SS 2005

Einführung in die Informatik IV

Ernst W. Mayr

Fakultät für Informatik
TU München

http://www14.in.tum.de/lehre/2005SS/info4/index.html.de

6. Mai 2005

Schnittproblem: Ist $L(G_1) \cap L(G_2) = \emptyset$?

Das Schnittproblem ist nur für die Familie der regulären Sprachen entscheidbar, nicht aber für die Familie der Chomsky-2-Sprachen.

Äquivalenzproblem: Ist $L(G_1) = L(G_2)$?

Schnittproblem: Ist $L(G_1) \cap L(G_2) = \emptyset$?

Das Schnittproblem ist nur für die Familie der regulären Sprachen entscheidbar, nicht aber für die Familie der Chomsky-2-Sprachen.

Äquivalenzproblem: Ist $L(G_1) = L(G_2)$?

Das Äquivalenzproblem lässt sich auch wie folgt formulieren:

$$L_1 = L_2 \quad \Leftrightarrow \quad (L_1 \cap \overline{L_2}) \cup (L_2 \cap \overline{L_1}) = \emptyset$$

Schnittproblem: Ist $L(G_1) \cap L(G_2) = \emptyset$?

Das Schnittproblem ist nur für die Familie der regulären Sprachen entscheidbar, nicht aber für die Familie der Chomsky-2-Sprachen.

Äquivalenzproblem: Ist $L(G_1) = L(G_2)$?

Das Äquivalenzproblem lässt sich auch wie folgt formulieren:

$$L_1 = L_2 \quad \Leftrightarrow \quad (L_1 \cap \overline{L_2}) \cup (L_2 \cap \overline{L_1}) = \emptyset$$

Schnittproblem: Ist $L(G_1) \cap L(G_2) = \emptyset$?

Das Schnittproblem ist nur für die Familie der regulären Sprachen entscheidbar, nicht aber für die Familie der Chomsky-2-Sprachen.

Äquivalenzproblem: Ist $L(G_1) = L(G_2)$?

Das Äquivalenzproblem lässt sich auch wie folgt formulieren:

$$L_1 = L_2 \quad \Leftrightarrow \quad (L_1 \cap \overline{L_2}) \cup (L_2 \cap \overline{L_1}) = \emptyset$$

Schnittproblem: Ist $L(G_1) \cap L(G_2) = \emptyset$?

Das Schnittproblem ist nur für die Familie der regulären Sprachen entscheidbar, nicht aber für die Familie der Chomsky-2-Sprachen.

Äquivalenzproblem: Ist $L(G_1) = L(G_2)$?

Das Äquivalenzproblem lässt sich auch wie folgt formulieren:

$$L_1 = L_2 \quad \Leftrightarrow \quad (L_1 \cap \overline{L_2}) \cup (L_2 \cap \overline{L_1}) = \emptyset$$

Schnittproblem: Ist $L(G_1) \cap L(G_2) = \emptyset$?

Das Schnittproblem ist nur für die Familie der regulären Sprachen entscheidbar, nicht aber für die Familie der Chomsky-2-Sprachen.

Äquivalenzproblem: Ist $L(G_1) = L(G_2)$?

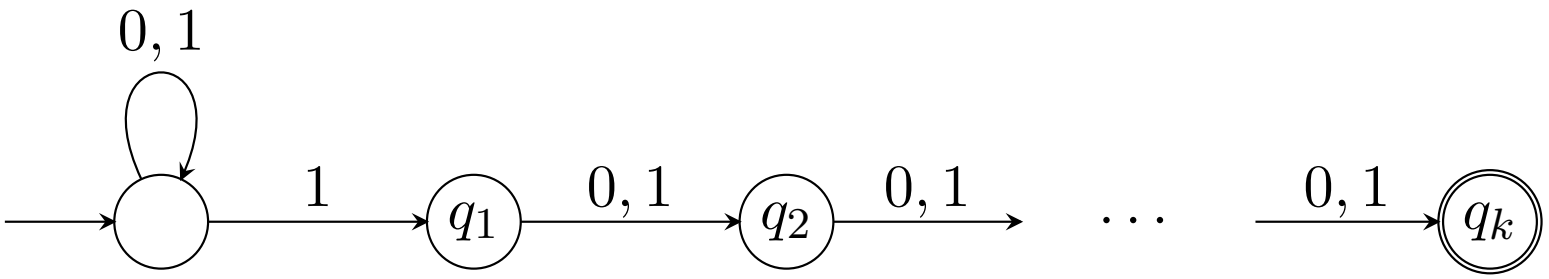
Das Äquivalenzproblem lässt sich auch wie folgt formulieren:

$$L_1 = L_2 \quad \Leftrightarrow \quad (L_1 \cap \overline{L_2}) \cup (L_2 \cap \overline{L_1}) = \emptyset$$

Wichtig für eine effiziente Lösung der Probleme ist, wie die Sprache gegeben ist. Hierzu ein Beispiel:

Beispiel 57

$L = \{w \in \{0,1\}^*$; das k-letzte Bit von w ist gleich $1\}$
Ein NFA für diese Sprache ist gegeben durch:

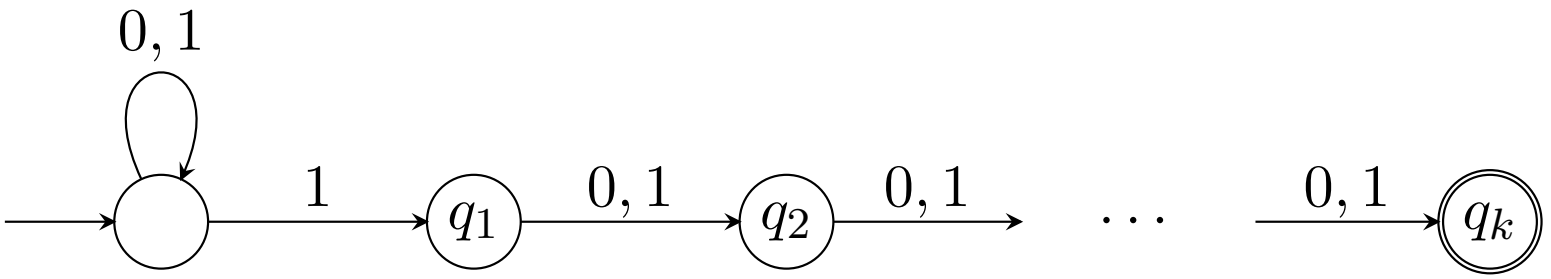


Insgesamt hat der NFA $k+1$ Zustände. Man kann nun diesen NFA in einen deterministischen Automaten umwandeln und stellt fest, dass der entsprechende DFA $\Omega(2^k)$ Zustände hat.

Wichtig für eine effiziente Lösung der Probleme ist, wie die Sprache gegeben ist. Hierzu ein Beispiel:

Beispiel 57

$L = \{w \in \{0,1\}^*$; das k-letzte Bit von w ist gleich $1\}$
Ein NFA für diese Sprache ist gegeben durch:



Insgesamt hat der NFA $k+1$ Zustände. Man kann nun diesen NFA in einen deterministischen Automaten umwandeln und stellt fest, dass der entsprechende DFA $\Omega(2^k)$ Zustände hat.

Da die Komplexität eines Algorithmus von der Größe der Eingabe abhängt, ist dieser Unterschied in der Eingabegröße natürlich wesentlich, denn es gilt:

kurze Eingabe wie beim NFA $\Rightarrow$ wenig Zeit für einen effizienten Algorithmus,

lange Eingabe wie beim DFA $\Rightarrow$ mehr Zeit für einen effizienten Algorithmus.

Da die Komplexität eines Algorithmus von der Größe der Eingabe abhängt, ist dieser Unterschied in der Eingabegröße natürlich wesentlich, denn es gilt:

kurze Eingabe wie beim NFA $\Rightarrow$ wenig Zeit für einen effizienten Algorithmus,

lange Eingabe wie beim DFA $\Rightarrow$ mehr Zeit für einen effizienten Algorithmus.

Da die Komplexität eines Algorithmus von der Größe der Eingabe abhängt, ist dieser Unterschied in der Eingabegröße natürlich wesentlich, denn es gilt:

kurze Eingabe wie beim NFA $\Rightarrow$ wenig Zeit für einen effizienten Algorithmus,

lange Eingabe wie beim DFA $\Rightarrow$ mehr Zeit für einen effizienten Algorithmus.

4. Kontextfeie Grammatiken und Sprachen

4.1 Grundlagen und ein Beispiel

Sei

$$L_= := \{w \in \{0,1\}^*;\ w \text{ enthält gleich viele 0en und 1en}\}.$$

Sei $\#_a(w)$ die Anzahl der Zeichen a in der Zeichenreihe w, d.h.

$$L_= = \{w \in \{0,1\}^*;\ \#_0(w) = \#_1(w)\}.$$

$L_=$ ist sicherlich nicht regulär (vgl. Pumping-Lemma).

4. Kontextfeie Grammatiken und Sprachen

4.1 Grundlagen und ein Beispiel

Sei

$$L_= := \{w \in \{0,1\}^*;\ w \text{ enthält gleich viele 0en und 1en}\}.$$

Sei $\#_a(w)$ die Anzahl der Zeichen a in der Zeichenreihe w, d.h.

$$L_= = \{w \in \{0,1\}^*;\ \#_0(w) = \#_1(w)\}.$$

$L_=$ ist sicherlich nicht regulär (vgl. Pumping-Lemma).

4. Kontextfeie Grammatiken und Sprachen

4.1 Grundlagen und ein Beispiel

Sei

$$L_= := \{w \in \{0,1\}^*;\ w \text{ enthält gleich viele 0en und 1en}\}.$$

Sei $\#_a(w)$ die Anzahl der Zeichen a in der Zeichenreihe w, d.h.

$$L_= = \{w \in \{0,1\}^*;\ \#_0(w) = \#_1(w)\}.$$

$L_=$ ist sicherlich nicht regulär (vgl. Pumping-Lemma).

Satz 58

Die (kontextfreie) Grammatik G

$$S \to \epsilon \mid T$$
$$T \to TT \mid 0T1 \mid 1T0 \mid 01 \mid 10$$

erzeugt $L_=$.

Beweis:

Sei $w \in L_=$. Betrachte für jedes Präfix x von w die Zahl

$$\#_1(x) - \#_0(x) \ .$$

Falls $w = w'w''$ für nichtleere $w', w'' \in L_=$, wende man Induktion über $|w|$ an, falls nicht, ist w von der Form $0w'1$ oder $1w'0$, und Induktion liefert wiederum die Behauptung. □

Satz 58

Die (kontextfreie) Grammatik G

$$S \to \epsilon \mid T$$
$$T \to TT \mid 0T1 \mid 1T0 \mid 01 \mid 10$$

erzeugt $L_=$.

Beweis:

Sei $w \in L_=$. Betrachte für jedes Präfix x von w die Zahl

$$\#_1(x) - \#_0(x) \ .$$

Falls $w = w'w''$ für nichtleere $w', w'' \in L_=$, wende man Induktion über $|w|$ an, falls nicht, ist w von der Form $0w'1$ oder $1w'0$, und Induktion liefert wiederum die Behauptung. □

Definition 59 (Wiederholung)

- Eine kontextfreie Grammatik G heißt eindeutig, wenn es für jedes $w \in L(G)$ genau einen Ableitungsbaum gibt.
- Eine kontextfreie Sprache L heißt eindeutig, falls es eine eindeutige kontextfreie Grammatik G mit $L = L(G)$ gibt. Ansonsten heißt L inhärent mehrdeutig.

Die oben angegebene Grammatik für $L_=$ ist nicht eindeutig.

Definition 59 (Wiederholung)

- Eine kontextfreie Grammatik G heißt eindeutig, wenn es für jedes $w \in L(G)$ genau einen Ableitungsbaum gibt.
- Eine kontextfreie Sprache L heißt eindeutig, falls es eine eindeutige kontextfreie Grammatik G mit $L = L(G)$ gibt. Ansonsten heißt L inhärent mehrdeutig.

Die oben angegebene Grammatik für $L_=$ ist nicht eindeutig.

Definition 59 (Wiederholung)

- Eine kontextfreie Grammatik G heißt eindeutig, wenn es für jedes $w \in L(G)$ genau einen Ableitungsbaum gibt.
- Eine kontextfreie Sprache L heißt eindeutig, falls es eine eindeutige kontextfreie Grammatik G mit $L = L(G)$ gibt. Ansonsten heißt L inhärent mehrdeutig.

Die oben angegebene Grammatik für $L_=$ ist nicht eindeutig.

Definition 59 (Wiederholung)

- Eine kontextfreie Grammatik G heißt eindeutig, wenn es für jedes $w \in L(G)$ genau einen Ableitungsbaum gibt.
- Eine kontextfreie Sprache L heißt eindeutig, falls es eine eindeutige kontextfreie Grammatik G mit $L = L(G)$ gibt. Ansonsten heißt L inhärent mehrdeutig.

Die oben angegebene Grammatik für $L_=$ ist nicht eindeutig.

4.2 Die Chomsky-Normalform

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

Definition 60

Eine kontextfreie Grammatik G ist in Chomsky-Normalform, falls alle Produktionen eine der Formen

$$\begin{aligned} A &\to a & A &\in V, a \in \Sigma, \\ A &\to BC & A, B, C &\in V, \text{ oder} \\ S &\to \epsilon \end{aligned}$$

haben.

Algorithmus zur Konstruktion einer (äquivalenten) Grammatik in Chomsky-Normalform

Eingabe: Eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$

1. Wir fügen für jedes $a \in \Sigma$ zu V ein neues Nichtterminal Y_a hinzu, ersetzen in allen Produktionen a durch Y_a und fügen $Y_a \to a$ als neue Produktion zu P hinzu.

   /* linearer Zeitaufwand, Größe vervierfacht sich höchstens */

2. Wir ersetzen jede Produktion der Form

   $$A \to B_1 B_2 \dots B_r \ (r \geq 3)$$

   durch

   $$A \to B_1 C_1,\ C_1 \to B_2 C_2,\ \dots,\ C_{r-2} \to B_{r-1} B_r$$

   wobei $C_1, \dots, C_{r-2}$ neue Nichtterminale sind.

   /* linearer Zeitaufwand, Größe vervierfacht sich höchstens */

Algorithmus zur Konstruktion einer (äquivalenten) Grammatik in Chomsky-Normalform

Eingabe: Eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$

1. Wir fügen für jedes $a \in \Sigma$ zu V ein neues Nichtterminal Y_a hinzu, ersetzen in allen Produktionen a durch Y_a und fügen $Y_a \to a$ als neue Produktion zu P hinzu.

   /* linearer Zeitaufwand, Größe vervierfacht sich höchstens */

2. Wir ersetzen jede Produktion der Form

   $$A \to B_1 B_2 \dots B_\ell \quad (\ell \geq 3)$$

   durch

   $$A \to B_1 C_2 \quad C_2 \to B_2 C_3 \quad \dots \quad C_{\ell-1} \to B_{\ell-1} B_\ell$$

   wobei $C_2, \dots, C_{\ell-1}$ neue Nichtterminale sind.

Algorithmus zur Konstruktion einer (äquivalenten) Grammatik in Chomsky-Normalform

Eingabe: Eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$

1. Wir fügen für jedes $a \in \Sigma$ zu V ein neues Nichtterminal Y_a hinzu, ersetzen in allen Produktionen a durch Y_a und fügen $Y_a \to a$ als neue Produktion zu P hinzu.

   /* linearer Zeitaufwand, Größe vervierfacht sich höchstens */

2. Wir ersetzen jede Produktion der Form

   $$A \to B_1 B_2 \cdots B_r \ (r \geq 3)$$

   durch

   $$A \to B_1 C_2,\ C_2 \to B_2 C_3,\ \ldots,\ C_{r-1} \to B_{r-1} B_r,$$

   wobei $C_2, \ldots, C_{r-1}$ neue Nichtterminale sind.

   /* linearer Zeitaufwand, Größe vervierfacht sich höchstens */

Algorithmus zur Konstruktion einer (äquivalenten) Grammatik in Chomsky-Normalform

Eingabe: Eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$

1. Wir fügen für jedes $a \in \Sigma$ zu V ein neues Nichtterminal Y_a hinzu, ersetzen in allen Produktionen a durch Y_a und fügen $Y_a \to a$ als neue Produktion zu P hinzu.

   /* linearer Zeitaufwand, Größe vervierfacht sich höchstens */

2. Wir ersetzen jede Produktion der Form

   $$A \to B_1 B_2 \cdots B_r \ (r \geq 3)$$

   durch

   $$A \to B_1 C_2,\ C_2 \to B_2 C_3, \ldots, C_{r-1} \to B_{r-1} B_r,$$

   wobei $C_2, \ldots, C_{r-1}$ neue Nichtterminale sind.

   /* linearer Zeitaufwand, Größe vervierfacht sich höchstens */

Algorithmus zur Konstruktion einer (äquivalenten) Grammatik in Chomsky-Normalform

Eingabe: Eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$

1. Wir fügen für jedes $a \in \Sigma$ zu V ein neues Nichtterminal Y_a hinzu, ersetzen in allen Produktionen a durch Y_a und fügen $Y_a \to a$ als neue Produktion zu P hinzu.

   /* linearer Zeitaufwand, Größe vervierfacht sich höchstens */

2. Wir ersetzen jede Produktion der Form

   $$A \to B_1 B_2 \cdots B_r \ (r \geq 3)$$

   durch

   $$A \to B_1 C_2,\ C_2 \to B_2 C_3, \ldots, C_{r-1} \to B_{r-1} B_r,$$

   wobei $C_2, \ldots, C_{r-1}$ neue Nichtterminale sind.

   /* linearer Zeitaufwand, Größe vervierfacht sich höchstens */

Algorithmus zur Konstruktion einer (äquivalenten) Grammatik in Chomsky-Normalform

Eingabe: Eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$

1. Wir fügen für jedes $a \in \Sigma$ zu V ein neues Nichtterminal Y_a hinzu, ersetzen in allen Produktionen a durch Y_a und fügen $Y_a \to a$ als neue Produktion zu P hinzu.

   /* linearer Zeitaufwand, Größe vervierfacht sich höchstens */

2. Wir ersetzen jede Produktion der Form

   $$A \to B_1 B_2 \cdots B_r \ (r \geq 3)$$

   durch

   $$A \to B_1 C_2,\ C_2 \to B_2 C_3, \ldots, C_{r-1} \to B_{r-1} B_r,$$

   wobei $C_2, \ldots, C_{r-1}$ neue Nichtterminale sind.

   /* linearer Zeitaufwand, Größe vervierfacht sich höchstens */

Algorithmus zur Konstruktion einer (äquivalenten) Grammatik in Chomsky-Normalform

Eingabe: Eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$

1. Wir fügen für jedes $a \in \Sigma$ zu V ein neues Nichtterminal Y_a hinzu, ersetzen in allen Produktionen a durch Y_a und fügen $Y_a \to a$ als neue Produktion zu P hinzu.

   /* linearer Zeitaufwand, Größe vervierfacht sich höchstens */

2. Wir ersetzen jede Produktion der Form

   $$A \to B_1 B_2 \cdots B_r \ (r \geq 3)$$

   durch

   $$A \to B_1 C_2,\ C_2 \to B_2 C_3, \ldots, C_{r-1} \to B_{r-1} B_r,$$

   wobei $C_2, \ldots, C_{r-1}$ neue Nichtterminale sind.

   /* linearer Zeitaufwand, Größe vervierfacht sich höchstens */

3. Für alle $A, B \in V$, $A \neq B$, mit

$$A \rightarrow^{+} B,$$

füge für jede Produktion der Form

$$A' \rightarrow \alpha A \in P$$

die Produktion

$$A' \rightarrow \alpha B$$

zu P hinzu.

/* quadratischer Aufwand */

4. Streiche alle Produktionen der Form $A \rightarrow \Pi$ aus P.

3. Für alle $A, B \in V$, $A \neq B$, mit

$$A \rightarrow^+ B,$$

füge für jede Produktion der Form

$$A' \rightarrow \alpha A \in P$$

die Produktion

$$A' \rightarrow \alpha B$$

zu P hinzu.

/* quadratischer Aufwand */

4. Streiche alle Produktionen der Form $A \rightarrow B$ aus P.

3. Für alle $A, B \in V$, $A \neq B$, mit

$$A \rightarrow^{+} B,$$

füge für jede Produktion der Form

$$A' \rightarrow \alpha A \in P$$

die Produktion

$$A' \rightarrow \alpha B$$

zu P hinzu.

/* quadratischer Aufwand */

4. Streiche alle Produktionen der Form $A \rightarrow B$ aus P.

3. Für alle $A, B \in V$, $A \neq B$, mit

$$A \rightarrow^{+} B,$$

füge für jede Produktion der Form

$$A' \rightarrow \alpha A \in P$$

die Produktion

$$A' \rightarrow \alpha B$$

zu P hinzu.

/* quadratischer Aufwand */

4. Streiche alle Produktionen der Form $A \rightarrow B$ aus P.

3. Für alle $A, B \in V$, $A \neq B$, mit

$$A \to^+ B,$$

füge für jede Produktion der Form

$$A' \to \alpha A \in P$$

die Produktion

$$A' \to \alpha B$$

zu P hinzu.

/* quadratischer Aufwand */

4. Streiche alle Produktionen der Form $A \to B$ aus P.

Zusammenfassend können wir festhalten:

Satz 61

Aus einer kontextfreien Grammatik $G = (V, \Sigma, P, S)$ der Größe $s(G)$ kann in Zeit $O(|V| \cdot s(G))$ eine äquivalente kontextfreie Grammatik in Chomsky-Normalform der Größe $O(|V| \cdot s(G))$ erzeugt werden.

4.3 Der Cocke-Kasami-Younger-Algorithmus

Der CYK-Algorithmus entscheidet das Wortproblem für kontextfreie Sprachen, falls die Sprache in Form einer Grammatik in Chomsky-Normalform gegeben ist.

Eingabe: Grammatik $G = (V, \Sigma, P, S)$ in Chomsky-Normalform, $w = w_1 \dots w_n \in \Sigma^*$ mit der Länge n. O.B.d.A. $n > 0$.

Definition 62

$$V_{ij} := \{A \in V;\ A \to w_i \dots w_j\}\ .$$

Es ist klar, dass $w \in L(G) \Leftrightarrow S \in V_{1n}$.

4.3 Der Cocke-Kasami-Younger-Algorithmus

Der CYK-Algorithmus entscheidet das Wortproblem für kontextfreie Sprachen, falls die Sprache in Form einer Grammatik in Chomsky-Normalform gegeben ist.

Eingabe: Grammatik $G = (V, \Sigma, P, S)$ in Chomsky-Normalform, $w = w_1 \dots w_n \in \Sigma^*$ mit der Länge n. O.B.d.A. $n > 0$.

Definition 62

$$V_{ij} := \{A \in V;\ A \to w_i \dots w_j\}\ .$$

Es ist klar, dass $w \in L(G) \Leftrightarrow S \in V_{1n}$.

4.3 Der Cocke-Kasami-Younger-Algorithmus

Der CYK-Algorithmus entscheidet das Wortproblem für kontextfreie Sprachen, falls die Sprache in Form einer Grammatik in Chomsky-Normalform gegeben ist.

Eingabe: Grammatik $G = (V, \Sigma, P, S)$ in Chomsky-Normalform, $w = w_1 \dots w_n \in \Sigma^*$ mit der Länge n. O.B.d.A. $n > 0$.

Definition 62

$$V_{ij} := \{A \in V;\ A \to w_i \dots w_j\}\ .$$

Es ist klar, dass $w \in L(G) \Leftrightarrow S \in V_{1n}$.

4.3 Der Cocke-Kasami-Younger-Algorithmus

Der CYK-Algorithmus entscheidet das Wortproblem für kontextfreie Sprachen, falls die Sprache in Form einer Grammatik in Chomsky-Normalform gegeben ist.

Eingabe: Grammatik $G = (V, \Sigma, P, S)$ in Chomsky-Normalform, $w = w_1 \dots w_n \in \Sigma^*$ mit der Länge n. O.B.d.A. $n > 0$.

Definition 62

$$V_{ij} := \{A \in V;\ A \to w_i \dots w_j\}\ .$$

Es ist klar, dass $w \in L(G) \Leftrightarrow S \in V_{1n}$.

Der CYK-Algorithmus berechnet alle V_{ij} rekursiv nach wachsendem $j - i$. Den Anfang machen die

$$V_{ii} := \{A \in V;\ A \to w_i \in P\},$$

der rekursive Aufbau erfolgt nach der Regel

$$V_{ij} = \bigcup_{i \le k < j} \{A \in V;\ (A \to BC) \in P \wedge B \in V_{ik} \wedge C \in V_{k+1,j}\} \quad \text{für } i < j.$$

Die Korrektheit dieses Aufbaus ist klar, wenn die Grammatik in Chomsky-Normalform vorliegt.

Der CYK-Algorithmus berechnet alle V_{ij} rekursiv nach wachsendem $j - i$. Den Anfang machen die

$$V_{ii} := \{A \in V;\ A \to w_i \in P\},$$

der rekursive Aufbau erfolgt nach der Regel

$$V_{ij} = \bigcup_{i \le k < j} \{A \in V;\ (A \to BC) \in P \wedge B \in V_{ik} \wedge C \in V_{k+1,j}\} \quad \text{für } i < j.$$

Die Korrektheit dieses Aufbaus ist klar, wenn die Grammatik in Chomsky-Normalform vorliegt.

Zur Komplexität des CYK-Algorithmus

Es werden $\frac{n^2+n}{2}$ Mengen V_{ij} berechnet. Für jede dieser Mengen werden $|P|$ Produktionen und höchstens n Werte für k betrachtet. Der Test der Bedingung $(A \to BC) \in P \wedge B \in V_{ik} \wedge C \in V_{k+1,j}$ erfordert bei geeigneter Repräsentation der Mengen V_{ij} konstanten Aufwand. Der Gesamtaufwand ist also $O(|P|n^3)$.

Mit der gleichen Methode und dem gleichen Rechenaufwand kann man zu dem getesteten Wort, falls es in der Sprache ist, auch gleich einen Ableitungsbaum konstruieren, indem man sich bei der Konstruktion der V_{ij} nicht nur merkt, welche Nichtterminale sie enthalten, sondern auch gleich, warum sie sie enthalten, d.h. aufgrund welcher Produktionen sie in die Menge aufgenommen wurden.

Zur Komplexität des CYK-Algorithmus

Es werden $\frac{n^2+n}{2}$ Mengen V_{ij} berechnet. Für jede dieser Mengen werden $|P|$ Produktionen und höchstens n Werte für k betrachtet. Der Test der Bedingung $(A \to BC) \in P \wedge B \in V_{ik} \wedge C \in V_{k+1,j}$ erfordert bei geeigneter Repräsentation der Mengen V_{ij} konstanten Aufwand. Der Gesamtaufwand ist also $O(|P|n^3)$.

Mit der gleichen Methode und dem gleichen Rechenaufwand kann man zu dem getesteten Wort, falls es in der Sprache ist, auch gleich einen Ableitungsbaum konstruieren, indem man sich bei der Konstruktion der V_{ij} nicht nur merkt, welche Nichtterminale sie enthalten, sondern auch gleich, warum sie sie enthalten, d.h. aufgrund welcher Produktionen sie in die Menge aufgenommen wurden.

Zur Komplexität des CYK-Algorithmus

Es werden $\frac{n^2+n}{2}$ Mengen V_{ij} berechnet. Für jede dieser Mengen werden $|P|$ Produktionen und höchstens n Werte für k betrachtet. Der Test der Bedingung $(A \to BC) \in P \wedge B \in V_{ik} \wedge C \in V_{k+1,j}$ erfordert bei geeigneter Repräsentation der Mengen V_{ij} konstanten Aufwand. Der Gesamtaufwand ist also $O(|P|n^3)$.

Mit der gleichen Methode und dem gleichen Rechenaufwand kann man zu dem getesteten Wort, falls es in der Sprache ist, auch gleich einen Ableitungsbaum konstruieren, indem man sich bei der Konstruktion der V_{ij} nicht nur merkt, welche Nichtterminale sie enthalten, sondern auch gleich, warum sie sie enthalten, d.h. aufgrund welcher Produktionen sie in die Menge aufgenommen wurden.