Zusammenfassung Typ-2

Wir wollen jetzt noch in aller Kürze zusammenfassen, was wir schon über Typ-2 wissen und was wir in den nächsten Einheiten noch behandeln werden.

Definitionen: Typ-2 Grammatik, Typ-2 Sprache, ε-Sonderregel

Typ-2 Grammatik: $G = (V, \Sigma, P, S)$, wobei für alle $(u, v) \in P$ gilt, dass $u \in V$ und $|v| \geq 1$.

Und was ist eine Typ-2 Sprache? Antwort: $L \subseteq \Sigma^*$ ist Typ-2 Sprache, wenn eine Typ-2 Grammatik G existiert mit $L = L(G)$.

Auch hier (wie bei Typ-1 und Typ-3) kann man per *ε-Sonderregel* erreichen, dass auch Sprachen Typ-2 sein können, die ε enthalten.

Syntaxbäume spielen eine wichtige Rolle bei Typ-2 Sprachen.

Spezielle Formen der Syntaxbäume bei Grammatiken in Normalform (CNF bzw. GNF).

Chomsky-Normalform

Definition, Satz, Beweistechnik, Besonderheiten

Die Typ-2 Grammatik G ist *in CNF*, wenn für alle $(u, v) \in P$ gilt:

$$u \in V \text{ und } v \in V^2 \cup \Sigma.$$

Der Satz über die CNF besagt, dass es zu jeder Typ-2 Grammatik G eine Typ-2 Grammatik G' in CNF gibt, so dass $L(G) = L(G')$.

Formal wichtige Voraussetzung hierbei: $\varepsilon \notin L(G)$.

Der Beweis besteht darin zu zeigen, dass man bei jeder Typ-2 Grammatik die 5 Punkte anwenden kann, um eine CNF zu erhalten:
1) Ringableitungen entfernen – 2) Variablen anordnen – 3) Abkürzungen verwenden – 4) Pseudoterminale einführen – 5) Lange rechte Seiten aufteilen

Besonderheit: Ableitungslänge genau $2n - 1$ für Wort der Länge n.

Greibach-Normalform

Definition, Satz, Beweistechnik, Besonderheiten

Die Typ-2 Grammatik G ist *in GNF*, wenn für alle $(u, v) \in P$ gilt:

$$u \in V \text{ und } v \in \Sigma V^*.$$

Der Satz über die GNF besagt, dass es zu jeder Typ-2 Grammatik G eine Typ-2 Grammatik G' in GNF gibt, so dass $L(G) = L(G')$.

Auch hier müssen wir verlangen: $\varepsilon \notin L(G)$.

Der Beweis besteht darin zu zeigen, dass man bei jeder Typ-2 Grammatik eine GNF erhält, wenn man zwei Algorithmen anwendet:

1) Regeln $A_i \to A_j\beta$ nur mit $i < j$.
2) Durch Einsetzen von A_m bis hinunter zu A_1 richtige Form erzeugen.

Besonderheit: Ableitungslänge genau n für Wort der Länge n.

Pumping-Lemma für Typ-2

Satz, Beweisskizze, Anwendungen, einelementiges Alphabet

Für jede Typ-2 Sprache L existiert n (die *Pumping Konstante),* so dass jedes *lange* Wort $z \in L$ (d.h. $|z| \geq n$), gepumpt werden kann.

D.h., z kann zerlegt werden in $uvwxy$, wobei $vx \neq \varepsilon$, $|vwx| \leq n$ und für alle i: $uv^iwx^iy \in L$ (insbesondere auch für $i = 0$.)

Zum Beweis geht man von einer Grammatik G in CNF aus und betrachtet den Ableitungsbaum für ein langes Wort. Bei genügend großem n müssen sich Variablen auf mindestens einem Pfad im Baum wiederholen! Der Rest ist Technik...

Wichtigste Anwendungen: 1. $\{a^nb^nc^n \mid n \geq 1\}$ ist nicht Typ-2.

2. Bei einelem. Alphabet sind alle Typ-2 Sprachen schon Typ-3.

Was kommt jetzt?

CYK-Algorithmus – wozu? wie? Grundidee?

Das Ziel ist eine *effiziente* Lösung des Wortproblems für Typ-2 Sprachen, also die Beantwortung der Frage, ob $w \in L(G)$ gilt.

Ausgangspunkt ist eine Grammatik in CNF, also beginnt die Ableitung eines Wortes der Länge mindestens 2 immer mit einer Regel der Form (S, AB). Das Wort $w \in L$ kann also zerlegt werden in $w = uv$, wobei $A \Rightarrow^* u$ und $B \Rightarrow^* v$. Da diese beiden Fragen von der gleichen Form sind wie die Frage, ob $S \Rightarrow^* w$ gilt, aber mit kürzeren Wörtern, kommt man mit geeigneter Iteration zur Lösung!

Der *CYK-Algorithmus,* benannt nach seinen Erfindern Cocke, Younger und Kasami, kombiniert diese Beweisidee mit der Technik der *dynamischen Programmierung* – Details in Einheit 26.

Typ-2 und Maschinen

Grammatiktyp vs. Maschinentyp, Kellerautomat

Bei den Typ-3 Sprachen haben wir viele Erkenntnisse aus der Darstellung mit endlichen Automaten gewinnen können. Auch für Typ-2 kann man eine ähnliche Charakterisierung entwickeln.

Allerdings muss das Modell zusätzliche Fähigkeiten haben, um Sprachen wie $\{a^n b^n \mid n \geq 1\}$ erkennen zu können.

Wir brauchen also eine Art der Speicherung (für beliebig große Zahlen) – der Speicher darf uns aber nicht die Erkennung der Sprache $\{a^n b^n c^n \mid n \geq 1\}$ erlauben. (Warum nicht?)

Die Lösung ist der sogenannte *Kellerautomat*, ein Automat wie ein DEA, aber zusätzlich mit einem *Keller* oder *Stack* als Speicher.

Die Details kommen in den nachfolgenden Einheiten.