Der CYK-Algorithmus

Ziel: Effiziente Lösung des Wortproblems für kontextfreie Sprachen, die durch Grammatiken in CNF gegeben sind.

CYK: Cocke-Younger-Kasami
Sei eine CNF-Grammatik G und ein Wort $w \in \Sigma^*$ gegeben.

Frage: Gilt $w \in L(G)$?

Zur Erinnerung: Alle Regeln haben die Form $A \to BC$ oder $A \to a$.

Tatsächlich soll der neue Algorithmus *ganz erheblich* viel schneller sein als der zu Anfang des Semesters vorgestellte (exponentielle) Typ-1 Algorithmus!

Die Grundidee des CYK-Algorithmus

Wie sieht die Ableitung eines Wortes $w \in L(G)$ mit $|w| \geq 2$ bei gegebener Typ-2 Grammatik in CNF im allgemeinen aus?

Gesamteffekt: $S \Rightarrow^* w$

Es muss zu Beginn eine Regel $S \to AB$ benutzt werden, und dann:

$$A \Rightarrow^* u \qquad\qquad B \Rightarrow^* v$$

für zwei Wörter u, v mit $w = uv$.

Jedes der Wörter u und v kann nun entweder von Länge 1 sein, oder es ist von Länge ≥ 2, und dann können wir ein ähnliches Argument iterieren!

Bei Länge 1 einfach in P nachsehen, ob Regel $A \to u$ bzw. $B \to v$ vorkommt...

Wie wird es wirklich gemacht?

Tatsächlich geht der CYK-Algorithmus genau anders herum vor:

Input sei das Wort $w = w_1 w_2 \dots w_n$ mit $w_i \in \Sigma$ und $n \geq 2$.

Beginne mit Länge 1, dann 2, etc. bis Länge n und prüfe *alle Teilwörter* darauf, aus welchen Variablen sie erzeugbar sind, d.h. zum Beispiel für Länge 4:

Für alle $i \leq n-3$, und für alle $A \in V$, prüfe ob

$$A \Rightarrow^* w_i w_{i+1} w_{i+2} w_{i+3}.$$

Und wie tut man das?

Man prüft für jede Regel $A \to XY$ mit $X, Y \in V$, ob es eine Zerlegung $w_i w_{i+1} w_{i+2} w_{i+3} = uv$ gibt mit $X \Rightarrow^* u$ und $Y \Rightarrow^* v$.

Der Plan des CYK-Algorithmus

Eingabe sei wie gesagt $w = w_1 w_2 \dots w_n$.

Wir definieren Teilmengen $T_{i,j}$ von V wie folgt:

Die Variable A gehöre zu $T_{i,j}$ genau dann, wenn

$$A \Rightarrow^* w_i w_{i+1} \dots w_{i+j-1}$$

Offenbar kann hier $i \in \{1, \dots, n\}$ beliebig sein, aber j nur aus der Menge $\{1, \dots, n - i + 1\}$.

Angenommen, wir könnten alle Mengen $T_{i,j}$ ermitteln und abspeichern, dann wäre das Wortproblem gelöst, denn es gilt:

$$w \in L(G) \iff S \in T_{1,n}$$

Berechnung der $T_{i,j}$

Es ist offensichtlich, wie man die Mengen $T_{i,1}$ ermittelt:

$$A \in T_{i,1} \iff (A, w_i) \in P$$

Wie schon erwähnt, genügt also eine Überprüfung der Menge P.

Aber wie ermittelt man $T_{i,2}$, $T_{i,3}$, $T_{i,4}$ usw. ?

Bei der Berechnung von $T_{i,j}$ mit $j > 1$ können wir davon ausgehen, dass die $T_{r,s}$ für $s < j$ und *beliebige* Werte von r schon bekannt sind. (Dazu müssen sie in einer Tabelle aufgehoben werden!)

Wir wollen möglichst präzise ausdrücken, was es heißt, zu $T_{i,j}$ zu gehören – diese Beschreibung kann man dann in Programmcode übersetzen!

Berechnung der $T_{i,j}$, Forts.

Eine Variable A gehört zu $T_{i,j}$ *genau dann, wenn:*

Es gibt Variablen B und C und ein $k \in \{1, j-1\}$, so dass

$$A \to BC \in P$$
$$B \in T_{i,k}$$
$$C \in T_{i+k,j-k}$$

Denn die erste Bedingung stellt sicher, dass $A \Rightarrow BC$, die zweite, dass $B \Rightarrow^* w_i \ldots w_{i+k-1}$, und die dritte, dass $C \Rightarrow^* w_{i+k} \ldots w_{i+k+j-k-1}$, also $C \Rightarrow^* w_{i+k} \ldots w_{i+j-1}$.

Zusammen also $A \Rightarrow^* w_i \ldots w_{i+j-1}$, wie gewünscht.