TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut für Künstliche Intelligenz, Professur für Computational Logic

FORMALE SYSTEME

12. Vorlesung: Das Wortproblem für kontextfreie Sprachen

Hannes Straß

TU Dresden, 18. November 2021

Ableitungen in CFGs als Bäume

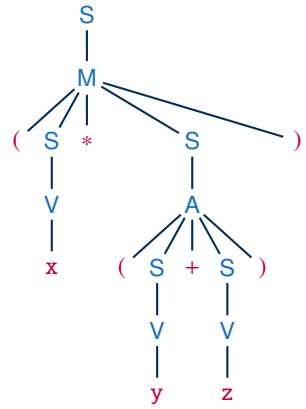
Grammatik:

$S \to A \mid M \mid V \quad A \to (S{+}S)$

$M \to (S{*}S) \quad V \to \mathtt{x} \mid \mathtt{y} \mid \mathtt{z}$

Ableitung:

$S \Rightarrow M \Rightarrow (S{*}S) \Rightarrow (V{*}S)$
$\Rightarrow (\mathtt{x}{*}S) \Rightarrow (\mathtt{x}{*}A)$
$\Rightarrow (\mathtt{x}{*}(S{+}S)) \Rightarrow (\mathtt{x}{*}(V{+}S))$
$\Rightarrow (\mathtt{x}{*}(\mathtt{y}{+}S)) \Rightarrow (\mathtt{x}{*}(\mathtt{y}{+}V))$
$\Rightarrow (\mathtt{x}{*}(\mathtt{y}{+}\mathtt{z}))$



Normalisierung kontextfreier Grammatiken

Motivation

Im Vergleich zu regulären Grammatiken erlauben CFGs wesentlich mehr Freiheiten bei der Formulierung von Produktionsregeln.

Für die Angabe von Algorithmen ist es hilfreich, die Grammatik erst in eine eingeschränktere Form zu überführen.

Beispiel: Wir haben in Vorlesung 2 gezeigt, wie man ϵ-Regeln in CFGs weitestgehend eliminieren kann, ohne dass sich dabei die Sprache ändert.

Es gibt mehrere sogenannte Normalformen, in die eine Grammatik durch derartige Vereinfachungen überführt werden kann.

Wiederholung: Erzeugung ϵ-freier CFGs

Der folgende Algorithmus kann ϵ-Regeln eliminieren.

Eingabe: CFG $G = \langle V, \Sigma, P, S \rangle$
Ausgabe: ϵ-freie CFG $G' = \langle V', \Sigma, P', S' \rangle$ mit $\mathbf{L}(G') = \mathbf{L}(G)$

- Initialisiere $P' := P$ und $V' := V$
- Berechne $V_\epsilon = \{A \in V \mid A \Rightarrow^* \epsilon\}$
  (einfaches rekursives Verfahren, siehe Vorlesung 2)
- Entferne alle ϵ-Regeln aus P'
- Solange es in P' eine Regel $B \to xAy$ gibt, mit $x, y \in (\Sigma \cup V)^*$ und
  $A \in V_\epsilon$ $\quad |x| + |y| \geq 1$ $\quad B \to xy \notin P'$
  wähle eine solche Regel und setze $P' := P' \cup \{B \to xy\}$
- Falls $S \in V_\epsilon$ dann definiere ein neues Startsymbol $S' \notin V$, setze $V' := V' \cup \{S'\}$ und $P' := P' \cup \{S' \to S, S' \to \epsilon\}$.
  Andernfalls ($S \notin V_\epsilon$) verwenden wir einfach $S' := S$ als Startsymbol.

Eliminierung von Kettenregeln

Eine Kettenregel ist eine Regel der Form $A \to B$.

Satz: Für jede CFG G gibt es eine äquivalente CFG G' ohne Kettenregeln, d.h. eine CFG G', für die $\mathbf{L}(G) = \mathbf{L}(G')$ gilt.

Beweis: Sei $G = \langle V, \Sigma, P, S \rangle$ ϵ-frei (o.B.d.A.).
G' ist die Grammatik $\langle V, \Sigma, P', S \rangle$, wobei sich P' wie folgt ergibt:

- Für jedes $A \in V$ bestimmen wir iterativ die Menge $E(A)$ aller $B \in V$, die man von A aus über Kettenregeln erreichen kann:
  (1) $A \in E(A)$
  (2) Falls $B \in E(A)$ und $B \to B' \in P$ mit $B' \in V$ dann $B' \in E(A)$
  (Schritt (2) wird wiederholt, bis keine Änderungen mehr auftreten.)
- Die Produktionsregeln von G' sind:

$$P' = \bigcup_{A \in V} \{A \to w \mid \text{es gibt } B \to w \in P \text{ mit } w \notin V \text{ und } B \in E(A)\}$$ □

Beispiel: Eliminierung von Kettenregeln

Grammatik G mit Kettenregeln:

S → A | M | V $\quad$ A → (S+S) $\quad$ M → (S∗S) $\quad$ V → x | y | z

$$P' = \bigcup_{A \in V} \{A \to w \mid \text{es gibt } B \to w \in P \text{ mit } w \notin V \text{ und } B \in E(A)\}$$

Mengen erreichbarer Variablen:

$E(\mathsf{S}) = \{\mathsf{S}, \mathsf{A}, \mathsf{M}, \mathsf{V}\}$ $\quad$ $E(\mathsf{A}) = \{\mathsf{A}\}$ $\quad$ $E(\mathsf{M}) = \{\mathsf{M}\}$ $\quad$ $E(\mathsf{V}) = \{\mathsf{V}\}$

Grammatik ohne Kettenregeln:

S → (S+S) | (S∗S) | x | y | z
A → (S+S)
M → (S∗S)
V → x | y | z

Quiz: Eliminierung von Kettenregeln

Quiz: Gegeben sei folgende CFG $G = \langle V, \Sigma, P, \mathsf{S} \rangle$ mit $\Sigma = \{\mathtt{0}, \mathtt{1}\}$, ...

Die Chomsky-Normalform

Für die algorithmische Verarbeitung kontextfreier Grammatiken ist die folgende Form besonders praktisch:

Eine kontextfreie Grammatik $G = \langle V, \Sigma, P, S \rangle$ ist in Chomsky-Normalform (CNF), wenn alle ihre Produktionsregeln eine der beiden folgenden Formen haben:

$$A \to BC \quad \text{(mit } B, C \in V\text{)} \qquad \text{oder} \qquad A \to c \quad \text{(mit } c \in \Sigma\text{)}$$

Beobachtung: Ist CFG G in Chomsky-Normalform, so gilt $\epsilon \notin \mathbf{L}(G)$.

- $\rightsquigarrow$ Grammatiken, deren Sprache das leere Wort enthalten, können nicht in CNF sein.
- $\rightsquigarrow$ Aber wie wir wissen, gibt es in ϵ-freien CFGs ohnehin höchstens eine Regel $S \to \epsilon$.
- $\rightsquigarrow$ Dieser Sonderfall kann in Algorithmen leicht berücksichtigt werden.
- $\rightsquigarrow$ Wir verzichten darauf, die CNF dafür noch zu erweitern.

Umwandlung in Chomsky-NF

Satz: Für jede Typ-2-Grammatik G mit $\epsilon \notin \mathbf{L}(G)$ gibt es eine äquivalente Typ-2-Grammatik CNF(G) in Chomsky-Normalform, d.h. so dass $\mathbf{L}(G) = \mathbf{L}(\text{CNF}(G))$.

Beweis: Wir haben bereits gezeigt:

(1) Durch Eliminierung von ϵ-Regeln kann jede CFG in eine äquivalente ϵ-freie CFG umgewandelt werden.

(2) Durch Eliminierung von Kettenregeln kann jede ϵ-freie CFG in eine äquivalente CFG umgewandelt werden, in der alle Regeln die Form $A \to w$ haben, mit $w \in \Sigma$ oder $|w| \geq 2$.

Zwei weitere Schritte führen zur Chomsky-NF:

(3) Extrahiere Regeln der Form $A \to c$, so dass alle anderen Regeln $B \to w$ keine Terminale mehr in w enthalten.

(4) Zerlege Regeln der Form $A \to B_1 \cdots B_n$ für $n > 2$, so dass solche Regeln nur noch mit $n = 2$ auftauchen.

Umwandlung in Chomsky-NF (2)

Satz: Für jede CFG G mit $\epsilon \notin \mathbf{L}(G)$ gibt es eine äquivalente CFG CNF(G) in Chomsky-Normalform, d.h. so dass $\mathbf{L}(G) = \mathbf{L}(\text{CNF}(G))$.

Beweis: Schritt (3): Extrahiere Regeln der Form $A \to c$, so dass alle anderen Regeln $B \to w$ keine Terminale mehr in w enthalten.

- Führe für jedes Symbol $a \in \Sigma$ eine neue Variable V_a und eine Regel $V_a \to a$ ein.
- Für jede Produktionsregel $A \to w$ mit $|w| > 1$:
  Ersetze jedes Vorkommen eines Symbols $a \in \Sigma$ in w durch V_a.

Anmerkungen:
Regeln $A \to w$ mit $|w| = 0$ existieren nicht (ϵ-Freiheit).
Regeln $A \to w$ mit $|w| = 1$ haben die Form $A \to c$ (keine Kettenregeln).

Umwandlung in Chomsky-NF (3)

Satz: Für jede Typ-2-Grammatik G mit $\epsilon \notin \mathbf{L}(G)$ gibt es eine äquivalente Typ-2-Grammatik CNF(G) in Chomsky-Normalform, d.h. so dass $\mathbf{L}(G) = \mathbf{L}(\text{CNF}(G))$.

Beweis: Schritt (4): Zerlege Regeln der Form $A \to B_1 \cdots B_n$ für $n > 2$, so dass solche Regeln nur noch mit $n = 2$ auftauchen.

Für jede Produktionsregel $A \to B_1 \cdots B_n$ mit $n > 2$:

- Führe $n - 2$ neue Variablen $C_1, \ldots, C_{n-2}$ ein (intuitiv: C_i steht für $B_{i+1} \cdots B_n$).
- Ersetze die Regel durch neue Regeln:

$$\begin{aligned} A &\to B_1 C_1 \\ C_1 &\to B_2 C_2 \\ &\vdots \\ C_{n-3} &\to B_{n-2} C_{n-2} \\ C_{n-2} &\to B_{n-1} B_n \end{aligned}$$

Umwandlung in Chomsky-NF (4)

Satz: Für jede Typ-2-Grammatik G mit $\epsilon \notin \mathbf{L}(G)$ gibt es eine äquivalente Typ-2-Grammatik CNF(G) in Chomsky-Normalform, d.h. so dass $\mathbf{L}(G) = \mathbf{L}(\text{CNF}(G))$.

Beweis: Die Korrektheit der Schritte lässt sich zeigen, indem man für eine beliebige Ableitung einer Grammatik G eine Ableitung der umgeformten Grammatik CNF(G) angibt.

Zum Beispiel für Schritt 4:

Eine Anwendung der Regel $A \to B_1 \cdots B_n$ wird dargestellt durch die Ableitungen

$$A \Rightarrow B_1C_1 \Rightarrow B_1B_2C_2 \Rightarrow \ldots \Rightarrow B_1 \cdots B_{n\text{-}2}C_{n\text{-}2} \Rightarrow B_1 \cdots B_{n\text{-}2}B_{n\text{-}1}B_n$$

Korrektheit von Schritt 3 ist noch einfacher. □

Beispiel

ϵ-freie CFG ohne Kettenregeln:

$$S \to (S{+}S) \mid (S{*}S) \mid \mathtt{x} \mid \mathtt{y} \mid \mathtt{z}$$

Schritt (3): Extrahiere Regeln der Form $A \to \mathtt{c}$, so dass alle anderen Regeln $B \to w$ keine Terminale mehr in w enthalten:

$$S \to V_(\; S \; V_+ \; S \; V_) \mid V_(\; S \; V_* \; S \; V_) \mid \mathtt{x} \mid \mathtt{y} \mid \mathtt{z}$$

$$V_(\to (\quad V_+ \to + \quad V_) \to) \quad V_* \to * \quad \underbrace{V_\mathtt{x} \to \mathtt{x} \quad V_\mathtt{y} \to \mathtt{y} \quad V_\mathtt{z} \to \mathtt{z}}_{\text{ungenutzt (dürfen entfallen)}}$$

Schritt (4): Zerlege Regeln der Form $A \to B_1 \cdots B_n$ für $n > 2$, so dass solche Regeln nur noch mit $n = 2$ auftauchen:

$$S \to V_(\; C_1 \mid V_(\; D_1 \mid \mathtt{x} \mid \mathtt{y} \mid \mathtt{z}$$

$$C_1 \to SC_2 \qquad C_2 \to V_+C_3 \qquad C_3 \to SV_)$$

$$D_1 \to SD_2 \qquad D_2 \to V_*D_3 \qquad D_3 \to SV_)$$

$$V_(\to (\quad V_+ \to + \quad V_) \to) \quad V_* \to *$$

Ableitungsbäume für CNF-Grammatiken

Besonderheit der Chomsky-Normalform: Alle Ableitungsbäume sind im Inneren Binärbäume und haben daher eine sehr reguläre Struktur:

- Bei einem Wort w der Länge $|w|$ werden genau $|w|$ Regeln vom Typ $A \to \mathtt{c}$ angewendet.
  - $\rightsquigarrow$ Es gibt in der Ebene darüber genau $|w|$ Variablen.
- Jede Anwendung einer Regel vom Typ $A \to BC$ erhöht die Anzahl der am Ende zu ersetzenden Variablen um 1.
  - $\rightsquigarrow$ Für ein Wort der Länge $|w|$ müssen genau $|w| - 1$ Regeln vom Typ $A \to BC$ angewendet werden.

Wir folgern:

Satz: Ist G eine Grammatik in Chomsky-Normalform und $w \in \mathbf{L}(G)$, so hat jede Ableitung von w in G genau $2|w| - 1$ Ableitungsschritte.

Das Wortproblem für CFGs

Das Wortproblem für CFGs

Das Wortproblem für eine Sprache **L** über Alphabet Σ besteht darin, die folgende Funktion zu berechnen:

Eingabe: ein Wort $w \in \Sigma^*$
Ausgabe: „ja“ wenn $w \in$ **L** und „nein“ wenn $w \notin$ **L**

Für Typ-2-Sprachen **L** können wir bereits ein (ineffizientes) Entscheidungsverfahren angeben:

- Es gibt eine CFG für **L**.
- Also gibt es eine CFG in Chomsky-NF für **L**.
- Also hat jedes Wort $w \in$ **L** eine Ableitung der Länge $2|w| - 1$.
- Wir können systematisch alle Ableitungen dieser Länge betrachten und prüfen, ob eine davon w erzeugt.

$\rightsquigarrow$ exponentieller Algorithmus

Geht es auch besser?

Quiz: Wortproblem für CFGs

Quiz: Wir betrachten die kontextfreie Grammatik $G = \langle V, \Sigma, P, \mathrm{S} \rangle$ mit $\Sigma = \{\mathtt{0}, \mathtt{1}\}$ und ...

Sakai, Kasami, Younger, Cocke und Schwartz

Verschiedene Menschen fanden eine bessere Lösung (besser als der naive, exponentielle Algorithmus):

- Itiroo Sakai, 1961: „Syntax in universal translation“
- Tadao Kasami, 1965: „An efficient recognition and syntax-analysis algorithm for context-free languages“
- Daniel H. Younger, 1967: „Recognition and parsing of context-free languages in time n^3“
- John Cocke, Jacob T. Schwartz, 1970: „Programming languages and their compilers“

Das Ergebnis ist bekannt als CYK-Algorithmus (Cocke-Younger-Kasami).

CYK: Grundidee

Der CYK-Algorithmus arbeitet mit einer kontextfreien Grammatik G in Chomsky-NF.

Wie kann man prüfen, ob ein Wort $w = \mathtt{a}_1 \cdots \mathtt{a}_n$ durch so eine Grammatik abgeleitet werden kann?

- Falls $|w| = 1$, dann ist $w \in \Sigma$ und es gilt:
  $w \in \mathbf{L}(G)$ genau dann, wenn es eine Regel $\mathrm{S} \to w$ in G gibt.
- Falls $|w| > 1$, dann ist:
  $w \in \mathbf{L}(G)$ genau dann, wenn es eine Regel $\mathrm{S} \to \mathrm{AB}$ und einen Index $i \in \{1, \ldots, n-1\}$ gibt, so dass gilt:

$$\mathrm{A} \Rightarrow^* \mathtt{a}_1 \cdots \mathtt{a}_i \qquad \text{und} \qquad \mathrm{B} \Rightarrow^* \mathtt{a}_{i+1} \cdots \mathtt{a}_n$$

Idee: Fall 2 reduziert das Problem $\mathrm{S} \overset{?}{\Rightarrow}^* \mathtt{a}_1 \cdots \mathtt{a}_n$ auf zwei einfachere Probleme $\mathrm{A} \overset{?}{\Rightarrow}^* \mathtt{a}_1 \cdots \mathtt{a}_i$ und $\mathrm{B} \overset{?}{\Rightarrow}^* \mathtt{a}_{i+1} \cdots \mathtt{a}_n$, die man allerdings für alle Regeln $\mathrm{S} \to \mathrm{AB}$ und Indizes $i \in \{1, \ldots, n-1\}$ lösen muss.

CYK: Praktische Umsetzung

Notation: Für $w = a_1 \cdots a_n$ schreiben wir $w_{i,j}$ für das Teilwort $a_i \cdots a_j$, also das Infix der Länge $j - i + 1$, welches an Position i beginnt.

Vorgehen: Wir berechnen für jedes Teilwort $w_{i,j}$ die Menge aller Variablen A, für die $A \Rightarrow^* w_{i,j}$ gilt. Diese Menge nennen wir $V[i,j]$.

- Wir beginnen mit den kürzesten Teilwörtern (Länge 1), also dem Fall $i = j$.
- Für längere Wörter betrachten wir jede mögliche Zweiteilung $w_{i,j} = w_{i,k}w_{k+1,j}$ und suchen Regeln der Form $A \to BC$, so dass $B \in V[i,k]$ und $C \in V[k+1,j]$. (Dann gilt nämlich $A \in V[i,j]$.)

Ist am Ende das Startsymbol $S \in V[1,|w|]$, dann liegt w in der Sprache.

Notation

Für die Teilwörter $w_{i,j}$ gilt $i \leq j$.
$\leadsto$ Die Mengen $V[i,j]$ können als Dreiecksmatrix notiert werden.

Beispiel: Wir betrachten das Wort $w = a + b * c$ der Länge $|w| = 5$.

Darstellung der Mengen $V[i,j]$:

a	$V[1,1]$	$V[1,2]$	$V[1,3]$	$V[1,4]$	$V[1,5]$
+		$V[2,2]$	$V[2,3]$	$V[2,4]$	$V[2,5]$
b			$V[3,3]$	$V[3,4]$	$V[3,5]$
*				$V[4,4]$	$V[4,5]$
c					$V[5,5]$
	a	+	b	*	c

Beispiel

Grammatik:

S → SA | SM | a | b | c
A → PS M → TS P → + T → *

Wort: $w = a + b * c$

Berechnung der Mengen $V[i,j]$:

a	S		S		S
+		P	A		A
b			S		S
*				T	M
c					S
	a	+	b	*	c

$S \in V[1,5] \leadsto w$ kann erzeugt werden

Der CYK-Algorithmus

Eingabe: Wort $w = a_1 \cdots a_n$, CNF-Grammatik $G = \langle V, \Sigma, P, S \rangle$
Ausgabe: „ja“ wenn $w \in \mathbf{L}(G)$; sonst „nein“

for $i = 1, \ldots, n$: // Teilwörter der Länge 1
  $V[i,i] := \{A \in V \mid A \to a_i \in P\}$

for $d = 1, \ldots, n-1$: // Differenz Endindex − Startindex
  for $i = 1, \ldots, n-d$: // Startindex
    $j := i + d$
    $V[i,j] := \emptyset$
    for $k = i, \ldots, j-1$: // Trennindex
      $V[i,j] := V[i,j] \cup \{A \in V \mid$ es gibt eine Regel $A \to BC \in P$ mit $B \in V[i,k]$ und $C \in V[k+1,j]\}$

return $S \in V[1,n]$ **?** „ja“ **:** „nein“

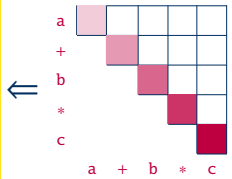
Bezug zur Tabelle

Eingabe: Wort $w = a_1 \cdots a_n$,
CNF-Grammatik $G = \langle V, \Sigma, P, S \rangle$
Ausgabe: „ja“ wenn $w \in \mathbf{L}(G)$; sonst „nein“

for $i = 1, \ldots, n$: // Teilwörter der Länge 1
  $V[i,i] := \{A \in V \mid A \to a_i \in P\}$

for $d = 1, \ldots, n-1$: // Differenz Endind. – Startind.
  for $i = 1, \ldots, n-d$: // Startindex
    $j := i + d$
    $V[i,j] := \emptyset$
    for $k = i, \ldots, j-1$: // Trennindex
      $V[i,j] := V[i,j] \cup$
        $\{A \in V \mid$ es gibt $A \to BC \in P$ mit
          $B \in V[i,k]$ und $C \in V[k+1,j]\}$

return $S \in V[1,n]$ **?** „ja“ **:** „nein“



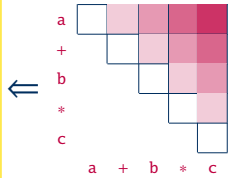
Bezug zur Tabelle

Eingabe: Wort $w = a_1 \cdots a_n$,
CNF-Grammatik $G = \langle V, \Sigma, P, S \rangle$
Ausgabe: „ja“ wenn $w \in \mathbf{L}(G)$; sonst „nein“

for $i = 1, \ldots, n$: // Teilwörter der Länge 1
  $V[i,i] := \{A \in V \mid A \to a_i \in P\}$

for $d = 1, \ldots, n-1$: // Differenz Endind. – Startind.
  for $i = 1, \ldots, n-d$: // Startindex
    $j := i + d$
    $V[i,j] := \emptyset$
    for $k = i, \ldots, j-1$: // Trennindex
      $V[i,j] := V[i,j] \cup$
        $\{A \in V \mid$ es gibt $A \to BC \in P$ mit
          $B \in V[i,k]$ und $C \in V[k+1,j]\}$

return $S \in V[1,n]$ **?** „ja“ **:** „nein“



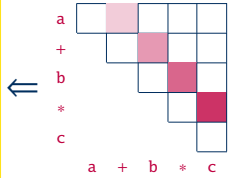
Bezug zur Tabelle

Eingabe: Wort $w = a_1 \cdots a_n$,
CNF-Grammatik $G = \langle V, \Sigma, P, S \rangle$
Ausgabe: „ja“ wenn $w \in \mathbf{L}(G)$; sonst „nein“

for $i = 1, \ldots, n$: // Teilwörter der Länge 1
  $V[i,i] := \{A \in V \mid A \to a_i \in P\}$

for $d = 1, \ldots, n-1$: // Differenz Endind. – Startind.
  for $i = 1, \ldots, n-d$: // Startindex
    $j := i + d$
    $V[i,j] := \emptyset$
    for $k = i, \ldots, j-1$: // Trennindex
      $V[i,j] := V[i,j] \cup$
        $\{A \in V \mid$ es gibt $A \to BC \in P$ mit
          $B \in V[i,k]$ und $C \in V[k+1,j]\}$

return $S \in V[1,n]$ **?** „ja“ **:** „nein“



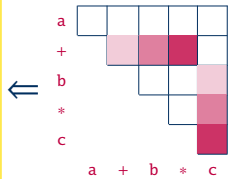
Bezug zur Tabelle

Eingabe: Wort $w = a_1 \cdots a_n$,
CNF-Grammatik $G = \langle V, \Sigma, P, S \rangle$
Ausgabe: „ja“ wenn $w \in \mathbf{L}(G)$; sonst „nein“

for $i = 1, \ldots, n$: // Teilwörter der Länge 1
  $V[i,i] := \{A \in V \mid A \to a_i \in P\}$

for $d = 1, \ldots, n-1$: // Differenz Endind. – Startind.
  for $i = 1, \ldots, n-d$: // Startindex
    $j := i + d$
    $V[i,j] := \emptyset$
    for $k = i, \ldots, j-1$: // Trennindex
      $V[i,j] := V[i,j] \cup$
        $\{A \in V \mid$ es gibt $A \to BC \in P$ mit
          $B \in V[i,k]$ und $C \in V[k+1,j]\}$

return $S \in V[1,n]$ **?** „ja“ **:** „nein“



Komplexität

Wie komplex ist der CYK-Algorithmus?

- Es gibt drei geschachtelte Schleifen mit Lauflänge $O(|w|)$.
- Operationen für Zugriff auf Grammatik und Teilmengen von V:
  - polynomiell bezüglich der Größe der Grammatik;
  - konstant, wenn die Grammatik vorher bekannt und fest ist.

Satz: Das Wortproblem für kontextfreie Sprachen ist in $O(n^3)$ lösbar.

Anmerkung: Die Umwandlung in Chomsky-NF verlangt $\epsilon \notin \mathbf{L}(G)$, aber im Fall $\epsilon \in \mathbf{L}(G)$ kann man eine Grammatik G' mit $\epsilon \notin \mathbf{L}(G')$ erzeugen und $w = \epsilon$ gesondert testen.

Zusammenfassung und Ausblick

Jede Typ-2-Grammatik kann in eine äquivalente* Grammatik in Chomsky-Normalform umgewandelt werden.

* Besser: äquivalent bis auf das leere Wort, welches von der normalisierten Grammatik nicht erzeugt wird.

Mit Hilfe des CYK-Algorithmus ist das Wortproblem für kontextfreie Sprachen in polynomieller Zeit lösbar.

Offene Fragen:

- Haben kontextfreie Sprachen ein Berechnungsmodell?
- Wie sehen nicht-kontextfreie Sprachen aus und wie erkennt man sie?