Entscheidung des Wortproblems für Typ-2-Sprachen

Beispiel: Die Sprache

$$L = \{a^n b^n c^m \mid n, m \geq 1\}$$

ist kontextfrei:

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow ab \mid aAb \\ B &\rightarrow c \mid cB \end{aligned}$$

Umformen in CNF ergibt:

$$\begin{aligned} S &\rightarrow AB & \qquad D &\rightarrow b \\ A &\rightarrow CD \mid CF & E &\rightarrow c \\ B &\rightarrow c \mid EB & F &\rightarrow AD \\ C &\rightarrow a \end{aligned}$$

Anwendung des Algorithmus für Typ-1-Sprachen?

- Generierung aller Worte bis zu einer Länge n bedeutet exponentiellen Aufwand
- Verwendung eines nichtdeterministischen Kellerautomaten?
  - Nicht besser, da Nichtdeterminismus durch Suche in einem Algorithmus ausgedrückt werden muß.
  - Besser im Worst-Case?
- Idee: Bei der Suche Zwischenergebnisse aufbewahren
- Warum beim Startsymbol anfangen?

Bottom-Up-Vorgehen

Sei $x = aaabbbcc$. Dann erzeugt der Algorithmus die folgende Tabelle:

$x =$	a	a	a	b	b	b	c	c
$j \downarrow$ / $i \rightarrow$								
1	C	C	C	D	D	D	E, B	E, B
2			A				B	
3			F					
4		A						
5		F						
6	A							
7	S							
8	S							

$$
\begin{aligned}
S &\rightarrow AB \\
A &\rightarrow CD \mid CF \\
B &\rightarrow c \mid EB \\
C &\rightarrow a
\end{aligned}
\qquad
\begin{aligned}
D &\rightarrow b \\
E &\rightarrow c \\
F &\rightarrow AD
\end{aligned}
$$

Da S im untersten Kästchen vorkommt, liegt x in der Sprache.

CYK-Algorithmus

Eingabe: $x = a_1 a_2 \ldots a_n$

FOR $i := 1$ TO n DO $(*$ Fall $j = 1 *)$
  $T[i,1] := \{A \in V \mid A \to a_i \in P\}$
END;
FOR $j := 2$ TO n DO $(*$ Fall $j > 1 *)$
  FOR $i := 1$ TO $n + 1 - j$ DO
    $T[i,j] := \emptyset$;
    FOR $k := 1$ TO $j - 1$ DO
      $T[i,j] := T[i,j] \cup \{A \in V \mid A \to BC \in P$
        $\wedge B \in T[i,k] \wedge C \in T[i+k, j-k]\}$
    END;
  END;
END;
IF $S \in T[1,n]$ THEN
  WriteString('x liegt in L(G)')
ELSE
  WriteString('x liegt nicht in L(G)')
END

Der Algorithmus ist benannt nach den Entwicklern: Cocke, Younger, Kasami

Analyse des Algorithmus

Es ist offensichtlich, dass dieser Algorithmus die Komplexität $O(n^3)$ hat, denn er besteht aus 3 ineinander verschachtelten FOR-Schleifen, die jeweils $O(n)$ viele Elemente durchlaufen.