Die ε-Sonderregel

Bevor wir uns dem Wortproblem zuwenden, wollen wir noch die bisher ausgeklammerte ε-Problematik klären:

Da alle Regeln in Typ-1, Typ-2 und Typ-3 Grammatiken nichtverkürzend sind, kann von diesen Grammatiken niemals das leere Wort erzeugt werden.

Das ist nicht erwünscht!

Dieses Problem kann aber leicht gelöst werden:

ε-Sonderregel: Die Produktion $S \to \varepsilon$ kann zugelassen werden. Allerdings darf S dann in keiner *rechten Seite* einer Regel vorkommen.

Durch diese Sonderregelung können keine Probleme entstehen, denn in allen drei Klassen kann man nachprüfen, dass jede Grammatik so angepasst werden kann, dass sie diese Bedingung erfüllt. (Die Details finden Sie im Buch.)

1.1.3 Das Wortproblem

Das Wortproblem für eine gegebene, d.h. feste Grammatik $G = (V, \Sigma, P, S)$ ist so definiert:

EINGABE: $w \in \Sigma^*$

AUSGABE: 1, falls $w \in L(G)$

0, falls $w \notin L(G)$

Dieses Problem erscheint in vielen Zusammenhängen und auf vielen Ebenen innerhalb der Theoretischen Informatik.
Wir werden ihm noch einige Male begegnen...

Entscheidbarkeit des Wortproblems

SATZ: Das Wortproblem für Typ-1 Sprachen ist entscheidbar.

(Dabei sei die Sprache durch eine Typ-1 Grammatik gegeben.)

1. Frage: Was heißt hier *entscheidbar?*

   Es existiert ein Algorithmus, der bei jeder Eingabe hält und JA ausgibt, falls das Eingabewort zur Sprache gehört, bzw. NEIN, falls nicht.

2. Frage: Wie sieht es mit Typ-2 und Typ-3 Sprachen aus?

   Auch für diese ist das Wortproblem entscheidbar – das folgt aus dem Satz!

3. Frage: Und was ist mit Typ-0?

   Es gibt Typ-0 Sprachen, deren Wortproblem unentscheidbar ist – diese Tatsache werden wir später noch zeigen.

Beweis der Entscheidbarkeit

Als Beweis genügt es nach der Definition der Entscheidbarkeit, einen geeigneten Algorithmus anzugeben.

Die vorliegende Grammatik sei $G = (V, \Sigma, P, S)$ und es gelte für alle $(u, v) \in P$ die Ungleichung $|u| \leq |v|$. Außerdem nehmen wir ohne Einschränkung an, dass $\varepsilon \notin L(G)$ gilt.

Der Algorithmus:

INPUT sei x mit $|x| = n$
$T := \{S\}$;
REPEAT $T_1 := T$; $T := \mathrm{Abl}_n(T_1)$
UNTIL $(x \in T)$ OR $(T = T_1)$;
IF $x \in T$ THEN OUTPUT(1) ELSE OUTPUT(0);

Was bedeutet hier $\mathrm{Abl}_n(\ldots)$? Definition auf der nächsten Folie...

Beweis (Fortsetzung)

Wir definieren für alle $X \subseteq (V \cup \Sigma)^*$:

$$\mathrm{Abl}_n(X) = X \cup \{w \in (V \cup \Sigma)^* \mid |w| \leq n \text{ und } \exists y \in X : y \Rightarrow_G w\}$$

Die Korrektheit des Algorithmus' haben wir dann gezeigt, wenn die folgende Äquivalenz klar ist:

$$x \in T \iff x \in L(G)$$

Die Implikation $x \in T \implies x \in L(G)$ ist offensichtlich.

Zur Rückrichtung: Wenn $x \in L(G)$ gilt, gibt es eine Kette

$$S \Rightarrow_G w_1 \Rightarrow_G w_2 \cdots \Rightarrow_G w_k = x$$

Aber dann kann man induktiv zeigen, dass w_i spätestens nach dem i-ten Durchlauf der REPEAT-Schleife zu T gehört. Also am Ende auch x, da $x = w_k$.

Ein Beispiel

Wir betrachten die Grammatik $G = (V, \Sigma, P, S)$ mit:

$$P = \{(S, aSBc), (S, abc), (cB, Bc), (bB, bb)\}$$

Es sei $n = 9$ und wir bezeichnen mit T_i den Zustand der Variablen T nach i Durchgängen. Dann gilt $T_0 = \{S\}$ und

$T_1 = \{S, aSBc, abc\}$

$T_2 = \{S, aSBc, abc, aaSBcBc, aabcBc\}$

$T_3 = T_2 \cup \{aaabcBcBc, aaSBBcc, aabBcc\}$

$T_4 = T_3 \cup \{aaabBccBc, aaabcBBcc, aabbcc\}$

$T_5 = T_4 \cup \{aaabbccBc, aaabBcBcc\}$

$T_6 = T_5 \cup \{aaabbcBcc, aaabBBccc\}$

$T_7 = T_6 \cup \{aaabbBccc\}$

$T_8 = T_7 \cup \{aaabbbccc\}$

$T_9 = T_8$ Hier erfolgt der Abbruch!

Also enthält $L(G)$ bis zur Länge 9 nur die Wörter abc, $a^2b^2c^2$ und $a^3b^3c^3$.

Welche Sprache ist $L(G)$?