $n \geqslant 1$).
- Sei $z \in L$ ein beliebiges Wort in $L$ mit mindestens $n$ markierten Positionen.
- Wegen $|z| \geqslant n \geqslant 1$ ist $z \in L \setminus \{\varepsilon\} = L(G)$.
  Somit gibt es einen Ableitungsbaum $B$ für $z$ bzgl. $G$.
- Um eine Zerlegung $z = uvwxy$ mit den gewünschten Eigenschaften zu finden, betrachten wir $B$ genauer.

# Ogden's Lemma: Der Beweis (2/5)

**Eigenschaften des Ableitungsbaums $B$ für $z$:**

- $B$ ist binär, da $G$ in Chomsky-Normalform ist.
  Die Elternknoten von Blättern sind die einzigen Knoten vom Grad 1;
  alle anderen Knoten haben Grad 2.
- $B$ hat mindestens $n = 2^{|V|+1}$ markierte Blätter, also Blätter, die einen markierten Buchstaben speichern.

**Notation:**

Ein Knoten $v$ von $B$ heißt Verzweigungsknoten, wenn $v$ den Grad 2 hat und markierte Blätter sowohl im linken als auch im rechten Teilbaum besitzt.

**Ein besonderer Weg in $B$:**

Sei $W$ der in der Wurzel beginnende Weg zu einem Blatt, der in jedem Schritt das Kind mit den meisten markierten Blättern wählt.

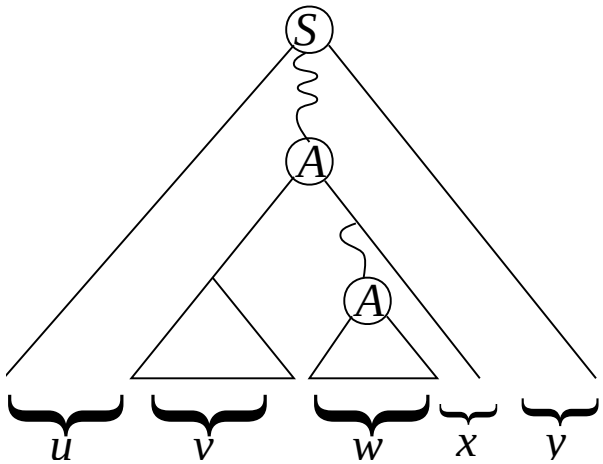
**Es gilt:** $W$ besitzt mindestens $|V| + 1$ Verzweigungsknoten.

**Denn:** Unter der Wurzel sind $\geqslant n$ markierte Blätter. Beim Durchlaufen eines Verzweigungsknotens wird die Anzahl der sich unterhalb des aktuellen Knoten befindenden markierten Blätter höchstens halbiert; beim Durchlaufen eines anderen Knotens bleibt die Anzahl unverändert. Das Blatt am Ende des Weges besitzt höchstens einen markierten Knoten.

# Ogden's Lemma: Der Beweis (3/5)

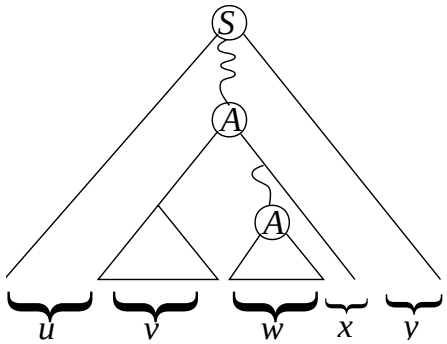
**Wir wissen:** $W$ ist ein Weg in $B$ von der Wurzel zu einem Blatt, der mindestens $|V| + 1$ Verzweigungsknoten besitzt. Daher gilt:

Unter den letzten $|V| + 1$ Verzweigungsknoten des Weges $W$ muss eine Variable $A$ zweimal vorkommen!




- Das unterste Vorkommen $v_2$ von $A$ nutzen wir zur Definition von $w$:
  $w$ ist die Konkatenation der Blätter unterhalb von Knoten $v_2$
- Das darüber liegende Vorkommen $v_1$ von $A$ nutzen wir zur Definition von $u, v, x, y$:
  siehe Skizze.

# Ogden's Lemma: Der Beweis (4/5)



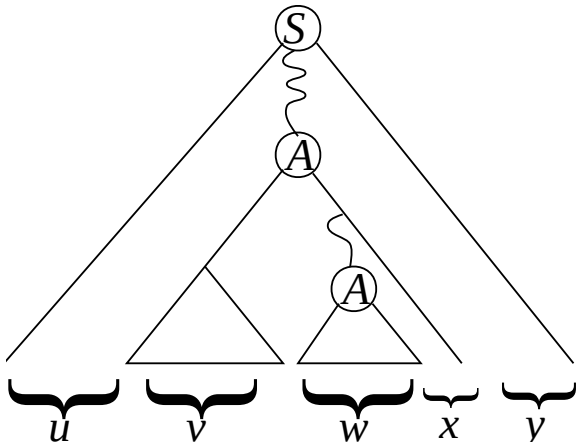

- Da $v_1$ ein Verzweigungsknoten ist, besitzt $vx$ mindestens einen markierten Buchstaben.
- $vwx$ besitzt höchstens $2^{|V|+1} = n$ markierte Buchstaben, denn

  Wenn $vwx$ mehr als $n$ markierte Buchstaben besitzt, dann besitzt $W$ (auf seinem Endstück) mehr als $|V| + 1$ Verzweigungsknoten. **(Warum?)**

Wir haben somit eine Zerlegung $z = uvwxy$ gefunden, so dass $vwx$ höchstens $n$ markierte Positionen und $vx$ mindestens eine markierte Position enthält.

**Frage:** Können wir, wie gewünscht, auf- und abpumpen?

# Ogden's Lemma: Der Beweis (5/5)




Wir erhalten

$$S \overset{*}{\Rightarrow} uAy \quad \text{sowie}$$
$$A \overset{*}{\Rightarrow} vAx \mid w$$

und damit auch die zusätzlichen Ableitungen

$$S \overset{*}{\Rightarrow} uAy \overset{*}{\Rightarrow} uwy = uv^0wx^0y$$
$$S \overset{*}{\Rightarrow} uAy \overset{*}{\Rightarrow} uvAxy \overset{*}{\Rightarrow} uv^2Ax^2y$$
$$S \overset{*}{\Rightarrow} \cdots \overset{*}{\Rightarrow} uv^iAx^iy \overset{*}{\Rightarrow} uv^iwx^iy$$

Somit ist $uv^iwx^iy \in L(G) \subseteq L$ für alle $i \geqslant 0$.
D.h. wir können tatsächlich auf- und abpumpen. □

# Anwendung und Grenzen von Ogden's Lemma

- Wir können Ogden's Lemma nutzen, um nachzuweisen, dass bestimmte Sprachen nicht kontextfrei sind.

  Beispiele dafür haben wir bereits kennen gelernt — sowohl in der Vorlesung als auch in den Übungsaufgaben.

- Wir haben auch schon eine Sprache kennen gelernt, deren Nicht-Kontextfreiheit wir mit Ogden's Lemma beweisen konnten, aber nicht mit dem Pumping Lemma.

- Aber auch Ogden's Lemma hat seine Grenzen:

  Es gibt Sprachen, die nicht kontextfrei sind, deren Nicht-Kontextfreiheit wir mit Ogden's Lemma aber nicht nachweisen können. (Details: Übung)

# Abschlusseigenschaften kontextfreier Sprachen

# Abschlusseigenschaften

## Satz:

$L, L_1, L_2 \subseteq \Sigma^*$ seien kontextfreie Sprachen.
Dann sind auch die folgenden Sprachen kontextfrei:

(a) $L_1 \cup L_2$.

(b) $L_1 \cdot L_2$.

(c) $L^*$.

(d) $L^R := \{w^R : w \in L\}$ (Rückwärtslesen von $L$).

(e) $h(L)$, wobei $h : \Sigma^* \to \Gamma^*$ ein Homomorphismus ist.

(f) $L \cap R$, wobei $R \subseteq \Sigma^*$ eine reguläre Sprache ist.

**Beweis: Übung!**

(a)–(e) folgen leicht durch Umbau der gegebenen KFGs für $L$, $L_1$, $L_2$.

Zum Beweis von (f) benutzt man am besten die Charakterisierung von KFGs durch nichtdeterministische Kellerautomaten, die im nächsten Abschnitt behandelt wird.

# Nicht-Abschlusseigenschaften

**Beobachtung:**

(a) Die Klasse der kontextfreien Sprachen ist nicht abgeschlossen unter Durchschnittsbildung, d.h.

Es gibt kontextfreie Sprachen $L_1, L_2$, so dass $L_1 \cap L_2$ **nicht** kontextfrei ist.

**Beispiel:** $L_1 := \{a^n b^n c^k : k, n \in \mathbb{N}\}$ und $L_2 := \{a^k b^n c^n : k, n \in \mathbb{N}\}$ sind beide kontextfrei, aber $L_1 \cap L_2 = \{a^n b^n c^n : n \in \mathbb{N}\}$ ist es nicht.

Aber laut dem Satz auf der vorherigen Folie ist immerhin der Durchschnitt jeder kontextfreien Sprache mit einer regulären Sprache wieder kontextfrei.

(b) Die Klasse der kontextfreien Sprachen ist nicht abgeschlossen unter Komplementbildung, d.h.

Es gibt eine kontextfreie Sprache $L$, deren Komplement $\overline{L}$ **nicht** kontextfrei ist.

**Beweis:** Wenn die kontextfreien Sprachen unter Komplementbildung abgeschlossen wären, dann auch unter Durchschnitten, denn

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}.$$

# Wer ist kontextfrei und wer nicht?

- $\{a^n b^n \mid n \in \mathbb{N}\}$ ..... (Ja!)
- $\{a^n b^m c^{n+m} \mid n, m \in \mathbb{N}\}$ ..... (Ja!)
- $\{a^n b^n c^k \mid k, n \in \mathbb{N}\} \cup \{a^k b^n c^n \mid k, n \in \mathbb{N}\}$ ..... (Ja!)
- $\{a^n b^n c^k \mid k, n \in \mathbb{N}\} \cap \{a^k b^n c^n \mid k, n \in \mathbb{N}\} = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ ..... (Nein!)
- $\{a^n b^m c^k \mid n = m = k$ gilt **nicht** $\}$ ..... (Ja!)
- $\{w \in \{a, b\}^* \mid w = w^R\}$ ..... **(Übung!)**
- $\{w \in \{a, b\}^* \mid w \neq w^R\}$ ..... **(Übung!)**
- $\{u\#v \mid u, v \in \{a, b\}^*,\ u = v\}$ ..... (Nein!)
- $\{u\#v \mid u, v \in \{a, b\}^*,\ u \neq v\}$ ..... (Ja!)
- Die Menge aller arithmetischen Ausdrücke. ..... (Ja!)
- Die Menge aller syntaktisch korrekten Java Programme..... (Nein, aber fast)

# Kellerautomaten

# Vorarbeit: Die Greibach Normalform

## Definition (Greibach Normalform)

Eine Grammatik $G = (\Sigma, V, S, P)$ in Greibach Normalform, wenn alle Produktionen die folgende Form haben:

$$A \to a\alpha \qquad \text{mit } A \in V,\ a \in \Sigma,\ \alpha \in V^*.$$

In Produktionen von Grammatiken in Greibach Normalform wird also jede Variable ersetzt durch einen Buchstaben, gefolgt evtl. von weiteren Variablen.

## Satz (hier ohne Beweis):

Jede kontextfreie Sprache $L$ mit $\varepsilon \notin L$ wird durch eine Grammatik in Greibach Normalform erzeugt.

Es gibt einen Algorithmus, der bei Eingabe einer kontextfreien Grammatik $G$ in kubischer Zeit eine Grammatik $G'$ in Greibach Normalform erzeugt mit $L(G') = L(G) \setminus \{\varepsilon\}$.

(Ein Beweis findet sich in Kapitel 7.1 des Buchs von Ingo Wegener.)

# Ein Maschinenmodell für kontextfreie Sprachen

- Aus der Perspektive einer Grammatik in Greibach Normalform:
  - Die Ableitung $S \to aX_1 \cdots X_k$ erzeugt den ersten Buchstaben.
  - Die Variablen $X_1, \ldots, X_k$ sind zu ersetzen, beginnend mit $X_1$.
  - Wenn der Ersetzungsprozess für $X_1$ abgeschlossen ist, dann ist $X_2$ dran. Danach $X_3$ usw.

- Aus der Perspektive eines Maschinenmodells:
  - Lese den ersten Buchstaben $a$.
  - Die Variablen $X_1, \ldots, X_k$ müssen abgespeichert werden. Aber mit welcher Datenstruktur?
  - Das sollte ein Keller sein! (engl: stack oder pushdown storage)

# Nichtdeterministische Kellerautomaten (PDAs, Pushdown Automata)

Ein PDA $A = (Q, \Sigma, \Gamma, q_0, Z_0, \delta)$ besteht aus

- einer endlichen Zustandsmenge $Q$,
- einem endlichen Eingabealphabet $\Sigma$,
- einem endlichen Kelleralphabet $\Gamma$,
- einem Anfangszustand $q_0 \in Q$,
- dem anfängliche Kellerinhalt $Z_0 \in \Gamma$, sowie
- dem Programm

$$\delta : Q \times \Big(\Sigma \cup \{\epsilon\}\Big) \times \Gamma \;\; \to \;\; \mathcal{P}\left(Q \times \Gamma^*\right).$$

## Wie arbeitet ein PDA?

Das Eingabewort wird von links nach rechts gelesen.

Eine Konfiguration $(q, u, \gamma)$ von $A$ besteht aus dem aktuellen Zustand $q \in Q$, dem noch nicht gelesenen Teil $u \in \Sigma^*$ des Eingabeworts und dem Kellerinhalt $\gamma \in \Gamma^*$.

Startkonfiguration bei Eingabe $w \in \Sigma^*$: $(q_0, w, Z_0)$.

# Wie arbeitet ein PDA $A$?

Die aktuelle Konfiguration sei $(q,\ a_1 \cdots a_m,\ \gamma_1 \cdots \gamma_r)$, d.h.: Der akutelle Zustand ist $q$; auf dem Eingabeband wird das Symbol $a_1$ gelesen; auf dem Keller das Symbol $\gamma_1$.

- Wenn $(q', \delta_1 \cdots \delta_s) \in \delta(q, a_1, \gamma_1)$,
  dann kann $A$ in den Zustand $q'$ gehen, den Kopf auf dem Eingabeband eine Stelle nach rechts bewegen und im Keller das oberste Symbol $\gamma_1$ durch das Wort $\delta_1 \cdots \delta_s$ ersetzen. D.h.:

  $$(q',\ a_2 \cdots a_m,\ \delta_1 \cdots \delta_s \gamma_2 \cdots \gamma_r)$$

  ist eine zulässige Nachfolgekonfiguration.

- **$\epsilon$-Übergänge:**

  Wenn $(q', \delta_1 \cdots \delta_s) \in \delta(q, \epsilon, \gamma_1)$, dann kann $A$ in den Zustand $q'$ gehen, auf dem Eingabeband den Kopf stehen lassen und im Keller das oberste Symbol $\gamma_1$ durch das Wort $\delta_1 \cdots \delta_s$ ersetzen. D.h.:

  $$(q',\ a_1 \cdots a_m,\ \delta_1 \cdots \delta_s \gamma_2 \cdots \gamma_r)$$

  ist eine zulässige Nachfolgekonfiguration.

# Wann akzeptiert ein PDA $A$?

Eine Berechnung von $A$ bei Eingabe $w \in \Sigma^*$ ist eine Folge $K = (k_0, k_1, \ldots, k_t)$ von Konfigurationen, so dass $k_0 = (q_0, w, Z_0)$ die Startkonfiguration von $A$ bei Eingabe $w$ ist, und für alle $i \geqslant 0$ gilt: $k_{i+1}$ ist eine zulässige Nachfolgekonfiguration von $k_i$.

$k_t = (q, u, \gamma)$ ist eine Endkonfiguration, wenn es keine zulässige Nachfolgekonfiguration gibt (u.a. ist dies der Fall, wenn $\gamma = \varepsilon$, d.h. der Keller leer ist).

Eine Berechnung $K = (k_0, k_1, \ldots, k_t)$ ist akzeptierend, falls $k_t = (q, \epsilon, \epsilon)$ ist (d.h. das Eingabewort wurde komplett gelesen und der Keller ist leer).

**Akzeptanz durch leeren Keller:** Eine Eingabe $w$ wird akzeptiert, wenn es eine Berechnung gibt, die mit leerem Keller endet.

## Die von einem PDA $A$ akzeptierte Sprache ist

$L(A) := \{ w \in \Sigma^* : \text{es gibt eine akzeptierende Berechnung von } A \text{ bei Eingabe } w \}.$

# Ein Beispiel

Ein PDA $A = (Q, \Sigma, \Gamma, q_0, Z_0, \delta)$, der die Sprache

$$L = \{w \cdot w^R \ : \ w \in \{a, b\}^*\}$$

akzeptiert:

- Wir arbeiten mit den Zuständen `push` (=: $q_0$), `pop` und `fail`.
- Der Automat legt zuerst die Eingabe im Zustand `push` auf dem Keller ab,
- rät die Mitte des Worts und wechselt in den Zustand `pop`.
- Jetzt wird der Keller abgebaut: Wechsle in den Zustand `fail`, wenn Eingabebuchstabe und Stackinhalt nicht übereinstimmen.
- Akzeptiere mit leerem Keller.

Details: siehe Tafel.

# Variante: Akzeptanz durch Zustände

Statt **Akzeptanz durch leeren Keller** sind auch andere Varianten denkbar, z.B.:

**Akzeptanz durch Zustände:**
Der PDA hat als zusätzliche Komponente eine Menge $F \subseteq Q$.

Eine Eingabe $w$ wird akzeptiert, wenn es eine Berechnung gibt, die das komplette Eingabewort verarbeitet und in einem Zustand $q \in F$ endet.

**Ein Beispiel:** Ein PDA mit akzeptierenden Zuständen für die Sprache $L := \{u\#v \ : \ u, v \in \{a, b\}^*, \ u \neq v \}$:

- Der Automat rät eine Position $i$ im ersten Teil der Eingabe, merkt sich den Buchstaben an dieser Position in seinem Zustand und benutzt den Keller, um sich an die Position zu erinnern (der Keller enthält ein Wort der Länge $i$).
- Dann wird verglichen, ob sich die erste und zweite Hälfte der Eingabe an der Position $i$ unterscheiden.
- Wenn ja, gehe in einen akzeptierenden Zustand und gehe auf dem Eingabeband noch ganz nach rechts.

Die beiden Varianten **Akzeptanz durch leeren Keller** und **Akzeptanz durch Zustände** sind **äquivalent**.

# Akzeptanz durch Zustände $\Longrightarrow$ Akzeptanz durch leeren Keller

Der PDA $A$ akzeptiere mit Zuständen.
Dann gibt es einen PDA $B$, der mit leerem Keller akzeptiert, so dass $L(B) = L(A)$.

Die Konstruktion von $B$:

- $B$ übernimmt das Programm von $A$.
- Wenn ein Zustand in $F$ erreicht wird, dann darf $B$ *nichtdeterministisch* seinen Keller leeren.
- Vorsicht: Wenn $A$ am Ende seiner Berechnung "zufälligerweise" einen leeren Keller produziert, dann würde $B$ akzeptieren.

  Lösung: Am Anfang ersetze das Kellersymbol $Z_0$ durch $Z_0 Z_0'$, wobei $Z_0'$ ein neues Symbol im Kelleralphabet ist.

# Akzeptanz durch leeren Keller $\Longrightarrow$ Akzeptanz durch Zustände

Der PDA $B$ akzeptiere mit leerem Keller.
Dann gibt es einen PDA $A$, der mit Zuständen akzeptiert, so dass $L(A) = L(B)$.

Die Konstruktion von $A$:

- Wir möchten, dass $A$ bei leerem Keller noch in einen akzeptierenden Zustand wechseln kann.
- Am Anfang ersetze $Z_0$ über einen $\epsilon$-Übergang durch $Z_0 Z_0'$, wobei $Z_0'$ ein neues Kellersymbol ist.
- Wenn das Kellersymbol $Z_0'$ wieder erreicht wird, dann springe in einen neuen, akzeptierenden Zustand.

# Kontextfreie Sprachen werden von PDAs akzeptiert

Sei $G = (\Sigma, V, S, P)$ eine kontextfreie Grammatik.

Dann gibt es einen PDA $A$ mit $L(A) = L(G)$.

- Der PDA $A$ arbeitet mit nur einem Zustand $q_0$, besitzt das Kelleralphabet $\Gamma := \Sigma \cup V$ und hat anfangs das Startsymbol $Z_0 := S$ im Keller.
- Wenn $A$ das Symbol $a \in \Sigma$ sowohl auf dem Eingabeband als auch auf dem Keller liest, dann darf $A$ dieses Kellersymbol löschen und den Kopf auf dem Eingabeband um eine Position nach rechts verschieben. D.h.:

$$(q_0, \varepsilon) \in \delta(q_0, a, a), \quad \text{für alle } a \in \Sigma$$

- Wenn $A$ das Symbol $X \in V$ auf dem Keller liest und wenn $X \to \alpha$ eine Produktion von $G$ ist (mit $\alpha \in (\Sigma \cup V)^*$), dann darf $A$ einen $\epsilon$-Übergang nutzen, um die auf dem Keller zuoberst liegende Variable $X$ durch das Wort $\alpha$ zu ersetzen. D.h.:

$$(q_0, \alpha) \in \delta(q_0, \varepsilon, X), \quad \text{für alle } X \in V, (X \to \alpha) \in P.$$

**Behauptung:** Dieser PDA $A$ akzeptiert genau die Sprache $L = L(G)$.

**Beweis:** Übung!

# Sind Kellerautomaten zu mächtig?

Die Sprache $L$ werde von einem PDA $A$ akzeptiert (mit leerem Keller).

**Ziel:** Konstruiere eine kontextfreie Grammatik $G$, so dass $L(G) = L(A)$.

- Sei $A = (Q, \Sigma, \Gamma, q_0, Z_0, \delta)$ der gegebene PDA.
  - Angenommen, $A$ ist im Zustand $p_1$, hat $w_1 \cdots w_k$ gelesen, und der Kellerinhalt ist $X_1 \cdots X_m$.

- **1. Idee:**

  Entwerfe die Grammatik $G$ so, dass die Ableitung $S \stackrel{*}{\Rightarrow} w_1 \cdots w_k \; p_1 \; X_1 \cdots X_m$ möglich ist.
  - $p_1$ ist die zu ersetzende Variable.
  - Problem: Der PDA $A$ arbeitet aber in Abhängigkeit von $p_1$ und $X_1$.

So kann das nicht funktionieren:
Wir bekommen so keine **kontextfreie** Grammatik!

# Wie sollen wir die Menge $V$ der Variablen wählen?

**2. Idee:**

Wir fassen das Paar $[p_1, X_1]$ als Variable auf!

Können wir

$$S \stackrel{*}{\Rightarrow} w_1 \cdots w_k \, [p_1, X_1] \, [p_2, X_2] \cdots [p_m, X_m]$$

erreichen?

- Was ist $p_1$? Der aktuelle Zustand!
- Was ist $p_2$? Der **geratene** aktuelle Zustand, wenn $X_2$ an die Spitze des Kellers gelangt.

Aber wie verifizieren wir, dass $p_2$ auch der richtige Zustand ist?

**3. Idee:**

“Erinnere dich an den geratenen Zustand” — die **Tripelkonstruktion**.

# Die Tripelkonstruktion (1/4)

Wir entwerfen die Grammatik so, dass

$$S \overset{*}{\Rightarrow} w_1 \cdots w_k \, [p_1, X_1, p_2] \, [p_2, X_2, p_3] \cdots [p_m, X_m, p_{m+1}]$$

ableitbar ist.

Die Absicht von Tripel $[p_i, X_i, p_{i+1}]$ ist, dass

- $p_i$ der aktuelle Zustand ist, wenn $X_i$ an die Spitze des Kellers gelangt und
- dass $p_{i+1}$ der aktuelle Zustand wird, wenn das direkt unter $X_i$ liegende Symbol an die Spitze des Kellers gelangt.

Wie können wir diese Absicht in die Tat umsetzen?

**Unsere Grammatik $G(\Sigma, V, S, P)$ hat die folgende Form:**

$\Sigma$ ist das Eingabealphabet des PDA $A$.
$V$ besteht dem Startsymbol $S$ und zusätzlich allen Tripeln $[q, X, p]$ mit $q, p \in Q$, $X \in \Gamma$.

# Die Tripelkonstruktion (2/4)

$P$ besteht aus folgenden Produktionen:

- $S \to [q_0, Z_0, p]$, für **jedes** $p \in Q$.

  Wir raten, dass $p$ der Schlusszustand einer akzeptierenden Berechnung ist.

- Für jeden Befehl $(p_1, X_1 \cdots X_r) \in \delta(p, a, X)$
  (mit $p_1 \in Q$, $a \in \Sigma \cup \{\varepsilon\}$ , $X_1, \ldots, X_r \in \Gamma$ für $r \geqslant 1$) und
  für alle Zustände $p_2, \ldots, p_{r+1} \in Q$ füge als Produktionen hinzu:

$$[p, X, p_{r+1}] \to a\,[p_1, X_1, p_2]\,[p_2, X_2, p_3] \cdots [p_r, X_r, p_{r+1}].$$

  Wir raten also, dass $A$ beim Offenlegen des Kellersymbols $X_i$ in den Zustand $p_i$ gelangt.

Und wie wird hier gewährleistet, dass wir die **richtigen** Zustände $p_2, \ldots, p_{r+1}$ geraten haben?

# Die Tripelkonstruktion (3/4)

Für jeden Befehl $(p_1, X_1 \cdots X_r) \in \delta(p, a, X)$
und alle $p_2, \ldots, p_{r+1} \in Q$ füge die Produktion
$[p, X, p_{r+1}] \rightarrow a\,[p_1, X_1, p_2]\,[p_2, X_2, p_3] \cdots [p_r, X_r, p_{r+1}]$ hinzu.

Direkt nachdem $A$ diesen Befehl ausgeführt hat, liegt ganz oben auf dem Keller das Symbol $X_1$.

- Wenn im nächsten Schritt $X_1$ durch ein nicht-leeres Wort ersetzt wird, dann wird die Verifikation vertagt: Der für die Offenlegung von $X_2$ geratene Zustand $p_2$ wird festgehalten.
- Wenn $X_1$ durch das leere Wort ersetzt wird, dann müssen wir verifizieren:

Für jeden Befehl $(q, \epsilon) \in \delta(p, a, X)$ nimm

$$[p, X, q] \rightarrow a$$

als neue Produktion in $P$ auf.

# Die Tripelkonstruktion (4/4)

**$P$ besteht also aus den Produktionen**

- $S \to [q_0, Z_0, p]$, für **jedes** $p \in Q$,
- $[p, X, p_{r+1}] \to a\,[p_1, X_1, p_2]\,[p_2, X_2, p_3] \cdots [p_r, X_r, p_{r+1}]$,

  für jeden Befehl $(p_1, X_1 \cdots X_r) \in \delta(p, a, X)$ von $A$ und
  für alle Zustände $p_2, \ldots, p_{r+1} \in Q$
- $[p, X, q] \to a$, für jeden Befehl $(q, \epsilon) \in \delta(p, a, X)$.

**Behauptung:**

Für alle $p, q \in Q$, $X \in \Gamma$ und $w \in \Sigma^*$ gilt:

$$[p, X, q] \overset{*}{\Rightarrow}_G w \quad \Longleftrightarrow \quad (p, w, X) \vdash_A^* (q, \varepsilon, \varepsilon).$$

$k \vdash_A^* k'$ bedeutet hier, dass es eine Berechnung von $A$ gibt, die in endlich vielen Schritten von Konfiguration $k$ zu Konfiguration $k'$ gelangt.

**Beweis:** (Details: Übung)

"$\Longrightarrow$": Per Induktion nach der Länge von Ableitungen.

"$\Longrightarrow$": Per Induktion nach der Länge von Berechnungen.

# Zusammenfassung

Insgesamt haben wir also folgende Äquivalenz von KFGs und PDAs bewiesen:

**Satz:**

Sei $\Sigma$ ein endliches Alphabet. Eine Sprache $L \subseteq \Sigma^*$ ist genau dann kontextfrei, wenn sie von einem nichtdeterministischen Kellerautomaten akzeptiert wird.

# Deterministisch kontextfreie Sprachen

# Deterministisch kontextfreie Sprachen

**Ziel:** Schränke die Definition von PDAs so ein, dass sie deterministisch sind, d.h. dass es bei jedem Schritt der Verarbeitung eines Eingabeworts $w \in \Sigma^*$ nur höchstens eine mögliche Nachfolgekonfiguration gibt.

## Definition: Ein deterministischer Kellerautomat (kurz: DPDA) ist

ein PDA $A = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$, der mit Zuständen akzeptiert, und für dessen Programm $\delta$ gilt:

für alle $q \in Q$, $a \in \Sigma$ und $X \in \Gamma$ ist $\quad |\delta(q, a, X)| + |\delta(q, \epsilon, X)| \leqslant 1.$

**Beachte:** Ein DPDA kann stets höchstens eine Anweisung ausführen.

**Frage:** Warum erlauben wir hier $\epsilon$-Übergänge?

Weil das unser Modell stärker macht. Beispielsweise wird die Sprache

$$\{a^n b^m \$ a^n : n, m \in \mathbb{N}\} \cup \{a^n b^m \# b^m : n, m \in \mathbb{N}\}$$

von einem DPDA mit $\epsilon$-Übergängen akzeptiert, aber nicht von einem "Echtzeit-DPDA", der keine $\epsilon$-Übergänge nutzt. (Details: Übung)

**Frage:** Warum akzeptieren wir hier mit Zuständen und nicht mit leerem Keller?

# Akzeptanzmodus

- Ein “DPDA”, der mit leerem Keller akzeptiert, kann stets durch einen DPDA simuliert werden, der mit Zuständen akzeptiert. **(Warum?)**
- Die Sprache $L = \{0,\ 01\}$ lässt sich mit Zuständen akzeptieren (klar: sogar durch einen DFA, der den Keller gar nicht nutzt), aber nicht durch einen “DPDA” mit Akzeptanzmodus “leerer Keller”, denn:
  - Schreibt der Automat beim Eingabezeichen 0 ein Zeichen auf den Keller, so wird das Eingabewort 0 nicht akzeptiert: Fehler.
  - Schreibt der Automat im Anfangszustand beim Eingabezeichen 0 nichts auf den Keller, so ist der Keller jetzt leer. Nachfolgende Buchstaben können jetzt nicht mehr verarbeitet werden! Fehler.
- Bei der Definition von DPDAs haben wir den stärkeren Modus „Akzeptanz durch Zustände“ gewählt. Damit ist insbesondere auch gewährleistet, dass jede reguläre Sprache von einem DPDA akzeptiert wird.

## Definition

Eine Sprache ist deterministisch kontextfrei, wenn sie von einem DPDA akzeptiert wird.

# Beispiele deterministisch kontextfreier Sprachen

- $L = \{a^n b^n \mid n \in \mathbb{N}_{>0}\}$:

  **Idee:**
  Lege zuerst alle $a$'s auf den Keller. Für jedes $b$ nimm ein $a$ vom Keller runter.

  **Details:**
  $L$ wird von dem DPDA $A = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$ akzeptiert mit $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{a, b\}$, $\Gamma = \{a, b, Z_0\}$, $F = \{q_2\}$ und

$$\begin{aligned}
\delta(q_0, a, Z_0) &= \{(q_0, aZ_0)\}, & \delta(q_0, a, a) &= \{(q_0, aa)\},\\
\delta(q_0, b, a) &= \{(q_1, \varepsilon)\}, & \delta(q_1, b, a) &= \{(q_1, \varepsilon)\},\\
\delta(q_1, \varepsilon, Z_0) &= \{(q_2, \varepsilon)\}.
\end{aligned}$$

- Die Sprache aller wohlgeformten Klammerausdrücke:

  **Idee:**

  - Lege jede öffnende Klammer auf den Keller
  - und für jede schließende Klammer entferne eine Klammer vom Keller.

  **Details:** Übung.

# Abschlusseigenschaften deterministisch kontextfreier Sprachen

## Satz:

$L, L_1, L_2 \subseteq \Sigma^*$ seien deterministisch kontextfreie Sprachen.
Dann sind auch die folgenden Sprachen deterministisch kontextfrei:

(a) $\overline{L} := \Sigma^* \setminus L$
(Abschluss unter Komplementbildung)

(b) $L \cap R$, wobei $R \subseteq \Sigma^*$ eine reguläre Sprache ist.
(Abschluss unter Durchschnitt mit regulären Sprachen)

(c) $h^{-1}(L)$, wobei $h : \Gamma^* \to \Sigma^*$ ein Homomorphismus ist.
(Abschluss unter inversen Homomorphismen)

**Beweisidee:**

(a) Sei $A$ ein DPDA, der $L$ akzeptiert. Für $\overline{L}$ bauen wir einen DPDA $B$, der ein Eingabewort genau dann verwirft, wenn $A$ nach dem Lesen des letzten Buchstabens und vor dem Lesen des nächsten Buchstabens akzeptiert.
**Vorsicht:** $\epsilon$-Übergänge von $A$ machen das ganze recht kompliziert.
Details finden sich in Kapitel 8 des Buchs von Wegener.

(b) und (c): Gleicher Beweis wie bei kontextfreien Sprachen (Übung!)

# Nicht-Abschlusseigenschaften

**Beobachtung:**

(a) Die Klasse der deterministisch kontextfreien Sprachen ist nicht abgeschlossen unter Durchschnittbildung, d.h.

Es gibt deterministisch kontextfreie Sprachen $L_1, L_2$, so dass $L_1 \cap L_2$ **nicht** deterministisch kontextfrei ist.

**Beispiel:** $L_1 := \{a^n b^n c^k : k, n \in \mathbb{N}\}$ und $L_2 := \{a^k b^n c^n : k, n \in \mathbb{N}\}$ sind beide deterministisch kontextfrei, aber $L_1 \cap L_2 = \{a^n b^n c^n : n \in \mathbb{N}\}$ ist nicht kontextfrei, also auch nicht deterministisch kontextfrei.

(b) Die Klasse der deterministisch kontextfreien Sprachen ist nicht abgeschlossen unter Vereinigung, d.h.

Es gibt deterministisch kontextfreie Sprachen $L_1, L_2$, so dass $L_1 \cup L_2$ **nicht** deterministisch kontextfrei ist.

**Beweis:** Übung!

# Weitere Nicht-Abschlusseigenschaften

**Beobachtung:**

Die Klasse der deterministisch kontextfreien Sprachen ist
**nicht** abgeschlossen unter

(a) Konkatenation, d.h. es gibt deterministisch kontextfreie Sprachen $L_1, L_2$, so dass die Sprache $L_1 \cdot L_2$ **nicht** deterministisch kontextfrei ist.

(b) Kleene-Stern, d.h. es gibt eine deterministisch kontextfreie Sprache $L$, so dass die Sprache $L^*$ **nicht** deterministisch kontextfrei ist.

(c) Rückwärtslesen, d.h. es gibt eine deterministisch kontextfreie Sprache $L$, so dass die Sprache $L^R$ **nicht** deterministisch kontextfrei ist.

(d) Homomorphismen, d.h. es gibt eine deterministisch kontextfreie Sprache $L$ und einen Homomorphismus $h$, so dass $h(L)$ **nicht** deterministisch kontextfrei ist.

**Beweis:** Übung!

# Was weiss man noch über deterministisch kontextfreie Sprachen?

- Jede reguläre Sprache ist deterministisch kontextfrei.
- Jede endliche Sprache ist regulär.
- Jede deterministisch kontextfreie Sprache ist eindeutig. **(Warum?)**

- Das Wortproblem für deterministisch kontextfreie Sprachen $L \subseteq \Sigma^*$

| | |
|---|---|
| *Eingabe:* | Ein Wort $w \in \Sigma^*$. |
| *Frage:* | Ist $w \in L$? |

kann in Linearzeit gelöst werden.

**Begründung:**
Sei $A$ ein DPDA, der $L$ akzeptiert. Bei Eingabe von $w$ lasse $A$ laufen.
Wenn $A$ keine $\epsilon$-Übergänge nutzt, dann macht $A$ höchstens $|w|$ Schritte.
Auch für den Fall, dass $A$ $\epsilon$-Übergänge nutzt, kann man zeigen, dass $O(|w|)$ Schitte ausreichen, um zu entscheiden, ob $w$ akzeptiert wird oder ob der DPDA in einer Endlosschleife gefangen ist. (**Details:** Übung.)

# Einige Beispiele (1/2)

- $L := \{ a^n b^m c^k \mid n, m, k \in \mathbb{N},\ n = m = k \text{ gilt } \mathbf{nicht} \}$ ist kontextfrei, aber nicht deterministisch kontextfrei.

  Kontextfrei: Haben wir bereits gesehen.
  Nicht deterministisch kontextfrei:

  **Intuitiver Grund:**
  Ein DPDA weiss nicht, ob er $a's$ mit $b$'s oder $a$'s mit $c$'s vergleichen soll.
  **Formaler Grund:**

  - Das Komplement der Sprache, geschnitten mit $L(a^* b^* c^*)$, ist $\{a^n b^n c^n : n \in \mathbb{N}\}$, also nicht kontextfrei.
  - Aber die Klasse der deterministisch kontextfreien Sprachen ist abgeschlossen unter Komplemenbildung und Schnitt mit regulären Sprachen.

  Man kann sogar zeigen, dass die Sprache $L$ inhärent mehrdeutig ist.

- $L = \{w \cdot w^R \mid w \in \{a, b\}^*\}$ ist eindeutig, aber nicht deterministisch kontextfrei.

  Eindeutig: Wir können leicht eine eindeutige KFG angeben, die $L$ erzeugt.
  Nicht deterministisch kontextfrei:

  **Intuitiver Grund:** Ein DPDA kann die Mitte eines Worts nicht raten.
  **Formaler Grund:** siehe Buch von Hopcroft und Ullman

# Einige Beispiele (2/2)

- $\{a^n b^m c^k \mid n, m, k \in \mathbb{N},\ n \neq m\}$ ist deterministisch kontextfrei.
- $\{a^n b^m c^k \mid n, m, k \in \mathbb{N},\ n \neq k\} \cup \{a^n b^m c^k \mid n, m, k \in \mathbb{N},\ m \neq k\}$ ist kontextfrei, aber nicht deterministisch kontextfrei.
  Die Sprache ist sogar inhärent mehrdeutig.
- $L_k = \{0,1\}^* \{1\} \{0,1\}^k$ ist regulär.
- $\{u\#v \mid u, v \in \{a,b\}^*,\ u = v\}$ ist nicht kontextfrei.
- $\{u\#v \mid u, v \in \{a,b\}^*,\ u \neq v\}$ ist kontextfrei, aber nicht deterministisch kontextfrei.

# Das Wortproblem für kontextfreie Sprachen

# Das Wortproblem für kontextfreie Sprachen (1/2)

Die Klasse der kontextfreien Sprachen bildet die Grundlage für die Syntaxspezifikation moderner Programmiersprachen.

Um zu überprüfen, ob ein eingegebener Programmtext ein syntaktisch korrektes Programm der jeweiligen Programmiersprache ist, muss ein Compiler das folgende Problem lösen:

Sei $G = (\Sigma, V, S, P)$ eine kontextfreie Grammatik, die die Grundform syntaktisch korrekter Programme unserer Programmiersprache beschreibt.

Das Wortproblem für $G = (\Sigma, V, S, P)$

*Eingabe:* Ein Wort $w \in \Sigma^*$

*Frage:* Ist $w \in L(G)$?

Wenn wir zusätzlich noch einen Ableitungsbaum für $w$ konstruieren, haben wir eine erfolgreiche Syntaxanalyse durchgeführt.

Natürlich wollen wir das Wortproblem möglichst schnell lösen.

# Das Wortproblem für kontextfreie Sprachen (2/2)

Wir wissen bereits, dass wir das Wortproblem für deterministisch kontextfreie Sprachen in Linearzeit lösen können.

In einer Veranstaltung zum Thema “Compilerbau” können Sie lernen, dass DPDAs durch bestimmte kontextfreie Grammatiken charakterisiert werden, so genannte LR($k$)-Grammatiken.

Aber wie schwer ist das Wortproblem für allgemeine kontextfreie Grammatiken?

Es reicht, KFGs in Chomsky Normalform zu betrachten (d.h. alle Produktionen haben die Form $A \to BC$ oder $A \to a$ für $A, B, C \in V$ und $a \in \Sigma$), denn wir wissen bereits, dass jede KFG $G$ in quadratischer Zeit in eine KFG $G'$ in Chomsky Normalform mit $L(G') = L(G) \setminus \{\varepsilon\}$ umgeformt werden kann.

## Der Satz von Cocke, Younger und Kasami

Sei $G = (\Sigma, V, S, P)$ eine kontextfreie Grammatik in Chomsky-Normalform.
Dann kann in Zeit

$$O\big(|P| \cdot |w|^3\big)$$

entschieden werden, ob $w \in L(G)$.

**Beweis:**

- Die Eingabe sei $w = w_1 \cdots w_n \in \Sigma^n$ (mit $n \geqslant 1$).
- Wir benutzen die Methode der dynamischen Programmierung.

  Für alle $i, j$ mit $1 \leqslant i \leqslant j \leqslant n$ bestimmen wir die Mengen

$$V_{i,j} := \big\{ A \in V : A \stackrel{*}{\Rightarrow} w_i \cdots w_j \big\}.$$

**Klar:** $w \in L(G) \iff S \in V_{1,n}$.

# Eine Lösung der Teilprobleme (1/2)

**Ziel:** Bestimme $V_{i,j} := \{ A \in V : A \stackrel{*}{\Rightarrow} w_i \cdots w_j \}$.

- Die Bestimmung der Mengen $V_{i,i}$ (für alle $i$ mit $1 \leqslant i \leqslant n$) ist leicht, denn $G$ ist in Chomsky Normalform,
  d.h. alle Produktionen sind von der Form $A \to BC$ oder $A \to a$. Somit ist

$$V_{i,i} = \{ A \in V : A \stackrel{*}{\Rightarrow} w_i \} = \{ A \in V : (A \to w_i) \in P \}.$$

- **Beachte:**
  Die Bestimmung der Mengen $V_{i,j}$ wird umso komplizierter je größer $j - i$.

  Wir bestimmen daher nach und nach für alle $s = 0, 1, 2 \ldots, n-1$ nur diejenigen Mengen $V_{i,j}$, für die $j-i = s$ ist.

  Für $s = 0$ wissen wir bereits, dass $V_{i,i} = \{A \in V : (A \to w_i) \in P\}$ ist.

  Um von $s-1$ nach $s$ zu gelangen, nehmen wir an, dass die Mengen $V_{i,j}$ mit $j-i < s$ bereits alle bekannt sind. Dann sieht $V_{i,j}$ für $j-i = s$ wie folgt aus:

# Eine Lösung der Teilprobleme (2/2)

**Ziel:** Bestimme $V_{i,j} := \{ A \in V : A \stackrel{*}{\Rightarrow} w_i \cdots w_j \}$.

- Wir nehmen an, dass $s \geqslant 1$ ist und dass alle Mengen $V_{i,j}$ mit $j-i < s$ bereits bekannt sind.
- Nun wollen wir die Mengen $V_{i,j}$ für $j-i = s$ bestimmen.
- Variable $A$ gehört genau dann zu $V_{i,j}$, wenn

$$A \stackrel{*}{\Rightarrow} w_i \cdots w_j,$$

d.h. wenn es eine Produktion $(A \to BC) \in P$ und ein $k$ mit $i \leqslant k < j$ gibt, so dass

$$B \stackrel{*}{\Rightarrow} w_i \cdots w_k \quad \text{und} \quad C \stackrel{*}{\Rightarrow} w_{k+1} \cdots w_j, \qquad \text{d.h.}$$

$$B \in V_{i,k} \quad \text{und} \quad C \in V_{k+1,j}.$$

- Somit ist

$$V_{i,j} = \left\{ A \in V : \begin{array}{l} \text{es gibt } (A \to BC) \in P \text{ und } k \text{ mit } i \leqslant k < j, \\ \text{so dass } B \in V_{i,k} \text{ und } C \in V_{k+1,j} \end{array} \right\}$$

- Für jedes feste $i, j$ können wir $V_{i,j}$ in Zeit $O(|P| \cdot n)$ bestimmen.

# Der CYK-Algorithmus

**Eingabe:** Eine KFG $G = (\Sigma, V, S, P)$ in Chomsky Normalform und ein Wort $w = w_1 \cdots w_n \in \Sigma^n$ (für $n \geqslant 1$).

**Frage:** Ist $w \in L(G)$?

**Algorithmus:**

(1) Für alle $i = 1, \ldots, n$ bestimme $V_{i,i} := \{A \in V : (A \to w_i) \in P\}$

(2) Für alle $s = 1, \ldots, n-1$ tue Folgendes:

(3) Für alle $i, j$ mit $1 \leqslant i < j \leqslant n$ und $j - i = s$ bestimme

$$V_{i,j} := \left\{ A \in V : \begin{array}{l} \text{es gibt } (A \to BC) \in P \text{ und } k \text{ mit } i \leqslant k < j, \\ \text{so dass } B \in V_{i,k} \text{ und } C \in V_{k+1,j} \end{array} \right\}.$$

(4) Falls $S \in V_{1,n}$ ist, so gib "ja" aus; sonst gib "nein" aus.

Dieser Algorithmus löst das Wortproblem! **Laufzeit:**

- für (1): $O(n \cdot |P|)$ Schritte
- jedes einzelne $V_{i,j}$ in (3) benötigt $O(|P| \cdot n)$ Schritte; insgesamt werden $O(n^2)$ dieser $V_{i,j}$ bestimmt.
- Somit ist die Gesamtlaufzeit $O(|P| \cdot n^3)$.

# Ein Beispiel

Sei $G$ die KFG in Chomsky Normalform mit den folgenden Produktionen:

$$\begin{array}{lcl} S & \to & AB \\ A & \to & CD \mid CF \\ B & \to & c \mid EB \\ C & \to & a \end{array} \qquad \begin{array}{lcl} D & \to & b \\ E & \to & c \\ F & \to & AD \end{array}$$

**Frage:** Gehört $w := aaabbbcc$ zu $L(G)$?

Die vom CYK-Algorithmus berechneten Mengen $V_{i,j}$ können wir wir folgt als Tabelle aufschreiben:

— siehe Tafel (bzw. Kapitel 1.3.4 im Buch von Schöning) —

Der CYK-Algorithmus löst das Wortproblem.

Aber können wir damit auch einen Ableitungsbaum bestimmen?

JA! Dazu müssen wir im CYK-Algorithmus beim Einfügen eines $A$ in die Menge $V_{i,j}$ zusätzlich speichern, warum es eingefügt wurde — d.h. wir merken uns die Produktion $(A \to BC) \in P$ und die Zahl $k$ mit $i \leqslant k < j$ so dass $B \in V_{i,k}$ und $C \in V_{k+1,j}$.

Wenn $S \in V_{1,n}$ ist, können wir anhand dieser gespeicherten Informationen direkt einen Ableitungsbaum für $w$ erzeugen.

# Zusammenfassung: Die wichtigsten Konzepte (1/2)

- Kontextfreie Grammatiken definieren die Syntax moderner Programmiersprachen.
  - Die Konstruktion eines Ableitungsbaums ist das wesentliche Ziel der Syntaxanalyse.
  - Der CYK-Algorithmus erlaubt eine effiziente Lösung des Wortproblems wie auch eine schnelle Bestimmung eines Ableitungsbaums in Zeit $O(|P| \cdot n^3)$.
- Wir haben eindeutige Grammatiken und Sprachen definiert.
  - Eindeutige Grammatiken definieren eine eindeutige Semantik.
  - Es gibt inhärent mehrdeutige kontextfreie Sprachen wie etwa $\{a^k b^l c^m \ : \ k = l = m \ \text{ gilt } \textbf{nicht} \ \}$.
- Wir haben die deterministisch kontextfreien Sprachen kennen gelernt.
  - Für diese kann das Wortproblem sogar in Linearzeit gelöst werden.
  - Jede deterministisch kontextfreie Sprache ist eindeutig.
  - Aber es gibt Sprachen, die eindeutig, aber nicht deterministisch kontextfrei sind. Beispiel: $\{w \cdot w^R : w \in \{a, b\}^*\}$

# Zusammenfassung: Die wichtigsten Resultate (2/2)

- Mit dem Pumping Lemma oder mit Ogden's Lemma oder den Abschlusseigenschaften der Klasse der kontextfreien Sprachen können wir nachweisen, dass (manche) Sprachen NICHT kontextfrei sind.
- Kontextfreie Grammatiken und nichtdeterministische Kellerautomaten definieren dieselbe Sprachenklasse.
  - Kellerautomaten haben uns erlaubt, deterministisch kontextfreie Sprachen einzuführen.
  - Deterministisch kontextfreie Sprachen sind eindeutig.
  - Ihr Wortproblem kann in Linearzeit gelöst werden.
    Da man die Syntax von Programmiersprachen durch deterministisch kontextfreie Sprachen definiert, ist die Compilierung super-schnell.
- Normalformen:
  - Die Chomsky Normalform war wesentlich für die Lösung des Wortproblems (und praktisch für den Beweis von Ogden's Lemma).
  - Die Greibach Normalform hat geholfen, das Konzept der Kellerautomaten herzuleiten. (Mit ihrer Hilfe kann man auch die Äquivalenz von ndet. PDAs und "Echtzeit-PDAs", d.h. ndet. PDAs ohne $\epsilon$-Übergänge, nachweisen.)