Karlsruher Institut für Technologie

Theoretische Grundlagen der Informatik

Vorlesung am 19.01.2023

Torsten Ueckerdt | 19. Januar 2023

Letzte Vorlesung

Eine **Grammatik** ist $G = (\Sigma, V, S, R)$ mit

- Alphabet Σ aller Terminale,
- Menge V aller Variablen / Nichtterminale,
- Startvariable S aus V,
- Menge R aller Regeln / Ableitungen, wobei eine Ableitung ein Tupel (ℓ, r) aus Wörtern in $(\Sigma \cup V)^*$ ist.

Wir schreiben auch $\ell \to r$ für die Regel (ℓ, r) und entsprechend $S \overset{*}{\to} w$ falls w aus S abgeleitet werden kann.

Die **erzeugte Sprache** ist $L(G) = \{w \in \Sigma^* \mid S \overset{*}{\to} w\}$.

Eine Grammatik ist **kontextfrei** oder **Chomsky Typ-2**, wenn alle Regeln die folgende Form haben:

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

Letzte Vorlesung

Beispiel
Grammatik $G = (\Sigma, V, S, R)$ mit

$$\begin{aligned} V &= \{S, A, B\} \\ \Sigma &= \{0, 1\} \\ R &= \{S \to 0B \mid 1A, \\ &\quad A \to 0 \mid 0S \mid 1AA, \\ &\quad B \to 1 \mid 1S \mid 0BB\} \end{aligned}$$

erzeugt die Sprache $L = L(G) \subseteq \{0, 1\}^+$ aller Wörter mit genauso vielen Einsen wie Nullen.

Zum Beispiel ist $110010 \in L(G)$, wegen der Ableitung

$$S \to 1A \to 11AA \to 11A0 \to 110S0 \to 1100B0 \to 110010.$$

Heute

- Syntaxbäume
- Eindeutigkeit
- Chomsky-Normalform
- CYK-Algorithmus

Syntaxbäume

Eine Grammatik ist **kontextfrei** oder **Chomsky Typ-2**, wenn alle Regeln die folgende Form haben:

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

Syntaxbäume visualisieren die Ableitung eines einzelnen Wortes in einer kontextfreien Grammatik.

- An der Wurzel eines Syntaxbaumes steht das Startsymbol.
- Jeder innere Knoten enthält eine Variable.
- Die Blätter sind Symbole aus Σ oder ε.
- Wenn ein innerer Knoten A als Nachfolger von links nach rechts $(\alpha_1, \ldots, \alpha_r) \in V \cup \Sigma$ hat, so muss $A \to \alpha_1 \cdots \alpha_r$ eine Ableitungsregel der Grammatik sein.

Syntaxbäume – Beispiel

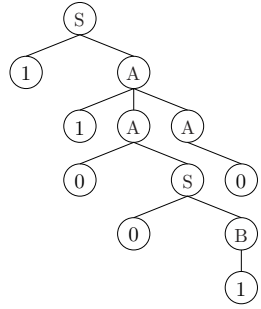
Betrachte die Regeln:

$R = \{S \to 0B|1A, \quad A \to 0|0S|1AA, \quad B \to 1|1S|0BB\}$

Betrachte die Ableitung:

$S \to 1A \to 11AA \to 11A0 \to 110S0 \to 1100B0 \to 110010.$

- Zu jeder Ableitung gehört genau ein Syntaxbaum.



Syntaxbäume – Beispiel

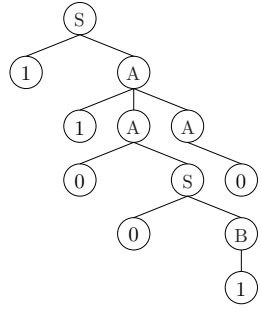
Betrachte die Regeln:

$R = \{S \to 0B|1A, \quad A \to 0|0S|1AA, \quad B \to 1|1S|0BB\}$

Betrachte die Ableitung:

$S \to 1A \to 11AA \to 11A0 \to 110S0 \to 1100B0 \to 110010.$

- Zu jeder Ableitung gehört genau ein Syntaxbaum.



Betrachte die Ableitung:

$S \to 1A \to 11AA \to 110SA \to 1100BA \to 11001A \to 110010$

- Zu einem Syntaxbaum können jedoch verschiedene Ableitungen des gleichen Wortes gehören.

Links/Rechtsableitung, Eindeutigkeit

Wegen der Kontextfreiheit ist bei Chomsky-2 Grammatiken die Reihenfolge, in der abgeleitet wird, für das Ergebnis unerheblich.

Definition.

Eine Linksableitung (Rechtsableitung) ist eine Ableitung, bei der in jedem Schritt die linkeste (rechteste) Variable abgeleitet wird.

Definition.

Eine kontextfreie Grammatik G heißt eindeutig, wenn es für jedes Wort $w \in L(G)$ genau einen Syntaxbaum gibt.

Eine kontextfreie Sprache L heißt eindeutig, wenn es eine eindeutige Grammatik G mit $L(G) = L$ gibt. Ansonsten heißt L inhärent mehrdeutig.

Beispiel: eindeutig

Die Sprache

$$L = \{0^n 1^n : n \geq 1\}$$

erzeugt durch die Grammatik

$$\begin{aligned} V &= \{S\} \\ \Sigma &= \{0, 1\} \\ R &= \{S \to 01 \mid 0S1\} \end{aligned}$$

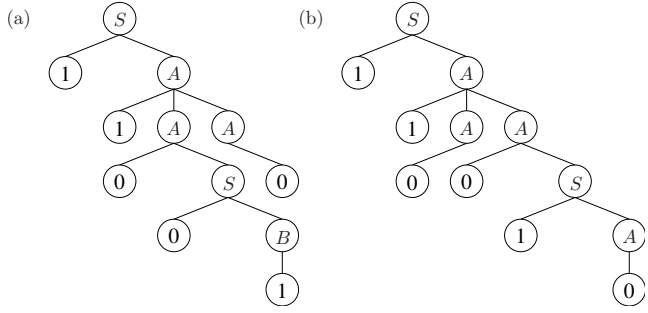
ist eindeutig.

Beispiel: nicht eindeutig

Die Sprache gegeben durch die Regeln

$$R = \{S \to 0B \mid 1A, A \to 0 \mid 0S \mid 1AA, B \to 1 \mid 1S \mid 0BB\}$$

ist nicht eindeutig.



Heute

- Syntaxbäume
- Eindeutigkeit
- Chomsky-Normalform
- CYK-Algorithmus

Chomsky-Normalform

Definition.

Eine kontextfreie Grammatik ist in Chomsky-Normalform, wenn alle Regeln von der Form:

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

sind, mit $A, B, C \in V$ und $a \in \Sigma$.

Erinnerung:
In **kontextfreien** Grammatiken sind alle Regeln von der Form:

$A \to v$

mit $A \in V$

und $v \in (\Sigma \cup V)^*$

Chomsky-Normalform

Definition.

Eine kontextfreie Grammatik ist in Chomsky-Normalform, wenn alle Regeln von der Form:

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

sind, mit $A, B, C \in V$ und $a \in \Sigma$.

- Grammatiken in Chomsky-Normalform können also nicht das Wort ε erzeugen.
- Für kontextfreie Sprachen, die ε enthalten, lässt sich eine Grammatik in Chomsky-Normalform leicht "erweitern" durch die Regeln

$$S' \to \varepsilon \qquad \text{und} \qquad S' \to S,$$

wobei S' ein neues Startsymbol zur Erzeugung von ε ist.

Erinnerung:
In **kontextfreien** Grammatiken sind alle Regeln von der Form:

$A \to v$
mit $A \in V$
und $v \in (\Sigma \cup V)^*$

Chomsky-Normalform

Satz.

Jede kontextfreie Grammatik kann in eine äquivalente Grammatik in erweiterter Chomsky-Normalform überführt werden.

Eine Grammatik in erweiterter Chomsky-Normalform ist eine Grammatik in Chomsky-Normalform mit den zusätzlichen Regeln $S' \to \varepsilon$ und $S' \to S$.

Beweis (konstruktiv):

- Wir geben eine "Schritt–für–Schritt"–Überführung der Regeln einer beliebigen kontextfreien Grammatik in Regeln in Normalform an.
- Großbuchstaben repräsentieren immer Variablen / Nichtterminale.
- Kleinbuchstaben repräsentieren immer Terminale.

Chomsky-Normalform

Wir veranschaulichen den Beweis an folgendem Beispiel:
Grammatik $G = (\Sigma, V, S, R)$ mit

$$\begin{aligned} \Sigma &= \{a, b\} \\ V &= \{A, B, C, D, E, S\} \end{aligned}$$

und der folgenden Regelmenge R:

$$\begin{aligned} S &\to A \mid aAa \mid bBb \mid \varepsilon \\ A &\to C \mid a \\ B &\to b \\ C &\to AF \mid CDE \mid \varepsilon \\ D &\to A \mid B \mid ab \\ E &\to B \\ F &\to D \mid E \end{aligned}$$

Schritt 1

Vorgehen:
Ersetze dazu in allen rechten Seiten von Regeln Symbole $a \in \Sigma$ durch neue Variablen Y_a und füge die Regeln $Y_a \to a$ hinzu.

Ziel:
Alle Regeln enthalten auf der rechten Seite nur Symbole aus V oder nur ein Symbol aus Σ.

Schritt 1

Vorgehen:

Ersetze dazu in allen rechten Seiten von Regeln Symbole $a \in \Sigma$ durch neue Variablen Y_a und füge die Regeln $Y_a \to a$ hinzu.

$$
\begin{array}{lcl}
S & \to & A \mid aAa \mid bBb \mid \varepsilon \\
A & \to & C \mid a \\
B & \to & b \\
C & \to & AF \mid CDE \mid \varepsilon \\
D & \to & A \mid B \mid ab \\
E & \to & B \\
F & \to & D \mid E
\end{array}
\quad \Rightarrow \quad
\begin{array}{lcl}
S & \to & A \mid Y_aAY_a \mid Y_bBY_b \mid \varepsilon \\
A & \to & C \mid a \\
B & \to & b \\
C & \to & AF \mid CDE \mid \varepsilon \\
D & \to & A \mid B \mid Y_aY_b \\
E & \to & B \\
F & \to & D \mid E \\
Y_a & \to & a \\
Y_b & \to & b
\end{array}
$$

Ziel:

Alle Regeln enthalten auf der rechten Seite nur Symbole aus V oder nur ein Symbol aus Σ.

Schritt 2

Ziel:

- Alle rechten Seiten haben Länge ≤ 2.

Vorgehen:

- Sei $A \to B_1 \cdots B_m$ Regel mit $m > 2$.
- Führe $m-2$ neue Variablen $C_1, \ldots, C_{m-2}$ ein, und ersetze die Regel

$$A \to B_1 \cdots B_m$$

durch neue Regeln

$$\begin{aligned} A &\to B_1 C_1 \\ C_i &\to B_{i+1} C_{i+1} \qquad \text{für } 1 \leq i \leq m-3 \\ C_{m-2} &\to B_{m-1} B_m \end{aligned}$$

Schritt 2

$$
\begin{array}{lcl}
S & \to & A \mid Y_aAY_a \mid Y_bBY_b \mid \varepsilon \\
A & \to & C \mid a \\
B & \to & b \\
C & \to & AF \mid CDE \mid \varepsilon \\
D & \to & A \mid B \mid Y_aY_b \\
E & \to & B \\
F & \to & D \mid E \\
Y_a & \to & a \\
Y_b & \to & b
\end{array}
\quad \Rightarrow \quad
\begin{array}{lcl}
S & \to & A \mid Y_aC_1 \mid Y_bC_2 \mid \varepsilon \\
A & \to & C \mid a \\
B & \to & b \\
C & \to & AF \mid CC_3 \mid \varepsilon \\
D & \to & A \mid B \mid Y_aY_b \\
E & \to & B \\
F & \to & D \mid E \\
C_1 & \to & AY_a \\
C_2 & \to & BY_b \\
C_3 & \to & DE \\
Y_a & \to & a \\
Y_b & \to & b
\end{array}
$$

Ziel:
Alle rechten Seiten haben Länge ≤ 2.

Schritt 3

Vorgehen, Phase 1:

Finde die Menge V' aller Variablen A für die $A \xrightarrow{*} \varepsilon$ existiert.

- Es werden erst alle A mit $A \to \varepsilon$ in V' aufgenommen.
- Dann wird geprüft, ob neue Regeln $B \to \varepsilon$ entstehen, wenn man A in allen Regeln auf der rechten Seite A durch ε ersetzt.
- Ist dies der Fall, so werden die entsprechenden Variablen B in V' aufgenommen und genauso behandelt.
- Das Verfahren hört auf, wenn V' sich nicht mehr ändert.

Ziel:
Es kommen keine Regeln $A \to \varepsilon$ vor.

Schritt 3

Vorgehen, Phase 1:

Finde die Menge V' aller Variablen A für die $A \overset{*}{\rightarrow} \varepsilon$ existiert.

- Es werden erst alle A mit $A \rightarrow \varepsilon$ in V' aufgenommen.
- Dann wird geprüft, ob neue Regeln $B \rightarrow \varepsilon$ entstehen, wenn man A in allen Regeln auf der rechten Seite A durch ε ersetzt.
- Ist dies der Fall, so werden die entsprechenden Variablen B in V' aufgenommen und genauso behandelt.
- Das Verfahren hört auf, wenn V' sich nicht mehr ändert.

Bemerkung:

- In Phase 1 werden noch keine Regeln geändert.
- Die Ersetzung ist also nur “testweise”.
- Am Ende enthält V' alle Variablen A mit $A \overset{*}{\rightarrow} \varepsilon$.

Ziel:
Es kommen keine Regeln $A \rightarrow \varepsilon$ vor.

Schritt 3

Vorgehen, Phase 1:

Finde die Menge V' aller Variablen A für die $A \overset{*}{\rightarrow} \varepsilon$ existiert.

Vorgehen, Phase 2:

Ersetzung der Regeln $A \rightarrow \varepsilon$.

- Gegeben V' aus Phase 1.
- Streiche alle Regeln $A \rightarrow \varepsilon$.
- Für $A \rightarrow BC$ füge die zusätzliche Regel
  - $A \rightarrow B$ falls $C \in V'$,
  - $A \rightarrow C$ falls $B \in V'$

  ein.
- (Die Regel $A \rightarrow BC$ wird nicht gestrichen).

Ziel:
Es kommen keine Regeln $A \rightarrow \varepsilon$ vor.

Schritt 3

Vorgehen, Phase 1:

- Initialisierung
  $V' = \{A \mid A \to \varepsilon\} = \{S, C\}$
- Erster Durchlauf liefert: A
  $\Rightarrow V' = \{A, C, S\}$
- Zweiter Durchlauf liefert: D
  $\Rightarrow V' = \{A, C, D, S\}$
- Dritter Durchlauf liefert: F
  $\Rightarrow V' = \{A, C, D, F, S\}$
- Vierter Durchlauf liefert nichts neues.

$$
\begin{aligned}
S &\to A \mid Y_a C_1 \mid Y_b C_2 \mid \varepsilon \\
A &\to C \mid a \\
B &\to b \\
C &\to AF \mid CC_3 \mid \varepsilon \\
D &\to A \mid B \mid Y_a Y_b \\
E &\to B \\
F &\to D \mid E \\
C_1 &\to AY_a \\
C_2 &\to BY_b \\
C_3 &\to DE \\
Y_a &\to a \\
Y_b &\to b
\end{aligned}
$$

Ziel:
Es kommen keine Regeln $A \to \varepsilon$ vor.

Schritt 3

Vorgehen, Phase 2:

- Streiche ε-Produktionen
- Simuliere Ableitungen $A \overset{*}{\rightarrow} \varepsilon$ auf den verbleibenden Regeln $V' = \{A, C, D, F, S\}$

$$
\begin{aligned}
S &\rightarrow A \mid Y_a C_1 \mid Y_b C_2 \\
A &\rightarrow C \mid a \\
B &\rightarrow b \\
C &\rightarrow AF \mid F \mid A \mid CC_3 \mid C_3 \\
D &\rightarrow A \mid B \mid Y_a Y_b \\
E &\rightarrow B \\
F &\rightarrow D \mid E \\
C_1 &\rightarrow AY_a \mid Y_a \\
C_2 &\rightarrow BY_b \\
C_3 &\rightarrow DE \mid E \\
Y_a &\rightarrow a \\
Y_b &\rightarrow b
\end{aligned}
$$

Ziel:
Es kommen keine Regeln $A \rightarrow \varepsilon$ vor.

Schritt 4

Beispiel:

$$
\begin{aligned}
A &\rightarrow B \mid C \\
B &\rightarrow C \\
C &\rightarrow c
\end{aligned}
$$

Ziel:
Die Grammatik enthält keine (Ketten-)Regeln der Form $A \rightarrow B$.

Schritt 4

Vorgehen:

- **Phase 1:** Finde alle Kreise $A_1 \to A_2 \to \ldots \to A_n \to A_1$.
  Ersetze alle A_i durch A_1.
- **Phase 2:** Betrachte die Regeln der Form $A \to B$
  in umgekehrt topologischer Sortierung.
  - Für jede Regel $A \to B$ und jede Regel $B \to \beta$
    füge Regel $A \to \beta$ hinzu.
  - Lösche Regel $A \to B$.

Ziel:
Die Grammatik enthält keine (Ketten-)Regeln der Form $A \to B$.

Schritt 4

Vorgehen:

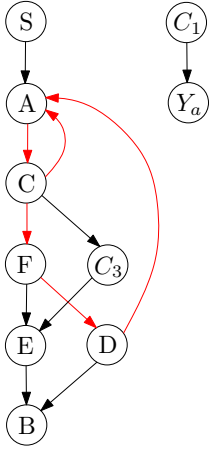
- **Phase 1:** Finde alle Kreise $A_1 \to A_2 \to \ldots \to A_n \to A_1$.
  Ersetze alle A_i durch A_1.
- **Phase 2:** Betrachte die Regeln der Form $A \to B$
  in umgekehrt topologischer Sortierung.
  - Für jede Regel $A \to B$ und jede Regel $B \to \beta$
    füge Regel $A \to \beta$ hinzu.
  - Lösche Regel $A \to B$.

Topologische Sortierung der Regelmenge:

- $V_1, \ldots, V_k$ Menge von Variablen aus Kettenregeln
- $V_1, \ldots, V_k$ topologisch sortiert, wenn gilt:
- $V_i \overset{*}{\to} V_j \Rightarrow i < j$
- Voraussetzung: Es gibt keine zyklischen Abhängigkeiten

Ziel:
Die Grammatik enthält keine (Ketten-)Regeln der Form $A \to B$.

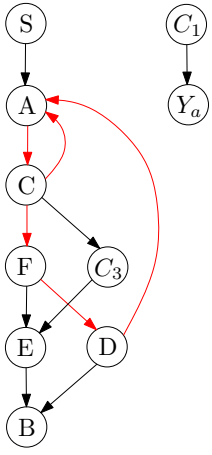
Abhängigkeitsgraph



$$
\begin{aligned}
S &\to A \mid Y_a C_1 \mid Y_b C_2 \\
A &\to C \mid a \\
B &\to b \\
C &\to AF \mid F \mid A \mid CC_3 \mid C_3 \\
D &\to A \mid B \mid Y_a Y_b \\
E &\to B \\
F &\to D \mid E \\
C_1 &\to AY_a \mid Y_a \\
C_2 &\to BY_b \\
C_3 &\to DE \mid E \\
Y_a &\to a \\
Y_b &\to b
\end{aligned}
$$

Ziel:
Die Grammatik enthält keine (Ketten-)Regeln der Form $A \to B$.

Schritt 4 – Phase 1



- Zyklen $A \to C$ und $A \to C \to F \to D \to A$
- $\Rightarrow A, C, F, D$ äquivalent
- Entferne an Zyklen beteiligte Regeln
- Ersetze Vorkommen von C, F, D in allen Regeln durch A
- Lösche Regeln der Form $A \to A$

Ziel:
Die Grammatik enthält keine (Ketten-)Regeln der Form $A \to B$.

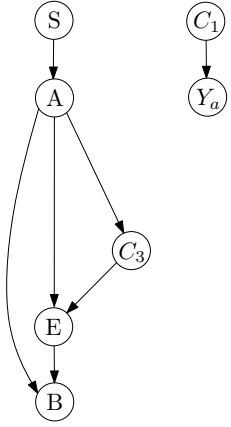
Schritt 4 – Phase 1

$$
\begin{array}{lcl}
S & \to & A \mid Y_aC_1 \mid Y_bC_2 \\
A & \to & C \mid a \\
B & \to & b \\
C & \to & AF \mid F \mid A \mid CC_3 \mid C_3 \\
D & \to & A \mid B \mid Y_aY_b \\
E & \to & B \\
F & \to & D \mid E \\
C_1 & \to & AY_a \mid Y_a \\
C_2 & \to & BY_b \\
C_3 & \to & DE \mid E \\
Y_a & \to & a \\
Y_b & \to & b
\end{array}
\quad \Rightarrow \quad
\begin{array}{lcl}
S & \to & A \mid Y_aC_1 \mid Y_bC_2 \\
A & \to & a \\
B & \to & b \\
A & \to & AA \mid AC_3 \mid C_3 \\
A & \to & B \mid Y_aY_b \\
E & \to & B \\
A & \to & E \\
C_1 & \to & AY_a \mid Y_a \\
C_2 & \to & BY_b \\
C_3 & \to & AE \mid E \\
Y_a & \to & a \\
Y_b & \to & b
\end{array}
$$

Ziel:
Die Grammatik enthält keine (Ketten-)Regeln der Form $A \to B$.

Zyklus:
$A \to C \to F \to D \to A$

Schritt 4 – Phase 2



$$
\begin{aligned}
S &\to A \mid Y_a C_1 \mid Y_b C_2 \\
A &\to a \mid AA \mid AC_3 \mid C_3 \mid B \mid Y_a Y_b \mid E \\
B &\to b \\
E &\to B \\
C_1 &\to AY_a \mid Y_a \\
C_2 &\to BY_b \\
C_3 &\to AE \mid E \\
Y_a &\to a \\
Y_b &\to b
\end{aligned}
$$

Ziel:
Die Grammatik enthält keine (Ketten-)Regeln der Form $A \to B$.

Topologische Sortierung: $S, A, C_3, E, B, C_1, Y_a$

Schritt 4 – Phase 2

- Ersetze $A \to B$ durch $A \to \beta$, falls Regel $B \to \beta$ existiert.

$$
\begin{array}{lcl}
S & \to & A \mid Y_aC_1 \mid Y_bC_2 \\
\\
A & \to & a \mid AA \mid AC_3 \mid C_3 \mid \\
 & & B \mid Y_aY_b \mid E \\
B & \to & b \\
E & \to & B \\
C_1 & \to & AY_a \mid Y_a \\
C_2 & \to & BY_b \\
C_3 & \to & AE \mid E \\
Y_a & \to & a \\
Y_b & \to & b
\end{array}
\quad \Rightarrow \quad
\begin{array}{lcl}
S & \to & A \mid Y_aC_1 \mid Y_bC_2 \\
\\
A & \to & a \mid AA \mid AC_3 \mid C_3 \mid \\
 & & B \mid Y_aY_b \mid E \\
B & \to & b \\
E & \to & B \\
C_1 & \to & AY_a \mid a \\
C_2 & \to & BY_b \\
C_3 & \to & AE \mid E \\
Y_a & \to & a \\
Y_b & \to & b
\end{array}
$$

Ziel:
Die Grammatik enthält keine (Ketten-)Regeln der Form $A \to B$.

Topologische Sortierung:
$S, A, C_3, E, B, C_1, Y_a$

Schritt 4 – Phase 2

- Ersetze $A \to B$ durch $A \to \beta$, falls Regel $B \to \beta$ existiert.

$$
\begin{array}{lcl}
S & \to & A \mid Y_a C_1 \mid Y_b C_2 \\
\\
A & \to & a \mid AA \mid AC_3 \mid C_3 \mid \\
 & & B \mid Y_a Y_b \mid E \\
B & \to & b \\
E & \to & B \\
C_1 & \to & AY_a \mid Y_a \\
C_2 & \to & BY_b \\
C_3 & \to & AE \mid E \\
Y_a & \to & a \\
Y_b & \to & b
\end{array}
\quad \Rightarrow \quad
\begin{array}{lcl}
S & \to & A \mid Y_a C_1 \mid Y_b C_2 \\
\\
A & \to & a \mid AA \mid AC_3 \mid C_3 \mid \\
 & & B \mid Y_a Y_b \mid E \\
B & \to & b \\
E & \to & B \\
C_1 & \to & AY_a \mid a \\
C_2 & \to & BY_b \\
C_3 & \to & AE \mid E \\
Y_a & \to & a \\
Y_b & \to & b
\end{array}
$$

Ziel:
Die Grammatik enthält keine (Ketten-)Regeln der Form $A \to B$.

Topologische Sortierung:
$S, A, C_3, E, B, C_1, Y_a$

Schritt 4 – Phase 2

- Ersetze $A \to B$ durch $A \to \beta$, falls Regel $B \to \beta$ existiert.

$$
\begin{array}{lcl}
S & \to & A \mid Y_a C_1 \mid Y_b C_2 \\
A & \to & a \mid AA \mid AC_3 \mid C_3 \mid \\
 & & B \mid Y_a Y_b \mid E \\
B & \to & b \\
E & \to & B \\
C_1 & \to & AY_a \mid Y_a \\
C_2 & \to & BY_b \\
C_3 & \to & AE \mid E \\
Y_a & \to & a \\
Y_b & \to & b
\end{array}
\quad \Rightarrow \quad
\begin{array}{lcl}
S & \to & A \mid Y_a C_1 \mid Y_b C_2 \\
A & \to & a \mid AA \mid AC_3 \mid C_3 \mid \\
 & & b \mid Y_a Y_b \mid E \\
B & \to & b \\
E & \to & b \\
C_1 & \to & AY_a \mid a \\
C_2 & \to & BY_b \\
C_3 & \to & AE \mid E \\
Y_a & \to & a \\
Y_b & \to & b
\end{array}
$$

Ziel:
Die Grammatik enthält keine (Ketten-)Regeln der Form $A \to B$.

Topologische Sortierung:
$S, A, C_3, E, B, C_1, Y_a$

Schritt 4 – Phase 2

- Ersetze $A \to B$ durch $A \to \beta$, falls Regel $B \to \beta$ existiert.

$$
\begin{array}{lcl}
S & \to & A \mid Y_aC_1 \mid Y_bC_2 \\
\\
A & \to & a \mid AA \mid AC_3 \mid C_3 \mid \\
 & & B \mid Y_aY_b \mid E \\
B & \to & b \\
E & \to & B \\
C_1 & \to & AY_a \mid Y_a \\
C_2 & \to & BY_b \\
C_3 & \to & AE \mid E \\
Y_a & \to & a \\
Y_b & \to & b
\end{array}
\quad \Rightarrow \quad
\begin{array}{lcl}
S & \to & A \mid Y_aC_1 \mid Y_bC_2 \\
\\
A & \to & a \mid AA \mid AC_3 \mid C_3 \mid \\
 & & b \mid Y_aY_b \\
B & \to & b \\
E & \to & b \\
C_1 & \to & AY_a \mid a \\
C_2 & \to & BY_b \\
C_3 & \to & AE \mid b \\
Y_a & \to & a \\
Y_b & \to & b
\end{array}
$$

Ziel:
Die Grammatik enthält keine (Ketten-)Regeln der Form $A \to B$.

Topologische Sortierung:
$S, A, C_3, E, B, C_1, Y_a$

Schritt 4 – Phase 2

- Ersetze $A \to B$ durch $A \to \beta$, falls Regel $B \to \beta$ existiert.

$$
\begin{array}{lcl}
S & \to & A \mid Y_a C_1 \mid Y_b C_2 \\
A & \to & a \mid AA \mid AC_3 \mid C_3 \mid \\
 & & B \mid Y_a Y_b \mid E \\
B & \to & b \\
E & \to & B \\
C_1 & \to & AY_a \mid Y_a \\
C_2 & \to & BY_b \\
C_3 & \to & AE \mid E \\
Y_a & \to & a \\
Y_b & \to & b
\end{array}
\quad \Rightarrow \quad
\begin{array}{lcl}
S & \to & A \mid Y_a C_1 \mid Y_b C_2 \\
A & \to & a \mid AA \mid AC_3 \mid AE \mid \\
 & & b \mid Y_a Y_b \\
B & \to & b \\
E & \to & b \\
C_1 & \to & AY_a \mid a \\
C_2 & \to & BY_b \\
C_3 & \to & AE \mid b \\
Y_a & \to & a \\
Y_b & \to & b
\end{array}
$$

Ziel:
Die Grammatik enthält keine (Ketten-)Regeln der Form $A \to B$.

Topologische Sortierung:
$S, A, C_3, E, B, C_1, Y_a$

Schritt 4 – Phase 2

- Ersetze $A \to B$ durch $A \to \beta$, falls Regel $B \to \beta$ existiert.

$$
\begin{array}{lcl}
S & \to & A \mid Y_aC_1 \mid Y_bC_2 \\
A & \to & a \mid AA \mid AC_3 \mid C_3 \mid \\
 & & B \mid Y_aY_b \mid E \\
B & \to & b \\
E & \to & B \\
C_1 & \to & AY_a \mid Y_a \\
C_2 & \to & BY_b \\
C_3 & \to & AE \mid E \\
Y_a & \to & a \\
Y_b & \to & b
\end{array}
\quad \Rightarrow \quad
\begin{array}{lcl}
S & \to & Y_aC_1 \mid Y_bC_2 \mid a \mid AA \\
 & & AC_3 \mid AE \mid b \mid Y_aY_b \\
A & \to & a \mid AA \mid AC_3 \mid AE \mid \\
 & & b \mid Y_aY_b \\
B & \to & b \\
E & \to & b \\
C_1 & \to & AY_a \mid a \\
C_2 & \to & BY_b \\
C_3 & \to & AE \mid b \\
Y_a & \to & a \\
Y_b & \to & b
\end{array}
$$

Ziel:
Die Grammatik enthält keine (Ketten-)Regeln der Form $A \to B$.

Topologische Sortierung:
$S, A, C_3, E, B, C_1, Y_a$

Schritt 4 – Phase 2

- Ersetze $A \to B$ durch $A \to \beta$, falls Regel $B \to \beta$ existiert.

$$
\begin{array}{lcl}
S & \to & A \mid Y_a C_1 \mid Y_b C_2 \\
A & \to & a \mid AA \mid AC_3 \mid C_3 \mid \\
 & & B \mid Y_a Y_b \mid E \\
B & \to & b \\
E & \to & B \\
C_1 & \to & AY_a \mid Y_a \\
C_2 & \to & BY_b \\
C_3 & \to & AE \mid E \\
Y_a & \to & a \\
Y_b & \to & b
\end{array}
\quad \Rightarrow \quad
\begin{array}{lcl}
S & \to & Y_a C_1 \mid Y_b C_2 \mid a \mid AA \\
 & & AC_3 \mid AE \mid b \mid Y_a Y_b \\
A & \to & a \mid AA \mid AC_3 \mid AE \mid \\
 & & b \mid Y_a Y_b \\
B & \to & b \\
E & \to & b \\
C_1 & \to & AY_a \mid a \\
C_2 & \to & BY_b \\
C_3 & \to & AE \mid b \\
Y_a & \to & a \\
Y_b & \to & b
\end{array}
$$

Ziel:
Die Grammatik enthält keine (Ketten-)Regeln der Form $A \to B$.

Topologische Sortierung:
$S, A, C_3, E, B, C_1, Y_a$

Sonderbehandlung $\varepsilon \in L(G)$

Karlsruher Institut für Technologie

Sonderbehandlung von ε-Produktionen

- Die konstruierte Grammatik ist nun in Chomsky-Normalform, aber
- falls in der ursprünglichen Grammatik G eine Ableitungsfolge $S \overset{*}{\longrightarrow} \varepsilon$ existierte, haben wir diese entfernt.
- Erweitere dann Grammatik um Regeln $S' \to S$ und $S' \to \varepsilon$ für neues Startsymbol S'.

Heute

- Syntaxbäume
- Eindeutigkeit
- Chomsky-Normalform
- CYK-Algorithmus

Der CYK-Algorithmus

Satz.

Es gibt einen Algorithmus (den Cocke-Younger-Kasami Algorithmus), der für eine kontextfreie Grammatik G in Chomsky-Normalform und ein Wort $w \in \Sigma^*$ in Zeit $O(|R| \cdot n^3)$ entscheidet, ob $w \in L(G)$, wobei $n = |w|$ und $|R|$ die Anzahl der Regeln von G ist.

Beweis – Beschreibung des CYK-Algorithmus

- Sei $w = w_1 \cdots w_n$.
- Sei $V_{ij} \subseteq V$ so dass $A \in V_{ij}$ genau dann, wenn $A \overset{*}{\rightarrow} w_i \cdots w_j$.
- Für alle $1 \leq i \leq j \leq n$ berechne die Menge $V_{ij} \subseteq V$.
- Dann ist $w \in L(G)$ genau dann, wenn $S \in V_{1n}$ ist.
- Die Tabelle der V_{ij} wird nach wachsendem $\ell := j - i$ aufgebaut, beginnend mit $\ell = 0$.
- Für $j - i = \ell > 0$ wird die Berechnung von V_{ij} systematisch auf zuvor berechnete V_{ik}, $V_{k+1\,j}$ mit $i \leq k < j$ zurückgeführt.

Bemerkung

- Wir benutzen dynamische Programmierung.

CYK-Algorithmus – Beispiel

$$
\begin{array}{rcl}
S & \to & SY_* \mid SC_1 \mid \\
 & & SS \mid Y_(C_2 \mid \\
 & & e \mid \emptyset \mid 0 \mid 1 \\
C_1 & \to & Y_\cup S \\
C_2 & \to & SY_) \\
Y_* & \to & * \\
Y_\cup & \to & \cup \\
Y_(& \to & (\\
Y_) & \to &)
\end{array}
$$

V_{ii+7}	V_{18}							
V_{ii+6}	V_{17}	V_{28}						
V_{ii+5}	V_{16}	V_{27}	V_{38}					
V_{ii+4}	V_{15}	V_{26}	V_{37}	V_{48}				
V_{ii+3}	V_{14}	V_{25}	V_{36}	V_{47}	V_{58}			
V_{ii+2}	V_{13}	V_{24}	V_{35}	V_{46}	V_{57}	V_{68}		
V_{ii+1}	V_{12}	V_{23}	V_{34}	V_{45}	V_{56}	V_{67}	V_{78}	
V_{ii}	V_{11}	V_{22}	V_{33}	V_{44}	V_{55}	V_{66}	V_{77}	V_{88}
	1	*	$\cup$	(	0	1	)	*
i	1	2	3	4	5	6	7	8

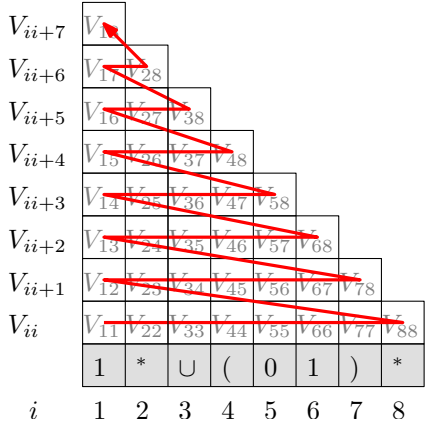
Wir testen, ob das Wort

$$w = 1^* \cup (01)^*$$

von der Grammatik erzeugt werden kann.

CYK-Algorithmus – Beispiel

$$
\begin{array}{rcl}
S & \to & SY_* \mid SC_1 \mid \\
 & & SS \mid Y_(C_2 \mid \\
 & & e \mid \emptyset \mid 0 \mid 1 \\
C_1 & \to & Y_\cup S \\
C_2 & \to & SY_) \\
Y_* & \to & * \\
Y_\cup & \to & \cup \\
Y_(& \to & (\\
Y_) & \to &)
\end{array}
$$



Wir testen, ob das Wort

$$w = 1^* \cup (01)^*$$

von der Grammatik erzeugt werden kann.

CYK-Algorithmus – Vorgehen

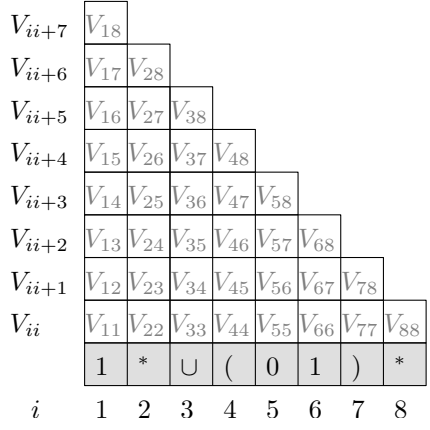
- Sei $V_{ij} \subseteq V$ so dass $A \in V_{ij}$ genau dann, wenn $A \overset{*}{\rightarrow} w_i \cdots w_j$.
- Die Tabelle der V_{ij} wird nach wachsendem $\ell := j - i$ aufgebaut, beginnend mit $\ell = 0$.
- Für $j - i = \ell > 0$ wird die Berechnung von V_{ij} systematisch auf zuvor berechnete V_{ik}, $V_{k+1\,j}$ mit $i \leq k < j$ zurückgeführt.

Fall $\ell = 0$:

- Konstruiere die Mengen V_{ii}, d.h. alle $A \in V$ mit $A \overset{*}{\rightarrow} w_i$.
- Da G in Chomsky-Normalform ist, gilt $A \overset{*}{\rightarrow} w_i$ nur, wenn $(A \rightarrow w_i) \in R$.
- Die Berechnung von V_{ii} ist für alle $i \in \{1, \ldots n\}$ in $O(|R|)$ möglich.

CYK-Algorithmus – Beispiel

$$
\begin{array}{rcl}
S & \to & SY_* \mid SC_1 \mid \\
 & & SS \mid Y_(C_2 \mid \\
 & & e \mid \emptyset \mid 0 \mid 1 \\
C_1 & \to & Y_\cup S \\
C_2 & \to & SY_) \\
Y_* & \to & * \\
Y_\cup & \to & \cup \\
Y_(& \to & (\\
Y_) & \to &)
\end{array}
$$



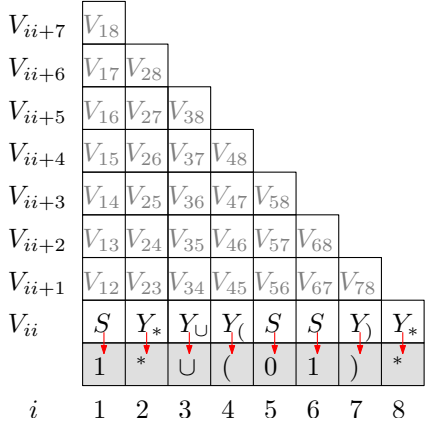
Wir testen, ob das Wort

$$w = 1^* \cup (01)^*$$

von der Grammatik erzeugt werden kann.

CYK-Algorithmus – Beispiel

$$
\begin{array}{rcl}
S & \to & SY_* \mid SC_1 \mid \\
 & & SS \mid Y_(C_2 \mid \\
 & & e \mid \emptyset \mid 0 \mid 1 \\
C_1 & \to & Y_\cup S \\
C_2 & \to & SY_) \\
Y_* & \to & * \\
Y_\cup & \to & \cup \\
Y_(& \to & (\\
Y_) & \to &)
\end{array}
$$



Wir testen, ob das Wort

$$w = 1^* \cup (01)^*$$

von der Grammatik erzeugt werden kann.

CYK-Algorithmus – Vorgehen

- Sei $V_{ij} \subseteq V$ so dass $A \in V_{ij}$ genau dann, wenn $A \overset{*}{\rightarrow} w_i \cdots w_j$.
- Die Tabelle der V_{ij} wird nach wachsendem $\ell := j - i$ aufgebaut, beginnend mit $\ell = 0$.
- Für $j - i = \ell > 0$ wird die Berechnung von V_{ij} systematisch auf zuvor berechnete V_{ik}, $V_{k+1\,j}$ mit $i \leq k < j$ zurückgeführt.

Fall $\ell > 0$:

- Jede Ableitung von $w_i \cdots w_j$ muss mit einer Regel der Form

$$A \rightarrow BC$$

beginnen, wobei ein $k \in \{i, \ldots, j-1\}$ existiert mit

- $B \overset{*}{\rightarrow} w_i \cdots w_k$ und
- $C \overset{*}{\rightarrow} w_{k+1} \cdots w_j$.

CYK-Algorithmus – Vorgehen

Verfahren

- Speichere alle Mengen V_{rs} als Arrays der Länge $|V|$, in denen für jedes $A \in V$ markiert ist, ob $A \in V_{rs}$.

- **Berechnung von V_{ij}:**
  Überprüfe für jede Regel $(A \to BC) \in R$ und jedes k, ob

$$B \overset{*}{\to} w_i \cdots w_k$$
$$C \overset{*}{\to} w_{k+1} \cdots w_j$$

  durch Ansehen der Stelle
  - B im Array zu V_{ik} und
  - C im Array zu $V_{k+1\ j}$.

CYK-Algorithmus – Vorgehen

Verfahren

- Speichere alle Mengen V_{rs} als Arrays der Länge $|V|$, in denen für jedes $A \in V$ markiert ist, ob $A \in V_{rs}$.

- **Berechnung von V_{ij}:**
  Überprüfe für jede Regel $(A \to BC) \in R$ und jedes k, ob

$$B \overset{*}{\to} w_i \cdots w_k$$
$$C \overset{*}{\to} w_{k+1} \cdots w_j$$

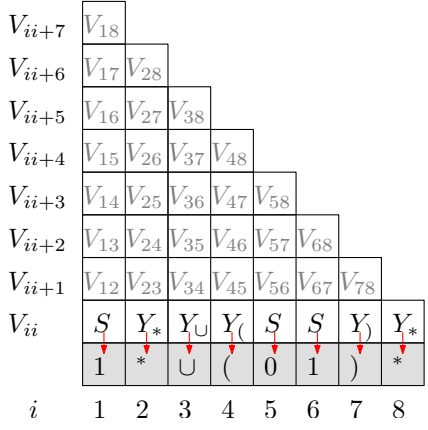
  durch Ansehen der Stelle
  - B im Array zu V_{ik} und
  - C im Array zu $V_{k+1\ j}$.

Bemerkung

- Dies benötigt Aufwand in $O(n \cdot |R|)$

CYK-Algorithmus – Beispiel

$$
\begin{array}{rcl}
S & \to & SY_* \mid SC_1 \mid \\
 & & SS \mid Y_(C_2 \mid \\
 & & e \mid \emptyset \mid 0 \mid 1 \\
C_1 & \to & Y_\cup S \\
C_2 & \to & SY_) \\
Y_* & \to & * \\
Y_\cup & \to & \cup \\
Y_(& \to & (\\
Y_) & \to &)
\end{array}
$$



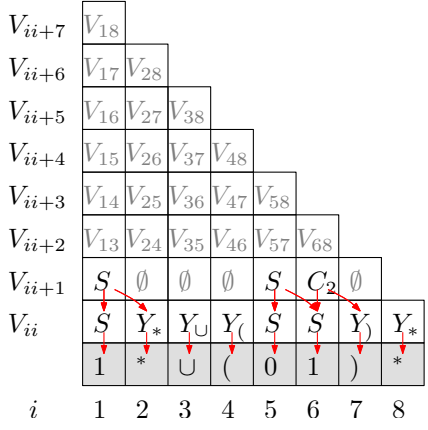
Wir testen, ob das Wort

$$w = 1^* \cup (01)^*$$

von der Grammatik erzeugt werden kann.

CYK-Algorithmus – Beispiel

$$
\begin{array}{rcl}
S & \to & SY_* \mid SC_1 \mid \\
 & & SS \mid Y_(C_2 \mid \\
 & & e \mid \emptyset \mid 0 \mid 1 \\
C_1 & \to & Y_\cup S \\
C_2 & \to & SY_) \\
Y_* & \to & * \\
Y_\cup & \to & \cup \\
Y_(& \to & (\\
Y_) & \to &)
\end{array}
$$



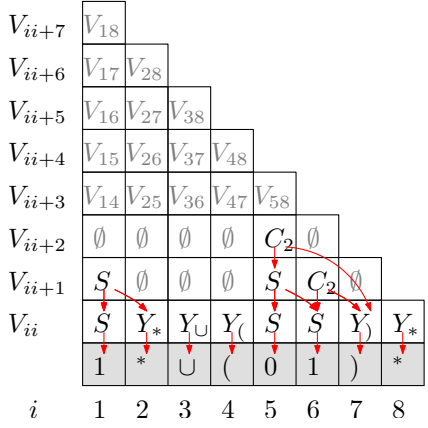
Wir testen, ob das Wort

$$w = 1^* \cup (01)^*$$

von der Grammatik erzeugt werden kann.

CYK-Algorithmus – Beispiel

$$
\begin{array}{rcl}
S & \to & SY_* \mid SC_1 \mid \\
 & & SS \mid Y_(C_2 \mid \\
 & & e \mid \emptyset \mid 0 \mid 1 \\
C_1 & \to & Y_\cup S \\
C_2 & \to & SY_) \\
Y_* & \to & * \\
Y_\cup & \to & \cup \\
Y_(& \to & (\\
Y_) & \to &)
\end{array}
$$



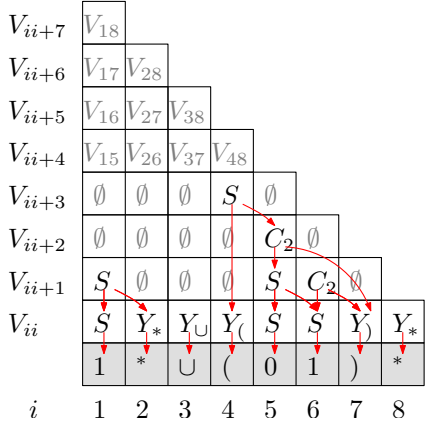
Wir testen, ob das Wort

$$w = 1^* \cup (01)^*$$

von der Grammatik erzeugt werden kann.

CYK-Algorithmus – Beispiel

$$
\begin{array}{rcl}
S & \to & SY_* \mid SC_1 \mid \\
 & & SS \mid Y_(C_2 \mid \\
 & & e \mid \emptyset \mid 0 \mid 1 \\
C_1 & \to & Y_\cup S \\
C_2 & \to & SY_) \\
Y_* & \to & * \\
Y_\cup & \to & \cup \\
Y_(& \to & (\\
Y_) & \to &)
\end{array}
$$



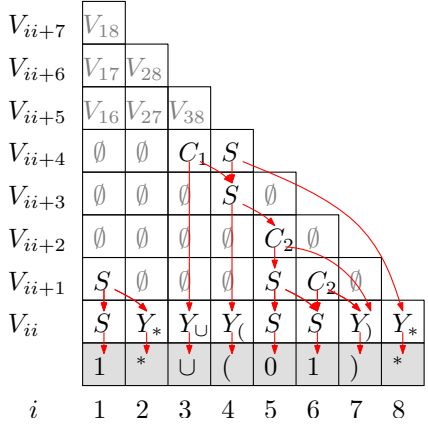
Wir testen, ob das Wort

$$w = 1^* \cup (01)^*$$

von der Grammatik erzeugt werden kann.

CYK-Algorithmus – Beispiel

$$
\begin{array}{rcl}
S & \to & SY_* \mid SC_1 \mid \\
 & & SS \mid Y_(C_2 \mid \\
 & & e \mid \emptyset \mid 0 \mid 1 \\
C_1 & \to & Y_\cup S \\
C_2 & \to & SY_) \\
Y_* & \to & * \\
Y_\cup & \to & \cup \\
Y_(& \to & (\\
Y_) & \to &)
\end{array}
$$



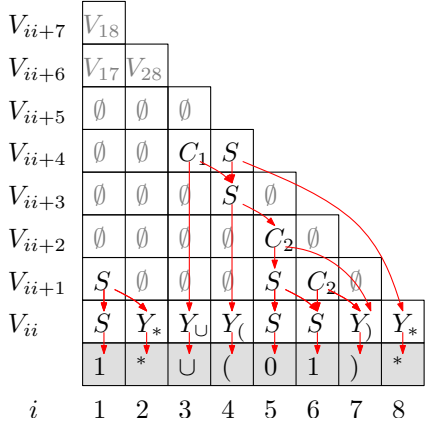
Wir testen, ob das Wort

$$w = 1^* \cup (01)^*$$

von der Grammatik erzeugt werden kann.

CYK-Algorithmus – Beispiel

$$
\begin{array}{rcl}
S & \to & SY_* \mid SC_1 \mid \\
 & & SS \mid Y_(C_2 \mid \\
 & & e \mid \emptyset \mid 0 \mid 1 \\
C_1 & \to & Y_\cup S \\
C_2 & \to & SY_) \\
Y_* & \to & * \\
Y_\cup & \to & \cup \\
Y_(& \to & (\\
Y_) & \to &)
\end{array}
$$



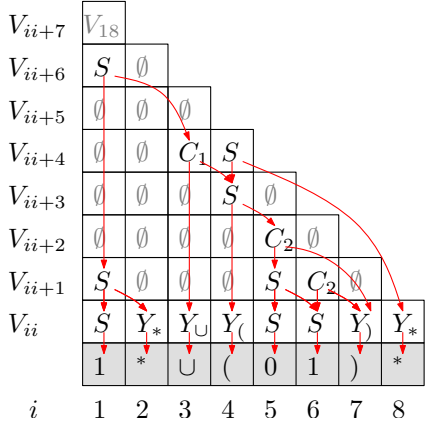
Wir testen, ob das Wort

$$w = 1^* \cup (01)^*$$

von der Grammatik erzeugt werden kann.

CYK-Algorithmus – Beispiel

$$
\begin{array}{rcl}
S & \to & SY_* \mid SC_1 \mid \\
 & & SS \mid Y_(C_2 \mid \\
 & & e \mid \emptyset \mid 0 \mid 1 \\
C_1 & \to & Y_\cup S \\
C_2 & \to & SY_) \\
Y_* & \to & * \\
Y_\cup & \to & \cup \\
Y_(& \to & (\\
Y_) & \to &)
\end{array}
$$



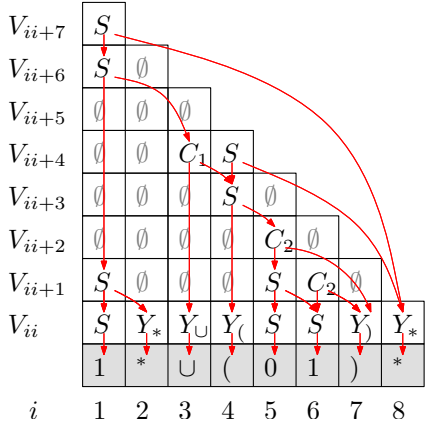
Wir testen, ob das Wort

$$w = 1^* \cup (01)^*$$

von der Grammatik erzeugt werden kann.

CYK-Algorithmus – Beispiel

$$
\begin{array}{rcl}
S & \to & SY_* \mid SC_1 \mid \\
 & & SS \mid Y_(C_2 \mid \\
 & & e \mid \emptyset \mid 0 \mid 1 \\
C_1 & \to & Y_\cup S \\
C_2 & \to & SY_) \\
Y_* & \to & * \\
Y_\cup & \to & \cup \\
Y_(& \to & (\\
Y_) & \to &)
\end{array}
$$



Wir testen, ob das Wort

$$w = 1^* \cup (01)^*$$

von der Grammatik erzeugt werden kann.

Ergebnisse zum Wortproblem

- **Typ-0 Grammatik.**
  Das Wortproblem ist nicht entscheidbar.
- **Typ-1 Grammatik.**
  Das Wortproblem ist $\mathcal{NP}$-vollständig.
- **Typ-2 Grammatik.**
  Das Wortproblem ist in polynomieller Zeit lösbar.
- **Typ-3 Grammatik.**
  Das Wortproblem ist in linearer Zeit lösbar.