# Kapitel 3:
# Kontextfreie Sprachen

# Themen

(a) Wann ist eine Sprache nicht kontextfrei?
- Das Pumping Lemma und Ogden's Lemma.

(b) Abschlusseigenschaften kontextfreier Sprachen.

(c) Kellerautomaten
- Kellerautomaten und Grammatiken: Äquivalente Sichtweisen.
- Deterministisch kontextfreie Sprachen.

(d) Das Wortproblem für kontextfreie Sprachen.
- Der Algorithmus von Cocke, Younger und Kasami.
- Das Wortproblem für deterministisch kontextfreie Sprachen :-))

(e) Entscheidungsprobleme :-((
- Das Postsche Korrespondenzproblem.

# Die Syntax von Programmiersprachen

Wie lässt sich die

**Syntax einer Programmiersprache**

definieren, so dass die **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

(a) 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**.

(b) 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.

**Idee:**

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

Wir benutzen die Produktionen

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

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

# Eine KFG für ein Fragment von Pascal

- 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.

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

# Programmiersprachen und kontextfreie Sprachen

Lassen sich die **syntaktisch korrekten** Programme einer 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/se8/html/index.html`
(zuletzt besucht am 10.05.2017)

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

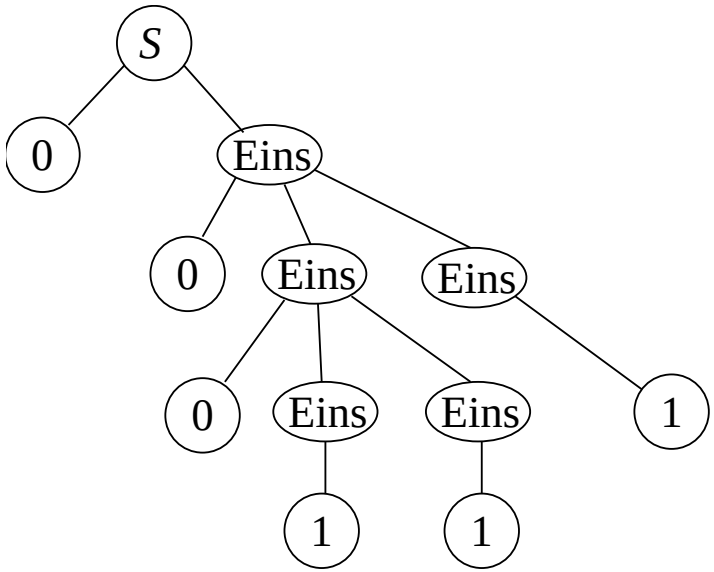
- $w$ ist die links-nach-rechts Konkatenation der Blätter von $B$.
- Die Wurzel von $B$ ist mit $S$ markiert.
- Innere Knoten sind mit Variablen, Blätter mit Elementen von $\Sigma \cup \{\varepsilon\}$ markiert.
- Für jeden Knoten $v$ gilt:
  - Wenn $v$ mit dem Symbol $\varepsilon$ markiert ist, so ist $v$ 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

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

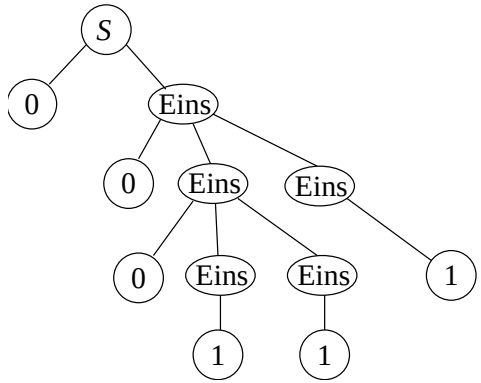
$$
\begin{aligned}
S &\Rightarrow 0\,\text{Eins} \Rightarrow 00\,\text{Eins}\,\text{Eins} \Rightarrow 000\,\text{Eins}\,\text{Eins}\,\text{Eins} \\
&\Rightarrow 0001\,\text{Eins}\,\text{Eins} \Rightarrow 00011\,\text{Eins} \Rightarrow 000111 \quad \text{mit dem Baum}
\end{aligned}
$$



# Links- und Rechtsableitungen

Ein Ableitungsbaum „produziert“

- **Linksableitungen**: Ersetze jeweils die linkeste Variable
- und **Rechtsableitungen**: Ersetze die jeweils 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

Unterschiedliche Ableitungsbäume $\Longrightarrow$ (mgl.) unterschiedliche Semantik

(a) Wir nennen eine Grammatik **mehrdeutig**, wenn ein Wort zwei oder mehrere Ableitungsbäume besitzt.

(b) Eine Grammatik ist **eindeutig**, wenn jedes Wort höchstens einen Ableitungsbaum besitzt.

(c) Ein Sprache $L$ ist **eindeutig**, wenn $L = L(G)$ für eine eindeutige kontextfreie Grammatik $G$ gilt. Ansonsten heißt $L$ **inhärent mehrdeutig**.

Die folgende kontextfreie Sprache ist inhärent mehrdeutig:

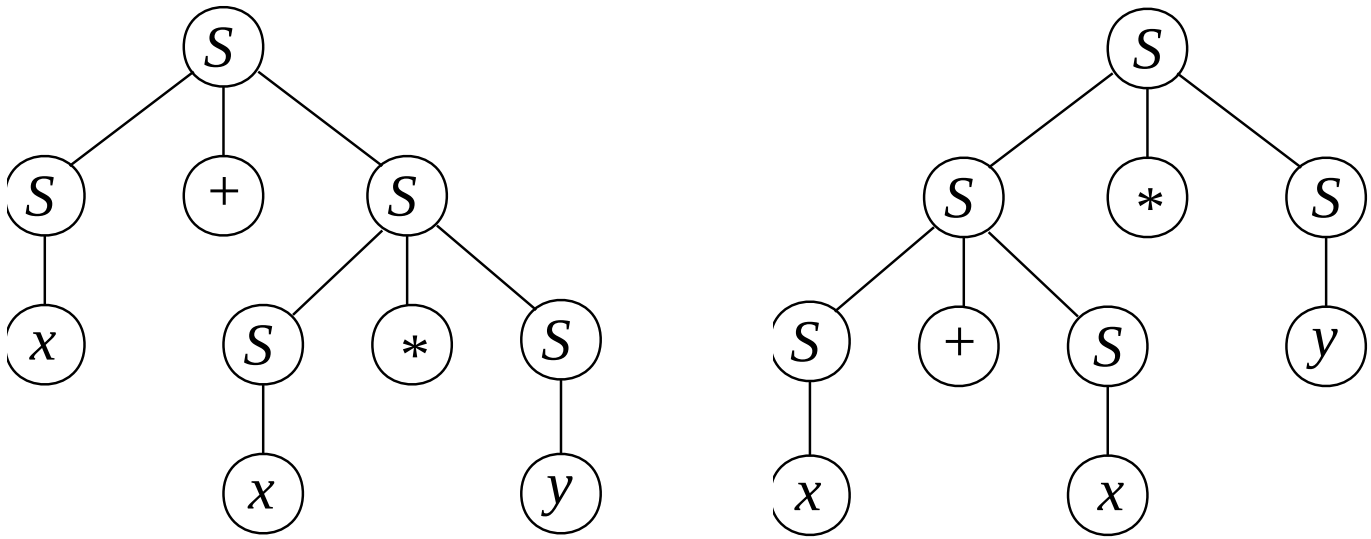
$$\{ a^j b^k c^\ell \ : \ j, k, \ell \in \mathbb{N} \ \text{ mit } \ j = k \ \text{ oder } \ k = \ell \}$$

Beweis: **Übungsaufgabe**.

# Beispiel: Arithmetische Ausdrücke (1/4)

Die Produktionen $S \to S + S \mid S * S \mid (S) \mid x \mid y$
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!

# Arithmetische Ausdrücke: Eine eindeutige Grammatik

Die neue Grammatik $G$ legt fest, dass Multiplikation stärker "bindet" als Addition.

- $V := \{S, T, F\}$: $S$ ist das Startsymbol, $T$ „erzeugt" Terme, $F$ „erzeugt" 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?

Welche Ableitungen hat $x + x * y$?

# Nachweis der Eindeutigkeit: Induktionsanfang

Die Produktionen von $G$:

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

Zeige für jede Variable $X$ und jeden arithmetischen Ausdruck $A$:

$A$ besitzt höchstens einen Ableitungsbaum mit Wurzel $X$.

**Induktionsanfang:** $|A| = 1$

- Die Länge von $A$ ist 1 $\implies$ $A = x$ oder $A = y$.
- Nur die Ableitungen $S \Rightarrow T \Rightarrow F \Rightarrow x$ bzw. $S \Rightarrow T \Rightarrow F \Rightarrow y$ sind möglich.

Insbesondere hat $A$ genau einen Ableitungsbaum.

# Nachweis der Eindeutigkeit: Induktionsschritt

Die Produktionen von $G$:

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

**Induktionsschritt:** $|A| = n + 1$

Welche Produktion wurde zuerst in der Ableitung von $A$ angewandt?

**Fall 1:** Die Produktion $S \to S + T$ wird als Erste angewandt.

- $S \to S + T$ führt den Buchstaben $+$ ohne äußere Klammerung ein.
- Jede andere erste Ableitung muss äußere Klammern um $+$ setzen setzen.

$\Longrightarrow$ Jeder Ableitungsbaum $B$ von $A$ beginnt mit der Produktion $S \to S + T$

$\Longrightarrow$ Die mit $S$ und $T$ beschrifteten Kinder der Wurzel erzeugen nach Induktionsannahme Ausdrücke mit eindeutigen Ableitungsbäumen. ✓

Die anderen Fälle werden analog behandelt. □

# Die Syntax von if-then-else

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 eindeutige Grammatik, die dieselbe Sprache erzeugt!

# Das Pumping Lemma

# 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 $L$ hinausführt (d.h. $uv^i wx^i y \notin L$), dann ist $L$

**NICHT** kontextfrei.

# Anwendung des Pumping Lemmas: Die Spielregeln

Wie zeigt man mit dem Pumping Lemma, 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

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) Was passiert?

- 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 gewonnen, die Sprache $L$ ist nicht kontextfrei.

# 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 $i \geqslant 0$.

# 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
  - $uv^iwx^iy \in L$ 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 muss gewährleisten, dass
  - $vwx$ höchstens $N$ markierte Positionen und
  - $vx$ **mindestens eine markierte Position** besitzt!

Das Pumping Lemma folgt aus Ogden's Lemma: Markiere **alle** Buchstaben von $z$.

# Ogden's Lemma: Ein Anwendungsbeispiel

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 $\leqslant N$ markierte Positionen.
- $\Longrightarrow$ Nur zwei der drei Buchstaben $b, c, d$ werden gepumpt.
- $\Longrightarrow$ 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 $uv^2wx^2y \notin L$.

# Vorarbeit zum Beweis von Ogden's Lemma: Chomsky Normalform

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

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

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

Beweis: Skript.

**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“:

- Es gibt eine Grammatik $G = (\Sigma, V, S, P)$ in Chomsky Normalform mit

$$L(G) = L \setminus \{\varepsilon\}.$$

- Wir wählen die Pumpingkonstante $N := 2^{|V|+1}$ (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: Blätter und ihre Elternknoten sind die einzigen Knoten vom Grad 1, alle anderen Knoten haben Grad 2.
- $B$ hat mindestens $N = 2^{|V|+1}$ **markierte Blätter**, also Blätter zu einem markierten Buchstaben.

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:

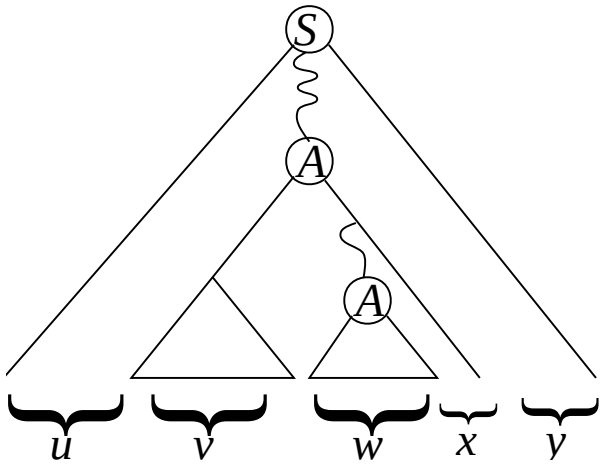
$W$ besitzt mindestens $|V| + 1$ Verzweigungsknoten. Warum?

- 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.

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

**Wir wissen:** $W$ ist ein Weg in $B$ von der Wurzel zu einem Blatt und $W$ hat mindestens $|V| + 1$ Verzweigungsknoten. $\Longrightarrow$

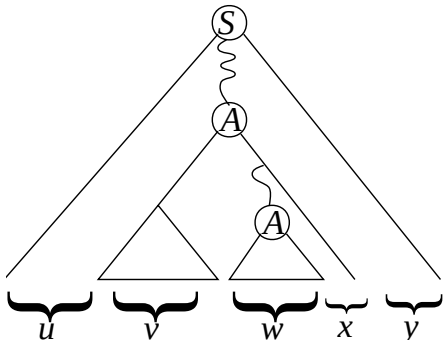
Unter den **letzten** $|V| + 1$ Verzweigungsknoten von $W$ kommt eine Variable $A$ zweimal vor!




- Das **vorletzte** Vorkommen von $A$ nutzen wir zur Definition von
  - $v$: Konkatenation der Blätter im linken Teilbaum,
  - $w$: Konkatenation der Blätter im Teilbaum zum letzten Vorkommen von $A$,
  - $wx$: Konkatenation der Blätter im rechten Teilbaum.

  Implizite werden damit auch $u$ und $y$ definiert: siehe Skizze.

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

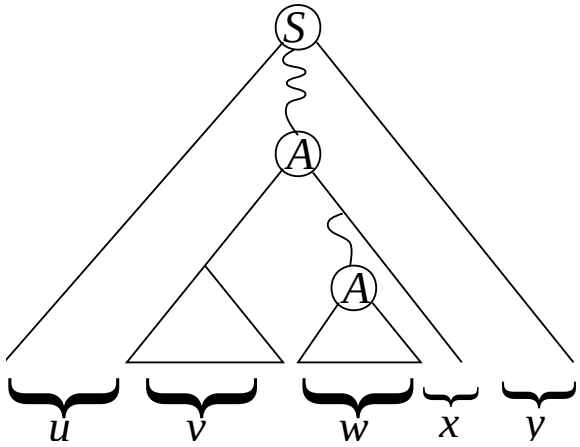



- Beide gezeigten Vorkommen von $A$ betreffen Verzweigungsknoten $\Longrightarrow$ $vx$ besitzt mindestens einen markierten Buchstaben.
- $vwx$ besitzt höchstens $2^{|V|+1} = N$ markierte Buchstaben, denn
  - Wenn $vwx$ mehr als $N$ markierte Buchstaben besitzt, dann besitzt $W$ (auf seinem Endstück) mehr als $|V| + 1$ Verzweigungsknoten.

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

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

$uv^iwx^iy \in L(G) \subseteq L$ gilt für alle $i \geqslant 0$: Wir können auf- und abpumpen. □

# Anwendung und Grenzen von Ogden's Lemma

1. Wir können Ogden's Lemma nutzen, um nachzuweisen, dass bestimmte Sprachen nicht kontextfrei sind.
   - Wir kennen eine Sprache, deren Nicht-Kontextfreiheit wir mit Ogden's Lemma beweisen konnten, aber nicht mit dem Pumping Lemma.
2. Es gibt aber auch Sprachen, die nicht kontextfrei sind, deren Nicht-Kontextfreiheit wir mit Ogden's Lemma aber nicht nachweisen können.

# Abschlusseigenschaften kontextfreier Sprachen

# Abschlusseigenschaften

$L, L_1, L_2 \subseteq \Sigma^*$ seien kontextfreie Sprachen. Dann sind auch 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) $L \cap R$, wobei $R \subseteq \Sigma^*$ eine reguläre Sprache ist.

# 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 immerhin ist der Durchschnitt einer 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!)
- $\{w \in \{a, b\}^* \mid w = w^R\}$ ..... (Ja!)
- $\{w \in \{a, b\}^* \mid w \neq w^R\}$ ..... (?)
- $\{u\#v \mid u, v \in \{a, b\}^*,\ u = v\}$ ..... (Nein!)
- $\{u\#v \mid u, v \in \{a, b\}^*,\ u \neq v\}$ ..... (?)
- Die Menge aller arithmetischen Ausdrücke. ..... (Ja!)
- Die Menge aller syntaktisch korrekten Java Programme. ..... (Nein, aber fast)

# Kellerautomaten

# Vorarbeit: Die Greibach Normalform

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

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

(b) 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.)

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

# 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 $a$.
  - In einer Linksableitung sind die Variablen $X_1, \ldots, X_k$ zu ersetzen.
    - ⋆ Beginne 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, der PDA führt Zustandsübergänge durch und benutzt seinen Stack als Speicher.

# Konfigurationen

# Wie arbeitet ein PDA $A$? (1/2)

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

Die aktuelle Konfiguration sei

$$(q,\ a_1 \cdots a_m,\ \gamma_1\gamma_2 \cdots \gamma_r)$$

Der aktuelle Zustand ist $q$; auf dem Eingabeband wird das Symbol $a_1$ gelesen, auf dem Keller das oberste 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.:

  $$(q, a_1 \cdots a_m, \gamma_1\gamma_2 \cdots \gamma_r) \vdash_A (q',\ a_2 \cdots a_m,\ \delta_1 \cdots \delta_s\gamma_2 \cdots \gamma_r)$$

  und wir haben eine von mgl. vielen **Nachfolgekonfigurationen** erhalten.

# Wie arbeitet ein PDA $A$? (2/2)

Die aktuelle Konfiguration sei

$$(q,\ a_1 \cdots a_m,\ \gamma_1\gamma_2 \cdots \gamma_r)$$

- **$\epsilon$-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 und im Keller das oberste Symbol $\gamma_1$ durch das Wort $\delta_1 \cdots \delta_s$ ersetzen.

  D.h.:

$$(q, a_1 \cdots a_m, \gamma_1\gamma_2 \cdots \gamma_r) \vdash_A (q',\ a_1 \cdots a_m,\ \delta_1 \cdots \delta_s\gamma_2 \cdots \gamma_r)$$

  und wir haben auch hier eine von mgl. vielen **Nachfolgekonfigurationen** erhalten.

# Akzeptanz-Modus

# 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, s. d.

- $k_0 = (q_0, w, Z_0)$ die Startkonfiguration von $A$ bei Eingabe $w$ ist,
- $k_{i+1}$ eine **Nachfolgekonfiguration** von $k_i$ für alle $i \geqslant 0$ ist (also $k_i \vdash_A k_{i+1}$)
- und $k_t = (q, u, \gamma)$ eine **Endkonfiguration** ist.

($k$ ist eine Endkonfiguration, wenn $k$ keine Nachfolgekonfiguration besitzt – also z. B., wenn $\gamma = \varepsilon$, d.h. wenn der Keller leer ist oder $u = \varepsilon$, d.h. die Eingabe wurde gelesen.)

Eine Eingabe $w$ wird **akzeptiert**, wenn es eine Berechnung gibt, die das komplette Eingabewort verarbeitet und

(a) mit leerem Keller endet: **Akzeptanz mit leerem Stack**,

(b) mit einem Zustand in $F$ endet: **Akzeptanz mit leerem Zustand**.
(Dazu spezifiziere eine Teilmenge $F \subseteq Q$.)

Die von einem PDA $A$ akzeptierte Sprache ist

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

# Beispiele (1/2)

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

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

mit **leerem Keller** akzeptiert:

- $A$ besitzt die Zustände `push` (=: $q_0$), `pop` und `fail`.
- $A$ 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, und erzwinge einen nicht-leeren Keller.

# Beispiele (2/2)

Ein PDA $A$, der die Sprache die Sprache

$$L := \{ a^i b^j c^k \ : \ i \neq j \text{ oder } j \neq k \}$$

mit **Zuständen** akzeptiert.

$A$ rät ob $i \neq j$ oder $j \neq k$ gilt. Angenommen $A$ rät, dass $i \neq j$.

- $A$ überprüft, ob
  - die Eingabe aus $a$'s, dann aus $b$'s und schließlich aus $c$'s besteht und ob
  - ob „$a$-Turm“ und „$b$-Turm“ ungleich hoch sind.
    - $A$ legt für jedes gelesene $a$ ein $b$ auf den Stack.
- Sind beide Bedingungen erfüllt, verbleibt $A$ in einem akzeptierenden Zustand.

Und wie sieht ein PDA aus für die Sprache

$$L = \{ u\#v \ : \ u, v \in \Sigma^*, \ u \neq v \} \ ?$$

# 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 der neue PDA $B$ 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 $Z_0'$ entfernt wird, dann springe in einen neuen, akzeptierenden „Schatten-Zustand“.

# Von KFGs zu PDAs

# PDAs akzeptieren alle kontextfreie Sprachen (1/2)

Sei $G = (\Sigma, V, S, P)$ eine Grammatik in **Greibach Normalform** (d.h. alle Produktionen sind von der Form $A \to a\alpha$ mit $A \in V$, $a \in \Sigma$, $\alpha \in V^*$).

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

- Der PDA $A$
  - arbeitet mit nur **einem**(!) Zustand $q_0$,
  - besitzt das Kelleralphabet $\Gamma := V$
  - und hat anfangs das Startsymbol $Z_0 := S$ im Keller.
- Wenn $A$ das Symbol $a \in \Sigma$ auf dem Eingabeband und das Symbol $X \in V$ auf dem Keller liest, und wenn $X \to a\alpha$ eine Produktion von $G$ ist, dann darf $A$ die auf dem Keller zuoberst liegende Variable $X$ durch das Wort $\alpha \in V^*$ ersetzen. Also

$$\delta(q_0, a, X) \;=\; \{\, (q_0, \alpha) \;:\; (X \to a\alpha) \in P \,\}.$$

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

# PDAs akzeptieren alle kontextfreie Sprachen (2/2)

Sei $G = (\Sigma, V, S, P)$ eine kontextfreie Grammatik (nicht notwendigerweise in Greibach Normalform). 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 $L = L(G)$.

# Von PDAs zu KFGs,
# die Tripelkonstruktion

# Sind Kellerautomaten zu mächtig? (1/2)

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 erhalten keine **kontextfreie** Grammatik!

# Sind Kellerautomaten zu mächtig? (2/2)

**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)

Der PDA $A$ hat $w_1 \cdots w_k$ gelesen, sein Kellerinhalt ist $X_1 \cdots X_m$ und $p$ ist sein Zustand. Wir entwerfen die Grammatik so, dass

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

ableitbar ist.

Die Absicht hinter 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 aus $S$ 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 $a \in \Sigma \cup \{\varepsilon\}$, $X_1, \ldots, X_r \in \Gamma$ 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, dass $A$ das Kellersymbols $X_i$ im Zustand $p_i$ offenlegt.

Wie wird überprüft, dass die **richtigen** Zustände $p_2, \ldots, p_{r+1}$ geraten wurden?

# 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

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

als Produktion hinzu.

Direkt nachdem $A$ den Befehl

$$(p_1, X_1 \cdots X_r) \in \delta(p, a, X)$$

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: $[p_2, X_2, p_3]$ „wartet“ im Keller.
- 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 hinzu.

# 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 $(\underline{q}, \epsilon) \in \delta(\underline{p}, a, X)$.

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

# Äquivalenz von KFGs und PDAs

Sei $\Sigma$ ein endliches Alphabet. Dann gilt

Eine Sprache $L \subseteq \Sigma^*$ ist kontextfrei $\iff$
$L$ wird von einem nichtdeterministischen Kellerautomaten akzeptiert.

Und die Konsequenzen?

1. KFGs und PDAs: Zwei äquivalente, aber unterschiedliche Sichtweisen.
   - Warum sind kontextfreie Sprachen abgeschlossen unter Durchschnitt mit einer regulären Sprache?
2. PDAs erlauben uns, deterministisch kontextfreie Sprachen einzuführen.

# Deterministisch kontextfreie Sprachen, DCFLs

# Deterministisch kontextfreie Sprachen

**Ziel:** Schränke die Definition von PDAs so ein, dass sie deterministisch sind, d.h. dass es stets nur höchstens eine mögliche Nachfolgekonfiguration gibt.

Ein **deterministischer Kellerautomat** (kurz: **DPDA**) ist ein PDA $A = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$, so dass für alle $q \in Q$, $a \in \Sigma$ und $X \in \Gamma$ gilt

$$|\delta(q, a, X)| + |\delta(q, \epsilon, X)| \leqslant 1.$$

$A$ akzeptiert mit Zuständen.

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

**Frage:** Warum erlauben wir hier $\epsilon$-Übergänge? Weil das unser Modell stärker macht. Die Sprache

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

wird von einem DPDA mit $\epsilon$-Übergängen akzeptiert, aber nicht von einem "Echtzeit-DPDA", der keine $\epsilon$-Übergänge nutzt.

**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 „leerem 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**.

- Beachte, dass jede reguläre Sprache von einem DPDA akzeptiert wird.

Eine Sprache $L$ ist

**deterministisch kontextfrei**,

d.h. $L \in$ DCFL, wenn $L$ von einem DPDA akzeptiert wird.

# Beispiele deterministisch kontextfreier Sprachen (1/2)

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

  **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

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

# Beispiele deterministisch kontextfreier Sprachen (2/2)

- Wohlgeformte Klammerausdrücke mit mehreren Klammertypen.

## Der Satz von Chomsky-Schützenberger:

Jede kontextfreie Sprache $L$ besitzt die Darstellung

$$L = h(D_m \cap R)$$

$D_m$ ist die Dyck-Sprache aller wohlgeformten Klammerausdrücke mit $m$ Klammertypen, $R$ eine reguläre Sprache und $h$ ein Homomorphismus.

DCFLs sind abgeschlossen unter dem Durchschnitt mit regulären Sprachen:

Zu jeder kontextfreien Sprache $L$ gibt eine Sprache $D \in DCFL$ mit $L = h(D)$.

# Deterministisch kontextfreie Sprachen: Abschluss-Eigenschaften

# Abschlusseigenschaften deterministisch kontextfreier Sprachen

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

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

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

**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**.

- $\epsilon$-Übergänge von $A$ machen das ganze etwas kompliziert.
- Details finden sich in Kapitel 8 des Buchs von Wegener bzw. im Abschnitt 4.7 des Buchs von Shallit.

(b) Lass DPDA und DFA „nebeneinander“ laufen.

# Nicht-Abschlusseigenschaften

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

- Die Klasse der deterministisch kontextfreien Sprachen ist **nicht** abgeschlossen unter **Durchschnittbildung**.

**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 deterministisch kontextfrei, aber $L_1 \cap L_2 = \{a^n b^n c^n : n \in \mathbb{N}\}$ ist noch nicht einmal kontextfrei.

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

- Die Klasse der deterministisch kontextfreien Sprachen ist **nicht** abgeschlossen unter **Vereinigung**,

Warum?

# Weitere Nicht-Abschlusseigenschaften

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.

**Beweis:** Übung!

# Fundamentale Eigenschaften

- Jede **reguläre** Sprache ist deterministisch kontextfrei.
- Jede deterministisch kontextfreie Sprache ist **eindeutig**.
- 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.

# 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 $a's$ mit $b$'s oder $a$'s mit $c$'s verglichen werden sollen.
  **Formaler Grund:**

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

  $L$ ist sogar inhärent mehrdeutig.

- $L = \{w \cdot w^R \mid w \in \{a, b\}^*\}$ 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.

# Einige Beispiele (2/2)

- $\{a^n b^m c^k \mid n, m, k \in \mathbb{N},\ n \neq k\}$ 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 Programmtext ein syntaktisch korrektes Programm der jeweiligen Programmiersprache ist, muss ein Parser das Wortproblem 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, sogenannte 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

$$A \to BC \text{ oder } A \to a$$

für $A, B, C \in V$ und $a \in \Sigma$), denn jede KFG $G$ kann in quadratischer Zeit in eine KFG $G'$ in Chomsky Normalform mit $L(G') = L(G) \setminus \{\varepsilon\}$ umgeformt werden.

## 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.

  Die Teilprobleme: Für alle $i, j$ mit $1 \leqslant i \leqslant j \leqslant n$ bestimme 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,
  - 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

$$s := j - i$$

  ist.

  - Für $s = 0$ wissen wir bereits, dass $V_{i,i} = \{A \in V : (A \to w_i) \in P\}$ ist.
  - Alle Mengen $V_{i,j}$ mit $j - i < s$ seien bereits bekannt $\Longrightarrow$
    Bestimme $V_{i,j}$ mit $j - i = s$.

# Eine Lösung der Teilprobleme (2/2)

**Ziel:** Bestimme $V_{i,j} := \left\{ A \in V : A \stackrel{*}{\Rightarrow} w_i \cdots w_j \right\}$ für $j - i = s$.
Alle Mengen $V_{i,j}$ mit $j - i < s$ sind bereits bekannt.

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

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

- Somit ist

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

$V_{i,j}$ kann in Zeit $O(|P| \cdot n)$ bestimmt werden.

# 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 $s = 1, \ldots, n-1$ :

(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.

**Laufzeit:**

- für (1): $O(|P|)$ Schritte
- für (3): jedes einzelne der $O(n^2)$ vielen $V_{i,j}$ benötigt $O(|P| \cdot n)$ Schritte;
- $\Longrightarrow$ die Gesamtlaufzeit beträgt $O(|P| \cdot n^3)$.

Der CYK-Algorithmus löst das Wortproblem.

Aber können wir damit auch einen Ableitungsbaum bestimmen?

JA! Wenn die Variable $A$ in die Menge $V_{i,j}$ eingefügt wird:

- 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, rechne den Ableitungsbaum für $w$ aus.

# Entscheidungsprobleme

# Entscheidungsprobleme

Nur wenige Entscheidungsprobleme für KFGs sind sogar effizient entscheidbar:

- Das Wortproblem, wie gerade gesehen, wird durch den CYK-Algorithmus gelöst.
- Das Leerheitsproblem ($L(G) \stackrel{?}{=} \emptyset$) ist effizient entscheidbar, weil
  - für jede Variable $X \in V$ festgestellt werden kann, ob $X$ *produktiv* ist, d.h. ob es eine Ableitung $X \stackrel{*}{\Rightarrow} w$ für irgendein Wort $w \in \Sigma^*$ gibt.
  - Wie geht das?
- Das Endlichkeitsproblem (Ist $L(G)$ endlich?) ist effizient lösbar: Übungsaufgabe.

Die Unentscheidbarkeit vieler Entscheidungsfragen für KFGs wird durch das Postsche Korrespondenzproblem erklärt.

# Das Postsche Korrespondenzproblem, ein SEHR schwieriges Dominospiel

# Das Postsche Korrespondenzproblem

Das Postsche Korrespondenzproblem (PKP)

*Eingabe:* Ein endliches Alphabet $\Sigma$, eine Zahl $k \in \mathbb{N}_{>0}$ und eine Folge von Wortpaaren $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$ mit $x_1, y_1, \ldots, x_k, y_k \in \Sigma^+$.

*Frage:* Gibt es ein $n \in \mathbb{N}_{>0}$ und Indizes $i_1, \ldots, i_n \in \{1, \ldots, k\}$, so dass $x_{i_1} x_{i_2} \cdots x_{i_n} = y_{i_1} y_{i_2} \cdots y_{i_n}$ ?

**Beispiel:** Das PKP mit Eingabe $\Sigma = \{0, 1\}$, $k = 3$ und

$$(x_1, y_1) = (1, 111), \qquad (x_2, y_2) = (10111, 10), \qquad (x_3, y_3) = (10, 0).$$

hat eine Lösung mit $n = 4$ und $i_1 = 2$, $i_2 = 1$, $i_3 = 1$, $i_4 = 3$, denn:

$$\begin{aligned} x_2 x_1 x_1 x_3 &= 10111\ 1\ \ 1\ 10 \\ y_2 y_1 y_1 y_3 &= 10\ \ 111\ 111\ 0. \end{aligned}$$

Das PKP lässt sich als ein Spiel mit hochgestellten Dominosteinen auffassen: Obere und untere Zeile müssen übereinstimmen!

# Wie schwierig ist das PKP?

Das PKP ist rekursiv aufzählbar.

Zeige, dass PKP nicht entscheidbar ist.

# Varianten des PKP

- Fixierter Startindex $i_1 = 1$:

Das modifizierte PKP (MPKP)

*Eingabe:* Ein endliches Alphabet $\Sigma$, eine Zahl $k \in \mathbb{N}_{>0}$ und eine Folge von Wortpaaren $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$ mit $x_1, y_1, \ldots, x_k, y_k \in \Sigma^+$.

*Frage:* Gibt es ein $n \in \mathbb{N}_{>0}$ und Indizes $i_1, \ldots, i_n \in \{1, \ldots, k\}$, so dass $i_1 = 1$ und $x_{i_1} x_{i_2} \cdots x_{i_n} = y_{i_1} y_{i_2} \cdots y_{i_n}$ ?

**Beispiel:** Hat das obige Beispiel eine Lösung für das MPKP?

- Fixiertes Alphabet $\Sigma$:

Das PKP über Alphabet $\Sigma$ ($\text{PKP}_\Sigma$)

*Eingabe:* Eine Zahl $k \in \mathbb{N}_{>0}$ und eine Folge von Wortpaaren $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$ mit $x_1, y_1, \ldots, x_k, y_k \in \Sigma^+$.

*Frage:* Gibt es ein $n \in \mathbb{N}_{>0}$ und Indizes $i_1, \ldots, i_n \in \{1, \ldots, k\}$, so dass $x_{i_1} x_{i_2} \cdots x_{i_n} = y_{i_1} y_{i_2} \cdots y_{i_n}$ ?

# Unentscheidbarkeit des PKP

**Satz:** $U \leqslant \text{MPKP} \leqslant \text{PKP} \leqslant \text{PKP}_{\{0,1\}}$.

PKP und seine Varianten MPKP und $\text{PKP}_{\{0,1\}}$ sind nicht entscheidbar.

**Beweis:** Nur für $\text{PKP} \leqslant \text{PKP}_{\{0,1\}}$:

- Für ein endliches Alphabet $\Sigma = \{a_1, \ldots, a_m\}$ wähle $h : \Sigma^* \to \{0,1\}^*$ mit

$$h(a_j) := 0\,1^j$$

für alle $j \in \{1, \ldots, m\}$ als Binärkodierung von $a_j$.

- Die Reduktion $f$ von PKP auf $\text{PKP}_{\{0,1\}}$ weist einer Eingabe

$$\Sigma,\ k,\ (x_1, y_1), \ldots, (x_k, y_k)$$

für's PKP die folgende Eingabe für's $\text{PKP}_{\{0,1\}}$ zu:

$$k,\ \big(h(x_1),\ h(y_1)\big), \ldots, \big(h(x_k),\ h(y_k)\big).$$

- Für alle $n \in \mathbb{N}_{>0}$ und alle $i_1, \ldots, i_n \in \{1, \ldots, k\}$ gilt:

$$x_{i_1} x_{i_2} \cdots x_{i_n} = y_{i_1} y_{i_2} \cdots y_{i_n} \iff h(x_{i_1}) h(x_{i_2}) \cdots h(x_{i_n}) = h(y_{i_1}) h(y_{i_2}) \cdots h(y_{i_n}).$$

Die Reduktion $f$ ist berechenbar. Somit ist $f$ eine Reduktion von PKP auf $\text{PKP}_{\{0,1\}}$.

# Unentscheidbarkeit des PKP: $U \leqslant$ MPKP (1/4)

Wir suchen eine Transformation $f$, so dass

$u \in U$, d.h. $u = \langle M \rangle w$ für eine TM $M$, die Eingabe $w$ akzeptiert $\iff$ das MPKP $f(u)$ besitzt eine Lösung.

**Leichter Fall: u ist keine Repräsentation einer TM.**

Dann sei $f(w)$ eine Eingabe für's MPKP, die keine Lösung besitzt, z.B., $\Sigma = \{0, 1\}$, $k = 1$, $x_1 = 0$, $y_1 = 1$.

**Schwieriger Fall: u $= \langle$M$\rangle$w für eine TM M. Idee:**

- Repräsentiere eine Berechnung von $M$ auf $w$ durch eine Folge von Konfigurationen. Die Konfiguration $uqv$ repräsentiert die Situation bei der die TM
  - im Zustand $q$ ist,
  - die Bandinschrift $uv$ ist, und
  - der Kopf auf dem ersten Symbol von $v$ steht.
- Das „Dominospiel" $f(\langle M \rangle w)$ für's MPKP ist **lösbar**

$\overset{!}{\iff}$

die $x$-Folge –also die obere Zeile– baut die Folge der Konfigurationen einer **akzeptierenden** Berechnung nach.

# Unentscheidbarkeit des PKP: $U \leqslant$ MPKP (2/4)

Wähle Alphabet und Regeln (= Wortpaare) so, dass ein **erfolgreiches** Dominospiel die Konfigurationen einer **akzeptierenden** Berechnung von $M$ auf $w$ nachbaut.

- Das MPKP $f(\langle M \rangle w)$ besitzt das Alphabet $\Gamma \dot\cup Q \dot\cup \{\#\}$.
  Das Symbol # dient als Trennsymbol zwischen einzelnen Konfigurationen.
- Die Startkonfiguration ist $q_0 w$: Wähle $x_1 = \#$ und $y_1 = \#q_0 w\#$.
  - Anfänglich hinkt die obere Zeile hinterher.
    Wir lassen die obere Zeile nur am Ende der Berechnung –wenn überhaupt– aufholen.

!!! Zwischendurch: MPKP darf nur dann lösbar sein, wenn die obere Zeile,

der unteren hinterher hechelnd,

die eindeutig bestimmte Konfigurationenfolge der deterministischen Maschine $M$ auf $w$ nachbaut.

Wir müssen Regeln finden, die (!!!) garantieren.

# Unentscheidbarkeit des PKP: $U \leqslant$ MPKP (3/4)

- Die untere Zeile sollte die $i+1$te Konfiguration von $M$ bauen, wenn die obere Zeile die $i$te Konfiguration nachbaut.
  - **Rechtsbewegung**:
    falls $\delta(q,a) = (q', a', \texttt{rechts})$, $(qa, a'q')$
  - **Linksbewegung**:
    falls $\delta(q,a) = (q', a', \texttt{links})$, für jedes $b \in \Gamma$ $(bqa, q'ba')$.
- Wie können Konfigurationen „aufgefüllt werden“?
  - Wähle die **Kopierregeln** $(a,a)$ mit $a \in \Gamma$ sowie die Regel $(\#,\#)$.

Zu jedem Zeitpunkt muss es

genau eine anwendbare Regel geben,

die einen zukünftigen Match zwischen oberer und unterer Zeile nicht ausschließt.

# Unentscheidbarkeit des PKP: $U \leqslant$ MPKP (4/4)

$M$ hält auf Eingabe $w$ $\overset{?}{\iff}$
$f(\langle M \rangle w)$ besitzt eine Lösung mit Starttupel $(x_1, y_1) = (\#, \#q_0 w \#)$.

Leider ist die Behauptung **falsch**.

- Die obere Zeile kann nicht aufholen.
- Wenn die Maschine genügend lange nach rechts läuft, liest sie zum ersten Mal ein Blank: Füge in diesem Fall die Regel $(\#, \mathtt{B}\#)$ hinzu.
- Und wenn die Maschine genügend lange nach links läuft?

  **Forderung**: $M$ verlässt nie ihren Eingabebereich nach links.
- Wir lassen die obere Zeile aufholen, wenn die untere Zeile einen akzeptierenden Zustand erreicht:
  - **Forderung**: $M$ wechselt nur dann in einen Zustand aus $F$, wenn sie unmittelbar danach hält.
  - Füge die „Aufholregeln“ $(aq, q)$ und $(qa, q)$ für jeden Zustand $q \in F$ hinzu. Lass die obere Zeile langsam aufholen, indem eine Aufholregel auf eine Reihe von Kopierregeln folgt.
  - Aber dann fehlt der oberen Zeile ein Begrenzer: Füge $(q\#\#, \#)$ hinzu.

# Konsequenzen der Unentscheidbarkeit von PKP

Die folgenden Probleme sind für kontextfreie Grammatiken $G$, $G_1$ und $G_2$ unentscheidbar:

(a) Ist $L(G_1) \cap L(G_2) \neq \emptyset$? ..... (Leerer Schnitt)

(b) Ist $G$ eindeutig? ..... (Eindeutigkeit)

(c) Ist $L(G) = \Sigma^*$? ..... (Universalität)

- Ist $L(G_1) = L(G_2)$? ..... (Äquivalenz)
- Ist $L(G_1) \subseteq L(G_2)$? ..... (Subsumption)

(d) Ist $L(G_1) \cap L(G_2)$ kontextfrei? ..... (Kontextfreier Schnitt)

# Ist $L(G_1) \cap L(G_2) \neq \emptyset$?

Löse das PKP $k, (x_1, y_1), \ldots, (x_k, y_k)$ mit Hilfe des Leerheitsproblems.

Wir verwenden die Sprachen

$$\begin{aligned} L(G_1) &= \{x_{i_1} \cdots x_{i_n} \$ y_{i_n}^{\text{reverse}} \cdots y_{i_1}^{\text{reverse}} \mid n \in \mathbb{N}\ i_1, \ldots, i_n \in \{1, \ldots, k\}\}, \\ L(G_2) &= \{w \$ w^{\text{reverse}} \mid w \in \{0,1\}^* \}. \end{aligned}$$

Beachte, dass $L(G_1)$ kontextfrei ist: Benutze die Produktionen

$$S \rightarrow x_i\, S\, y_i^{\text{reverse}} \mid x_i\, \$\, y_i^{\text{reverse}} \quad \text{für } i = 1, \ldots, k.$$

Das PKP ist genau dann lösbar, wenn $L(G_1) \cap L(G_2) \neq \emptyset$ gilt.

# Ist $G$ eindeutig ?

Löse das PKP $k, (x_1, y_1), \ldots, (x_k, y_k)$ mit Hilfe des Eindeutigkeitsproblems.

Wir verwenden das Alphabet $\Sigma = \{0, 1, \overline{1}, \overline{2}, \ldots, \overline{k}\}$ und wählen die kontextfreie Grammatik $G$ mit den Produktionen

$$\begin{aligned} S &\to A \mid B \\ A &\to x_1 A\overline{1} \mid x_2 A\overline{2} \mid \cdots \mid x_k A\overline{k} \mid \varepsilon \\ B &\to y_1 B\overline{1} \mid y_2 B\overline{2} \mid \cdots \mid y_k B\overline{k} \mid \varepsilon. \end{aligned}$$

Das PKP ist genau dann nicht lösbar, wenn $G$ eindeutig ist.

# Ist $L(G) = \Sigma^*$ ?

Löse das PKP $k, (x_1, y_1), \ldots, (x_k, y_k)$ mit Hilfe des Universalitätsproblems.

- Setze

$$L_1 = \{ w\$w^{\text{reverse}} \mid w \in \{0,1\}^* \},$$

wobei aber das Zeichen # an beliebiger Stelle in beliebiger Anzahl vorkommen kann und

$$L_2 = \{ x_{i_1} \# \cdots \# x_{i_n} \$ \, y_{i_n}^{\text{reverse}} \# \cdots \# y_{i_1}^{\text{reverse}} \mid n \in \mathbb{N},\ i_1, \ldots, i_n \in \{1, \ldots, k\} \}.$$

- $L_1$ und $L_2$ sind deterministisch kontextfrei. Also sind sowohl $\{0,1,\#,\$\}^* \setminus L_1$ wie auch $\{0,1,\#,\$\}^* \setminus L_2$ kontextfrei und damit auch

$$\begin{aligned} L &= (\{0,1,\#,\$\}^* \setminus L_1) \cup (\{0,1,\#,\$\}^* \setminus L_2) \\ &= \{0,1,\#,\$\}^* \setminus (L_1 \cap L_2) \end{aligned}$$

Das PKP ist genau dann unlösbar, wenn $L = \{0,1,\#,\$\}^*$.

# Zusammenfassung: Die wichtigsten Konzepte (1/3)

- Kontextfreie Grammatiken definieren die Syntax moderner Programmiersprachen.
  - Die Konstruktion eines Ableitungsbaums ist das wesentliche Ziel der Syntaxanalyse.
  - Der CYK-Algorithmus erlaubt eine „halbwegs effiziente“ Lösung des Wortproblems wie auch eine „halbwegs 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 deterministisch kontextfreien Sprachen kennen gelernt.
  - Jede deterministisch kontextfreie Sprache ist eindeutig.
  - Aber es gibt Sprachen, die eindeutig, aber nicht deterministisch kontextfrei sind. Beispiel: $\{w \cdot w^R : w \in \{a, b\}^*\}$.
  - Das Wortproblem ist in Linearzeit lösbar.

# Zusammenfassung: Die wichtigsten Resultate (2/3)

- Mit dem Pumping Lemma oder mit Ogden's Lemma oder den Abschluss-eigenschaften der Klasse der kontextfreien Sprachen können wir nachweisen, dass (bestimmte) Sprachen NICHT kontextfrei sind.
- Kontextfreie Grammatiken und nichtdeterministische Kellerautomaten definieren dieselbe Sprachenklasse. (Tripelkonstruktion)
  - Kellerautomaten haben uns erlaubt, deterministisch kontextfreie Sprachen einzuführen.
- 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.

# Zusammenfassung: Die wichtigsten Resultate (3/3)

- Nur wenige Entscheidungsprobleme für kontextfreie Sprachen sind effizient lösbar: Dazu gehören
  - das Wortproblem,
  - das Leerheitsproblem $L(G) \stackrel{?}{=} \emptyset$,
  - das Endlichkeitsproblem: Ist $L(G)$ endlich?
- Viele wichtige Entscheidungsprobleme für KFGs sind nicht entscheidbar, weil das Postsche Korrespondenzproblem auf sie reduzierbar ist:
  - $L(G_1) \cap L(G_2) \stackrel{?}{=} \emptyset$,
  - das Universalitätsproblem $L(G) \stackrel{?}{=} \Sigma^*$,
  - das Eindeutigkeitsproblem: Ist die Grammatik $G$ eindeutig?
  - das Schnittproblem: Ist $L(G_1) \cap L(G_2)$ kontextfrei?

  Das Äquivalenzproblem für Finite State Transducer ist ebenfalls unentscheidbar.