# Theoretische Informatik III (T3INF2002)

Formale Sprachen und Automaten | Einführung Compilerbau

Vorlesung im Wintersemester 2022/23

# Formale Sprachen und Automaten

- Chomsky-Hierarchie
- Reguläre Sprachen

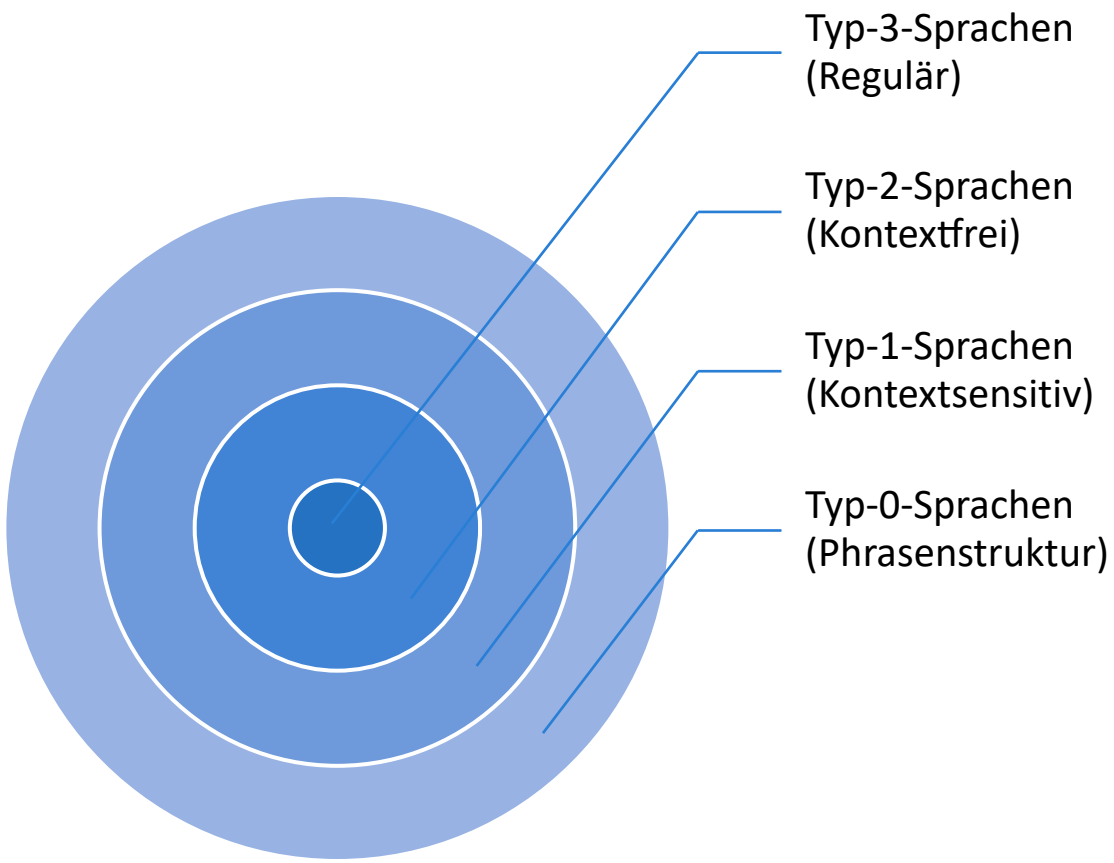
# Chomsky-Hierarchie

# Chomsky-Hierarchie

- Formale Grammatiken sind ein mächtiges Werkzeug für die Erzeugung unterschiedlichster Sprachen.
- Spanne reicht von einelementigen Wortmengen bis hin zu komplexen Sprachgebilden, die dem gesprochenen Wort gleichen können.
- Struktur der Produktionen einer Grammatik *G* hat dabei einen großen Einfluss auf die Eigenschaften der erzeugten Sprache L (*G*).

-> 1957 veröffentlichte der amerikanische Sprachwissenschaftler Noam Chomsky ein Regelwerk, mit dessen Hilfe formale Grammatiken in vier Klassen eingeteilt werden können.

# Chomsky-Hierarchie




Chomsky-Hierarchie teilt die Menge der formalen Sprachen in vier Typklassen ein.

Eine Sprache *L* ist eine Typ-*n*-Sprache, wenn eine Typ-*n*-Grammatik existiert, die *L* erzeugt.

Zwischen den Sprachklassen besteht eine echte Inklusionsbeziehung,
d. h., für alle $n$ mit $0 \leq n < 3$ gilt $L_n \supset L_{n+1}$ und $L_n \neq L_{n+1}$.

# Chomsky-Hierarchie Typ-0

**Phrasenstrukturgrammatiken(Typ-0-Grammatiken)**

Jede Grammatik ist per Definition immer auch eine Typ-0-Grammatik. Insbesondere unterliegt die Struktur der Produktionen keinen weiteren Einschränkungen.

$L_{C0}$ := {ω|ω codiert eine terminierende Turing-Maschine} ist eine Typ-0-Sprache, aber keine Typ-1-Sprache.

# Chomsky-Hierarchie Typ-1

**Kontextsensitive Grammatiken(Typ-1-Grammatiken)**

Eine Grammatik heißt *kontextsensitiv*, wenn jede Produktionsregel $l \rightarrow r$ entweder die Beziehung $|r| \geq |l|$ erfüllt oder die Form $S \rightarrow \varepsilon$ aufweist. Ist die Regel $S \rightarrow \varepsilon$ enthalten, so darf $S$ in keiner anderen rechten Seite einer Regel vorkommen.

$L_{C1} := \{a^n b^n c^n \mid n \in \mathbb{N}^+\}$ ist eine Typ-1-Sprache, aber keine Typ-2-Sprache.

*Regeln:*
*1) S -> aSBc*
*2) S -> abc*
*3) cB -> Bc*
*4) bB -> bb*

# Chomsky-Hierarchie Typ-2

**Kontextfreie Grammatiken (Typ-2-Grammatiken)**

Typ-2-Grammatiken sind dadurch charakterisiert, dass die linke Seite einer Produktionsregel ausschließlich aus einer einzigen Variablen besteht. Für alle Produktionen $l \rightarrow r$ gilt also $l \in V$.

$L_{C2} := \{a^n b^n \mid n \in \mathbb{N}^+\}$ ist eine Typ-2-Sprache, aber keine Typ-3-Sprache.

Regeln:
*1) S → aSb*
*2) S → ab*

# Chomsky-Hierarchie Typ-3

**Reguläre Grammatiken (Typ-3-Grammatiken)**

Reguläre Grammatiken sind kontextfrei und besitzen die zusätzliche Eigenschaft, dass die rechte Seite einer Produktion entweder aus dem leeren Wort ε oder einem Terminalsymbol, gefolgt von einem Nonterminal, besteht. Formal gesprochen besitzt jede Produktion die Form $l \rightarrow r$ mit $l \in V$ und $r \in \{\varepsilon\} \cup \Sigma V$.

$L_{C3} := \{ (ab)^n \mid n \in \mathbb{N}^+ \}$ ist eine Typ-3-Sprache.

# Beispiel für reguläre Sprache (Typ 3)

V = { S, A, B }

Σ = { a, b }

P={ 1. S -> bS,
2. S -> aA,
3. A -> bS,
4. A -> aB,
5. A -> a,
6. B -> bB,
7. B -> aB,
8. B -> b,
9. B -> a }

Die Grammatik ist vom Typ 3, da…

…auf der linken Seite bei jeder Regel nur eine Variable steht.

… auf der rechten Seite nur ein einziges Terminalsymbol (Regeln 5, 8, 9) oder ein Terminalsymbol gefolgt von einer Variablen (Regeln 1, 2, 3, 4, 6, 7) steht.

# Abschlusseigenschaften

- Bei der Untersuchung der verschiedenen Sprachklassen spielen die Abschlusseigenschaften eine wichtige Rolle.
- Diese beantworten die Frage, ob die Verknüpfung zweier $L_n$-Sprachen zu einer Sprache führt, die wiederum in der Sprachklasse $L_n$ liegt oder aus dieser herausfällt.

- Vereinigung
  - Ist mit $L_1, L_2 \in Ln$ auch die Sprache $L_1 \cup L_2 \in L_n$?
- Durchschnitt
  - Ist mit $L_1, L_2 \in L_n$ auch die Sprache $L_1 \cap L_2 \in L_n$?
- Komplement
  - Ist mit $L \in L_n$ auch die Sprache $\Sigma^* \setminus L \in L_n$?
- Konkatenation
  - Ist mit $L_1, L_2 \in L_n$ auch die Sprache $L_1 L_2 \in L_n$?
- Kleene'sche Hülle
  - Ist mit $L \in L_n$ auch die Sprache $L^* \in L_n$?

# Syntaxbaum

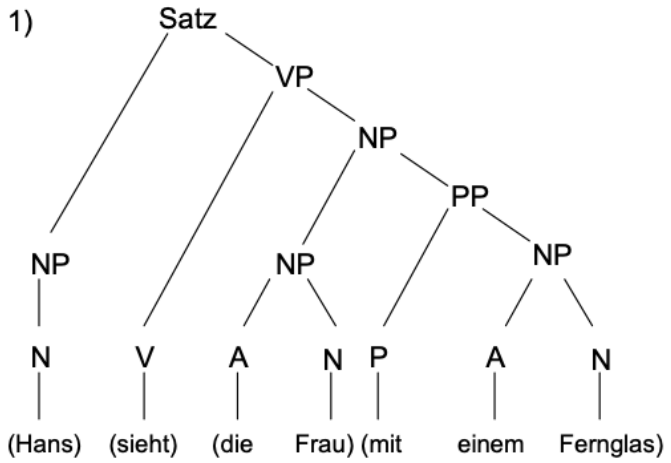
NP = Nominalphrase
VP = Verbalphrase
N = Nomen
A = Artikel
PP = Präpositionalphrase
V = Verb
P = Präposition

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Satz | → | NP VP | P | → | mit | V | → | sieht |
| NP | → | NP PP | P | → | in | V | → | geht |
| NP | → | N | P | → | auf | A | → | der |
| NP | → | A N | N | → | Hans | A | → | die |
| VP | → | V | N | → | Frau | A | → | das |
| VP | → | V NP | N | → | Fernglas | A | → | einem |
| VP | → | VP PP | N | → | Park | | | |
| PP | → | P NP | | | | | | |

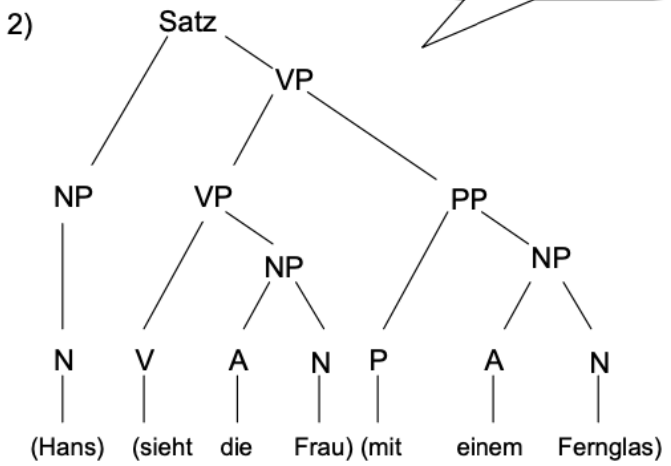
➢ Syntaxbaum zu „Hans sieht die Frau mit einem Fernglas“

Mehrdeutige Grammatik

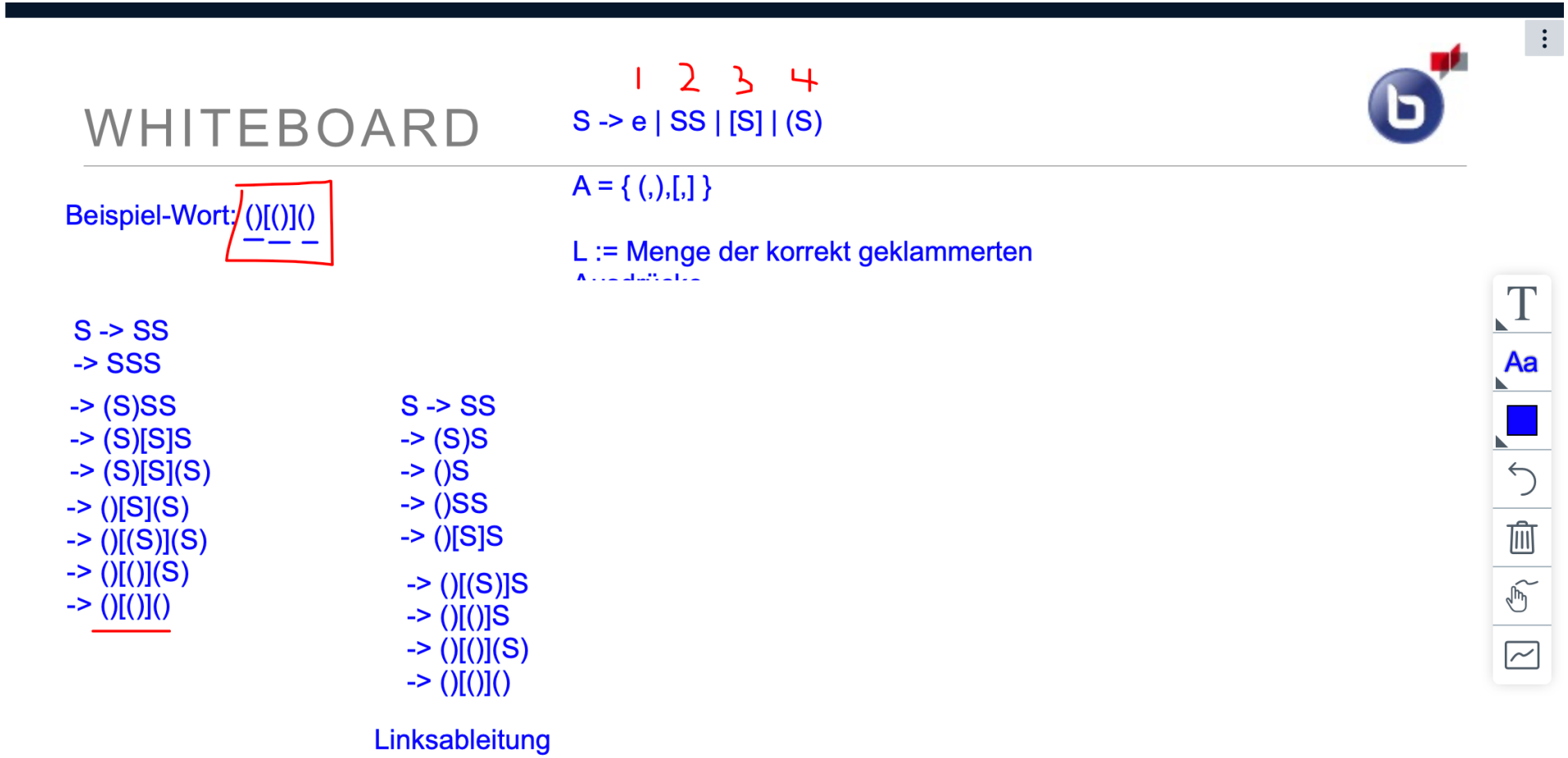
1)




2)



# Übung: Syntaxbaum für das Dyck-Wort ()[()]()



# Übung: Syntaxbaum für das Dyck-Wort ()[()]()

1. Die erste Ableitungssequenz besitzt die Eigenschaft, dass in jedem Schritt das am weitesten links stehende Nonterminal ersetzt wird. Eine solche Sequenz heißt Linksableitung.
2. Die zweite Ableitungssequenz wird so konstruiert, dass in jedem Schritt das am weitesten rechts stehende Nonterminal ersetzt wird. Eine solche Sequenz heißt Rechtsableitung.
3. Die dritte Ableitungssequenz ist ebenfalls eine Linksableitung, da genau wie im ersten Beispiel in jedem Schritt das am weitesten links stehende Nonterminal ersetzt wird. Der Unterschied zwischen beiden Sequenzen besteht alleine in der Wahl der Produktionen, die auf das entsprechende Nonterminal angewendet werden.

# Übung: Erstellen Sie den Syntaxbaum für das Dyck-Wort ()[()]()

# Reguläre Sprachen

- Menge der regulären Sprachen (Typ-3-Sprachen) ist die kleinste Sprachklasse in der Chomsky-Hierarchie.
- Obwohl diese über eine vergleichsweise einfache Struktur verfügen, nehmen die Sprachen einen prominenten Platz in der Informatik ein.
- Viele Datenformate sind regulär
- Suchmuster, die uns z. B. auf der Shell-Ebene das Auffinden von Dateien erlauben, sind ebenfalls reguläre Ausdrücke.

# Reguläre Sprachen

- Produktionen unterliegen einer regulären Grammatik erheblichen Einschränkungen.
- Regeln erlauben, die linke Seite aus einem Nonterminal und die rechte Seite entweder aus dem leeren Wort ε oder einem Terminalzeichen, gefolgt von einem Nonterminal, zu bilden.

# Reguläre Sprachen

Unter Beachtung dieser Einschränkungen lässt sich die Sprache

$$L_{C3} = \{(ab)^n \mid n \in N^+\}$$

mit der Grammatik

$$G = (\{S, B, C\}, \{a, b\}, P, S)$$

erzeugen.

*Regeln:*

$S \rightarrow aB$

$B \rightarrow bC$

$C \rightarrow \varepsilon \mid aB$

# Reguläre Sprachen

$L_{C3} = \{(ab)^n \mid n \in N^+\}$

- Dies zeigt, wie sich die Wörter *abab* und *ababab* aus dem Startsymbol ableiten lassen.
- Ein Blick auf die Ableitungssequenzen zeigt die Rolle des Nonterminals B auf.

$$S \rightarrow aB$$
$$B \rightarrow bC$$
$$C \rightarrow \varepsilon \mid aB$$

- Es tritt immer dann auf, wenn zuletzt ein *a* erzeugt wurde, und stellt sicher, dass die Zeichenkette mit einem *b* fortgesetzt wird.

# Übung: Syntaxbaum regulärer Grammatiken erzeugte

- Bilden Sie die Ableitung von *ababab* aus der Grammatik
  G = ({S, B, C}, {a, b}, P, S)
- Erstellen Sie den Syntaxbaum aus den Produktionsregeln

$$S \rightarrow aB$$
$$B \rightarrow bC$$
$$C \rightarrow \varepsilon \mid aB$$

# Übung: Syntaxbaum regulärer Grammatiken erzeugte

- Bilden Sie die Ableitung von *ababab* aus der Grammatik
  G = ({S, B, C}, {a, b}, P, S)
- Erstellen Sie den Syntaxbaum aus den Produktionsregeln

$$S \rightarrow aB$$
$$B \rightarrow bC$$
$$C \rightarrow \varepsilon \mid aB$$

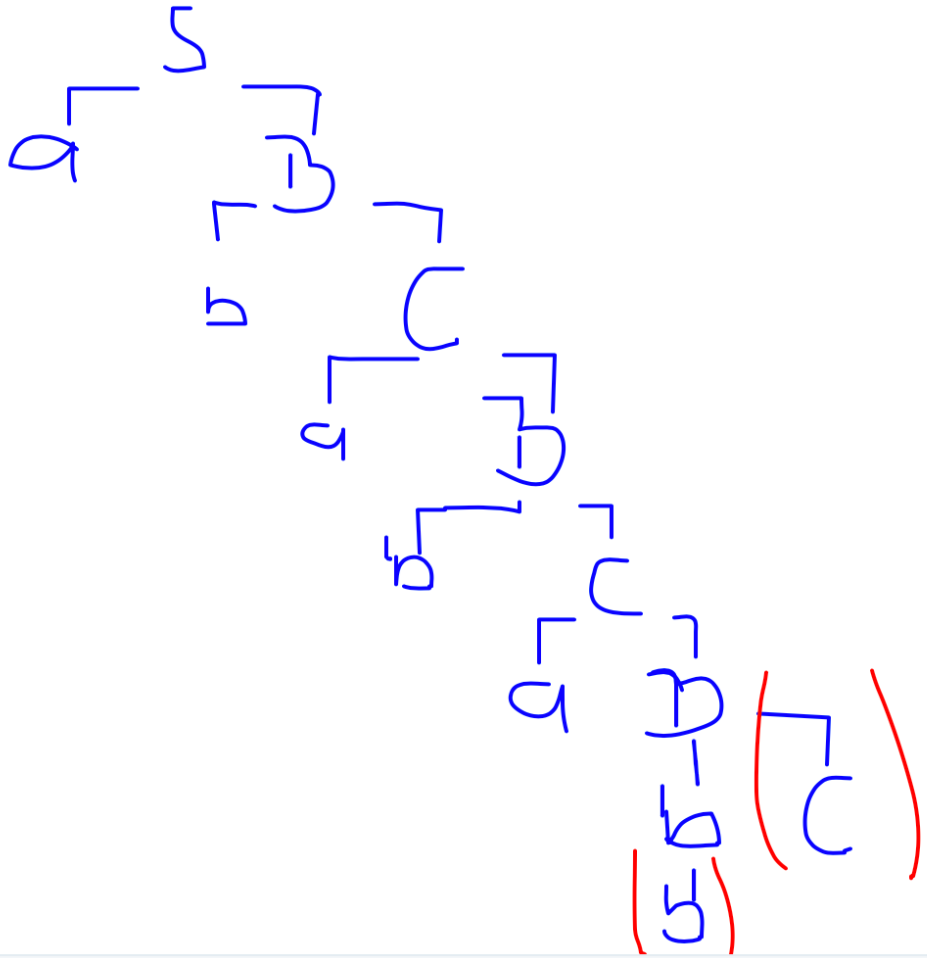
S -> abB
B -> e | abB
abB
-> e = ab
-> ababB

Rechte Seite besitzt ein Terminalzeichen gefolgt von einem Nonterminal

Rechte Seite besitzt zwei Terminalzeichen gefolgt von einem Nonterminal

# Übung: Syntaxbaum regulärer Grammatiken erzeugte

S -> aB
-> abC
-> ababB
-> abababC
-> abababC
-> ababab



# Reguläre Sprachen

- Die einfache Struktur der Produktionen einer regulären Grammatik wirkt sich unmittelbar auf das Erscheinungsbild der entstehenden Syntaxbäume aus.
- Da die rechte Seite einer Produktion aus maximal zwei Zeichen besteht und das erste immer aus der Menge der Terminalzeichen stammt, weisen die entstehenden Syntaxbäume die Struktur einer linearen Kette auf.
- Aufgrund dieser Eigenschaft werden reguläre Grammatiken auch als **rechtslineare** Grammatiken bezeichnet.

# Reguläre Sprachen

- Die Linearitätseigenschaft sorgt dafür, dass Wörter in regulären Grammatiken immer nach dem gleichen Prinzip erzeugt werden.
- Ausgehend von dem leeren Wort in Form des Startsymbols fügt jede Produktion der Form $A \rightarrow \sigma\ B$ das Zeichen σ an und ersetzt das aktuell vorhandene Nonterminal $A$ durch das neue Symbol $B$.
- Hierdurch wird die erzeugte Zeichenkette mit jedem Ableitungsschritt um ein einzelnes Zeichen verlängert und nach rechts durch ein wechselndes Nonterminal begrenzt.

# Reguläre Sprachen

- Das Nonterminal fungiert als Zustandsmerker und reglementiert die Anzahl der im nächsten Schritt anwendbaren Produktionen.
- Epsilon-Regeln der Form $A \rightarrow \varepsilon$ spielen ebenfalls eine entscheidende Rolle.
- Wird eine von ihnen angewendet, so verschwindet das Nonterminal aus der erzeugten Zeichenkette und der Produktionsprozess kommt zum Erliegen.

Produktionsmenge $P$
$S \rightarrow aB$
$B \rightarrow bC$
$C \rightarrow \varepsilon | aB$

# Reguläre Sprachen

- Bei manchen Anwendungsfällen ist es wünschenswert, möglichst viele Regeln der Form $l \rightarrow \varepsilon$ aus der Menge der Produktionen zu entfernen.
- In der Tat kann auf die meisten Epsilon-Regeln verzichten, indem man Produktionen zulässt, deren rechte Seiten aus einem einzigen Terminalzeichen bestehen.
- Im Beispiel lässt sich die Epsilon-Regel $C \rightarrow \varepsilon$ eliminieren, indem die Grammatik durch die zusätzliche Regel $B \rightarrow b$ erweitert wird.

$S \rightarrow aB$
$B \rightarrow b$
$B \rightarrow bC$
$C \rightarrow aB$

# Reguläre Sprachen

- In der entstandenen Form besteht die rechte Seite einer Produktion nur noch aus einem isolierten Terminalzeichen oder einem Terminalzeichen, gefolgt von einem Nonterminal.
- Würde man auf der rechten Seite mehr als ein Terminalzeichen zulassen, so ließe sich die Beispielgrammatik noch weiter vereinfachen:

$$S \rightarrow aB$$
$$B \rightarrow b$$
$$B \rightarrow baB$$

# Reguläre Sprachen

- Umformung kann auf beliebige Grammatiken angewendet werden und auf diese Weise fast alle Epsilon-Regeln nach und nach eliminieren.
- Die einzige Ausnahme bildet die Regel $S \rightarrow \varepsilon$
- Würden man diese – falls vorhanden – aus der Menge der Produktionen entfernen, so ließe sich das leere Wort ε nicht mehr ableiten.

# Reguläre Sprachen

- Zu Beginn der Vorlesung wurden mit dem ***Wortproblem***, dem ***Leerheitsproblem***, dem ***Endlichkeitsproblem*** und dem ***Äquivalenzproblem*** vier wichtige Fragestellungen für die Untersuchung formaler Sprachen eingeführt.
- Alle vier sind für reguläre Sprachen entscheidbar, d. h., es existiert ein Verfahren, das für alle Eingaben feststellt, ob die betreffende Eigenschaft zutrifft oder nicht.

# Entscheidungsprobleme regulärer Sprachen

| Problem | Eingabe | Fragestellung | Entscheidbar? |
|---|---|---|---|
| Wortproblem | Sprache L, Wort $\omega \in \Sigma^*$ | Ist $\omega \in L$? | Ja |
| Leerheitsproblem | Sprache L | Ist $L = \emptyset$ ? | Ja |
| Endlichkeitsproblem | Sprache L | Ist $\|L\| < \infty$? | Ja |
| Äquivalenzproblem | Sprachen $L_1$ und $L_2$ | Ist $L_1 = L_2$? | Ja |

# Abschlusseigenschaften regulärer Sprachen

| Operation | Eingabe | Fragestellung | Erfüllt? |
|---|---|---|---|
| Vereinigung | Sprache $L_1, L_2 \in L_3$ | Ist $L_1 \cup L_2 \in L_3$? | Ja |
| Schnitt | Sprache $L_1, L_2 \in L_3$ | Ist $L_1 \cap L_2 \in L_3$? | Ja |
| Komplement | Sprache $L \in L_3$ | Ist $\Sigma^* \backslash L \in L_3$? | Ja |
| Produkt | Sprache $L_1, L_2 \in L_3$ | Ist $L_1 L_2 \in L_3$? | Ja |

# Pumping-Lemma für reguläre Sprachen

# Pumping-Lemma für reguläre Sprachen

$S \rightarrow aB$
$B \rightarrow b$
$B \rightarrow bC$
$C \rightarrow aB$

- Sieht man von der letzten Regelanwendung ab, so wird in jedem Ableitungsschritt ein Terminalzeichen und ein neues Nonterminal erzeugt.
- Das Nonterminal steht immer an letzter Stelle und begrenzt die Auswahl der im nächsten Schritt anwendbaren Regeln.
- Es wirkt wie ein Zustandsspeicher, der einen Rückschluss auf die bisher erzeugten Zeichen erlaubt.
- Da die Menge der Nonterminale endlich ist, lassen sich während der Erzeugung eines Wortes nur endlich viele Zustände unterscheiden.
- Hierin liegt eine der grundlegenden Limitierungen regulärer Sprachen.

# Ableitungssequenz regulärer Sprachen

Betrachtet man eine Ableitungssequenz, die mehr Ableitungsschritte enthält als Nonterminale zur Verfügung stehen, so muss mindestens ein Nonterminal mehrfach auftauchen. Bezeichnet man dieses Nonterminalzeichen mit A, so besitzt die Ableitungssequenz die folgende Form:

$$
\begin{aligned}
& \dots \\
\Rightarrow \quad & \underbrace{\sigma_1 \sigma_2 \dots \sigma_i}_{u} A \\
\Rightarrow \quad & \underbrace{\sigma_1 \sigma_2 \dots \sigma_i}_{u} \underbrace{\sigma_{i+1} \sigma_{i+2} \dots \sigma_j}_{v} A \\
\Rightarrow \quad & \underbrace{\sigma_1 \sigma_2 \dots \sigma_i}_{u} \underbrace{\sigma_{i+1} \sigma_{i+2} \dots \sigma_j}_{v} \underbrace{\sigma_{j+1} \sigma_{j+2} \dots \sigma_k}_{w}
\end{aligned}
$$

# Ableitungssequenz regulärer Sprachen

- Die Ableitungssequenz zeigt, dass es möglich ist, aus dem Nonterminal A die Zeichenkette

$$vA = \sigma_{i+1}\sigma_{i+2} \ldots \sigma_j A$$

abzuleiten, die mindestens ein Terminalzeichen enthält ($|v| \geq 1$).

- Da die erzeugte Sequenz erneut mit dem Nonterminal A endet, lassen sich zusätzlich die Sequenzen ableiten

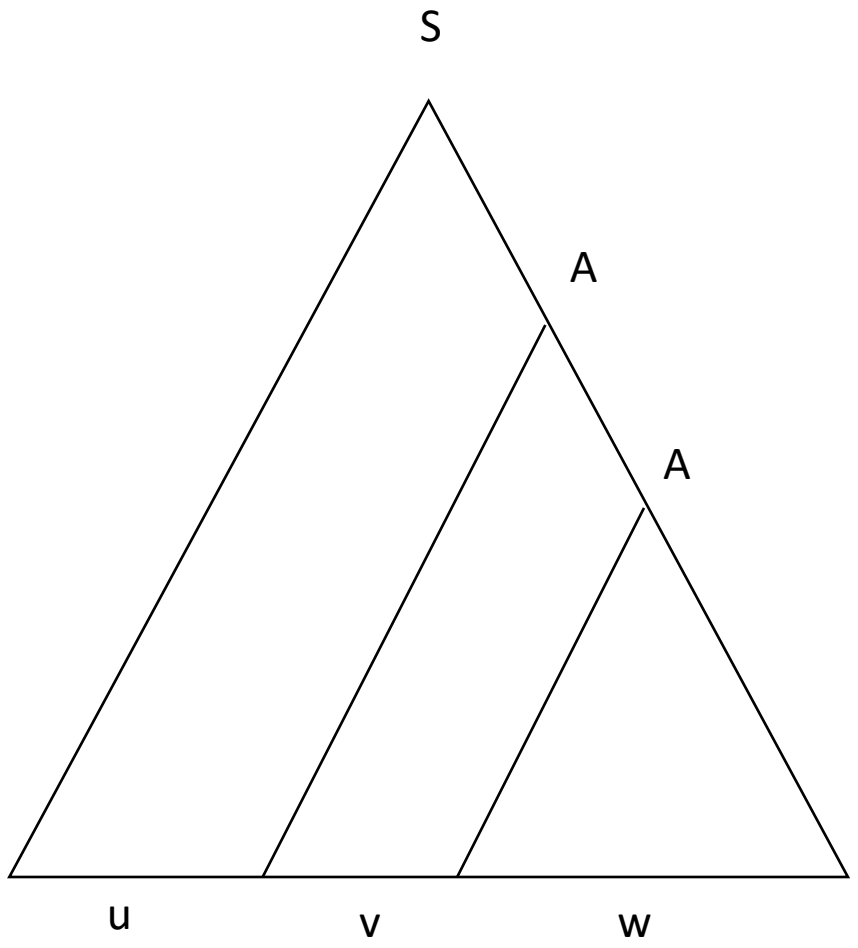
$$v^2A = \sigma_{i+1}\sigma_{i+2} \ldots \sigma_j\sigma_{i+1}\sigma_{i+2} \ldots \sigma_j A$$

$$v^3A = \sigma_{i+1}\sigma_{i+2} \ldots \sigma_j\sigma_{i+1}\sigma_{i+2} \ldots \sigma_j\sigma_{i+1}\sigma_{i+2} \ldots \sigma_j A$$

$$\ldots = \ldots$$

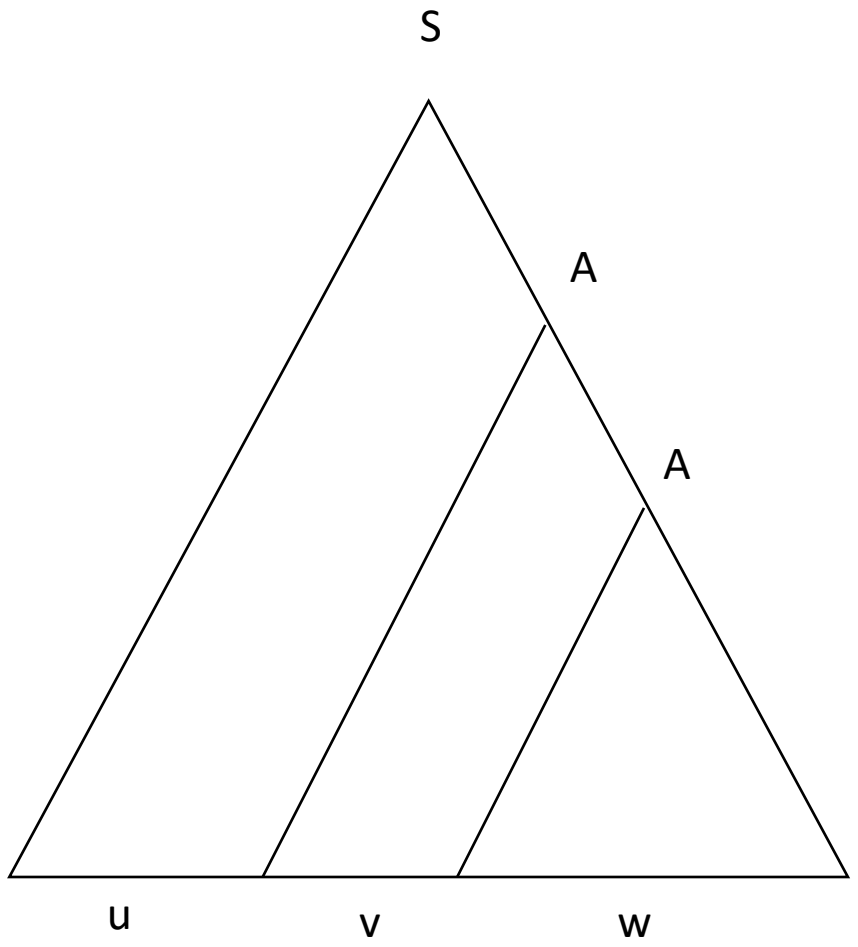
# Ableitungssequenz regulärer Sprachen

- Das zeigt, dass sich…
  - zum einen jedes hinreichend lange Wort einer regulären Sprache in der Form *uvw* ausdrücken lässt.
  - zum anderen neben dem Wort *uvw* auch die Wörter $uv^iw$ für alle $i \in \mathbb{N}$ in der Sprache enthalten sein müssen.
- Das ist die Kernaussage des Pumping-Lemmas für reguläre Sprachen.



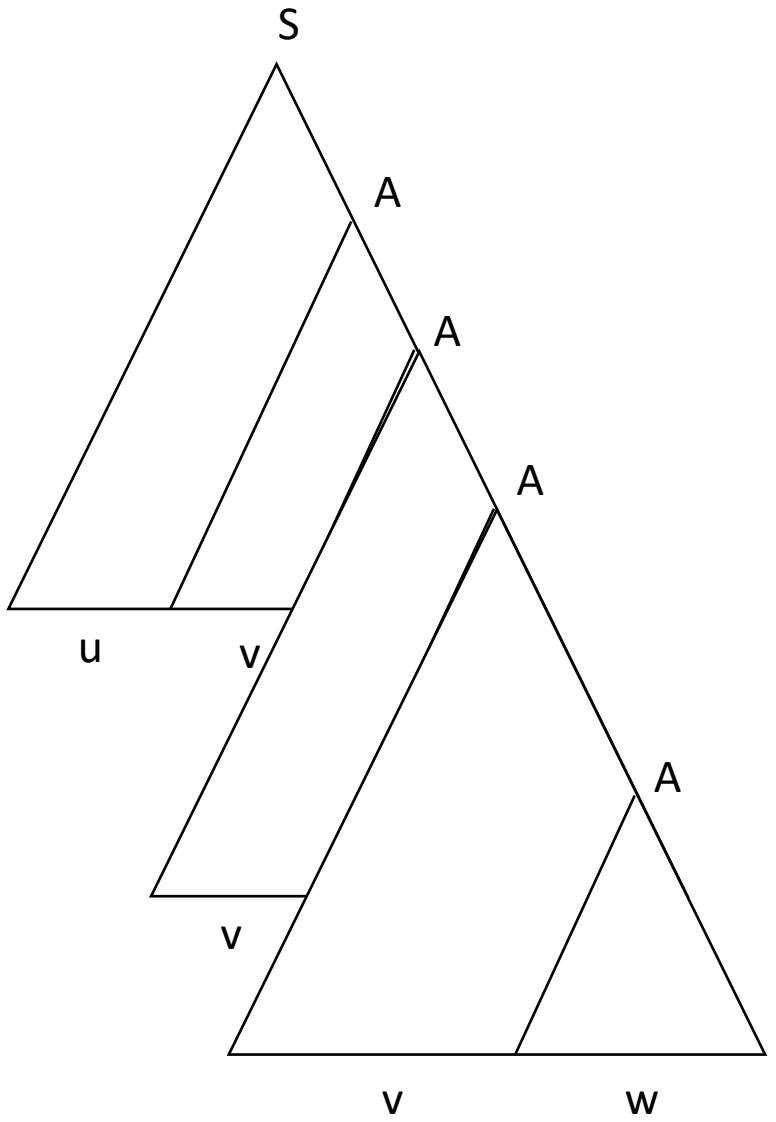
# Wortstruktur regulärer Sprachen

- Überschreitet die Anzahl der Ableitungsschritte eines Worts eine gewisse Grenze *j*, so muss aufgrund der endlichen Anzahl der Nonterminale mindestens eines davon mehrfach im Syntaxbaum vorkommen (hier das Nonterminal *A*).
- Daher lässt sich jedes hinreichend lange Wort in der Form *uvw* darstellen mit $|v| \geq 1$ und $|uv| \leq j$.

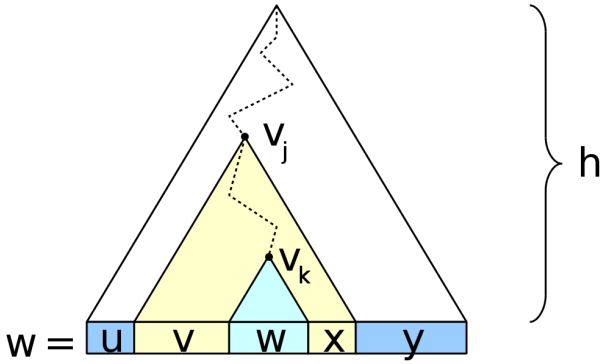


# „Aufpumpen“ des Mittelstücks

- Die Ableitung des Mittelstücks lässt sich beliebig oft wiederholen.
- Damit sind neben *uvw* immer auch die Wörter $uv^iw$ für alle $i \in N$ in der Sprache enthalten.

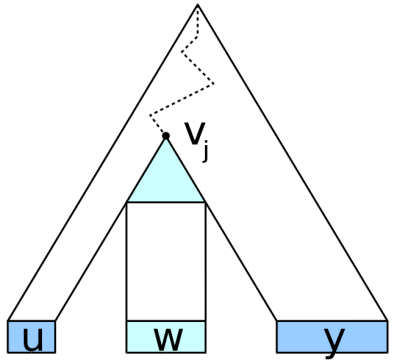


# Ableitungssequenz regulärer Sprachen

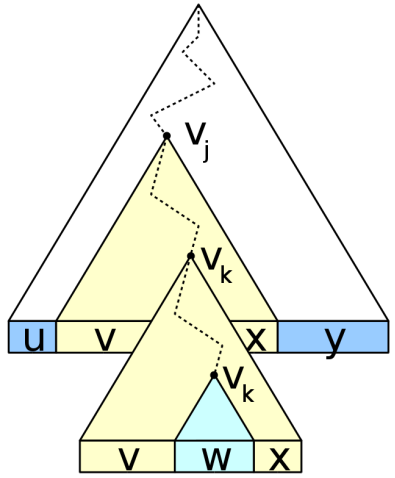



Gegeben: Wort $x \in L$
mit $|x| \geq n$

Ableitungsbaum T für x
mit Höhe $h \geq N$




Erzeugen von $uv^0wx^0y$




Erzeugen von $uv^2wx^2y$



https://de.wikipedia.org/wiki/Pumping-Lemma#/media/Datei:Pumping_Lemma_f%C3%BCr_kontextfreie_Sprachen.svg

# Pumping-Lemma für reguläre Sprachen

- Für jede reguläre Sprache $L$ existiert ein $j \in \mathbb{N}$, so dass sich alle Wörter $\omega \in L$ mit $|\omega| \geq j$ in der folgenden Form darstellen lassen:

$$\omega = uvw \text{ mit } |v| \geq 1 \text{ und } |uv| \leq j$$

- Dann ist mit $\omega$ auch das Wort $uv^i w$ für alle $i \in \mathbb{N}$ in $L$ enthalten.

# Pumping-Lemma für reguläre Sprachen

- Pumping-Lemma ist ein leistungsfähiges Instrument, um eine Sprache als nicht regulär zu erkennen
- Beweisführung folgt dem gleichen Muster:
  - Zunächst wird für die untersuchte Sprache L gezeigt, dass sich ein Wort $\omega \in L$ in der Form *uvw* darstellen lässt.
  - Dem Pumping-Lemma folgend muss dann auch das Wort $uv^iw$ in der Sprache enthalten sein.
  - Ist dies nicht der Fall, so kann L keine reguläre Sprache sein.

# Pumping-Lemma für reguläre Sprachen

- Mithilfe des Pumping-Lemmas lässt ich zeigen, dass die eingeführte Sprache

$$L_{C2} := \{a^n b^n \mid n \in N+\} \text{ nicht regulär ist.}$$

- Wäre $L_{C2}$ eine reguläre Sprache, so würde nach dem Pumping-Lemma ein $j \in N$ existieren, so dass sich jedes Wort $\omega$ mit $|\omega| \geq j$ in der Form *uvw* darstellen lässt mit $|v| \geq 1$ und $|uv| \leq j$.

# Pumping-Lemma für reguläre Sprachen

- Für das Wort $a^j b^j$ folgt hieraus, dass der (nichtleere) Mittelteil v nur aus a's bestehen kann.
- Mit *uvw* wäre dann aber auch das Wort $uv^2w = a^j a^{|v|} b^j = a^j\, a...a b^j$ $j \geq 1$ in L enthalten, im Widerspruch zum Aufbau von $L_{C2}$.

- Pumping-Lemma gibt ein Hilfsmittel an die Hand, um nachzuweisen, dass eine Sprache L nicht regulär ist.
- Schlussrichtung darf nicht umgekehrt werden:
  - Auch wenn die Wörter einer Sprache alle Eigenschaften des Pumping-Lemmas erfüllen, ist diese nicht notwendigerweise regulär.

# Pumping-Lemma für reguläre Sprachen

*Vergleicht man die Sprache $L_{C2} := \{a^n b^n \mid n \in N^+\}$ mit der Sprache $L_{C3} := \{(ab)^n \mid n \in N^+\}$, erscheint der Unterschied auf den ersten Blick nur marginal zu sein.*

- Trotzdem lässt sich $L_{C2}$, anders als $L_{C3}$, nicht mithilfe einer regulären Grammatik erzeugen.
- Schuld daran ist die Anordnung der Symbole **a** und **b**.
- Erzeugen man die einzelnen Zeichen eines Worts, wie in regulären Grammatiken gefordert, von links nach rechts, so muss man für alle Wörter der Form $a^n b^n$ zunächst die Anzahl der produzierten a's merken, um anschließend die richtige Anzahl von b's hervorbringen zu können.

# Pumping-Lemma für reguläre Sprachen

*Vergleicht man die Sprache $L_{C2} := \{a^n b^n \mid n \in N^+\}$ mit der Sprache $L_{C3} := \{(ab)^n \mid n \in N^+\}$, erscheint der Unterschied auf den ersten Blick nur marginal zu sein.*

- Da der Wert von n nach oben unbeschränkt ist, wären zu diesem Zweck unendlich viele Zustände notwendig.
- Aufgrund der endlichen Anzahl an Nonterminalen – unseren Zustandsmerkern – ist dies nicht möglich.
- Um die Wörter der Sprache $L_{C3}$ zu erzeugen, ist deutlich weniger logistische Arbeit erforderlich.
- Hier muss man sich nur merken, ob ein Terminalsymbol a erzeugt wurde oder nicht:
  - Im ersten Fall ist zunächst ein b zu produzieren, um ein gültiges Wort zu erhalten.
  - Im zweiten Fall hat man bereits ein korrektes Wort der Sprache vor sich.