5.10 Wo endliche Automaten scheitern

LK

LK

Die Chomsky-Hierarchie und die Beispiele nicht-regulärer und nicht-kontextfreier Sprachen stehen nur im LK-Lehrplan. Im Grundkurs genügt die Einsicht aus dem ersten Absatz: Der Zustand eines endlichen Automaten ist sein ganzes Gedächtnis, und das ist endlich.

Ein Türsteher soll am Ende des Abends sagen, ob alle Gäste wieder draußen sind. Mit einer Strichliste ist das leicht. Darf er sich aber nur merken, ob „wenig“, „mittel“ oder „viel“ los ist, kann er es nicht — egal, wie aufmerksam er ist. Genau in dieser Lage ist ein endlicher Automat: Er hat nur endlich viele Zustände, die Gästeschlange ist aber beliebig lang.

Konstruieren Sie einen Automaten, der genau die Wörter $a^n b^n$ akzeptiert, also ab, $aabb$, $aaabbb$ und so weiter — gleich viele a wie b.

Es gelingt nicht, und der Grund ist grundsätzlich: Der Automat müsste sich merken, wie viele a er gelesen hat, bevor das erste b kommt. Diese Zahl ist unbeschränkt, seine Zustandsmenge aber endlich. Bei genügend langer Eingabe muss er zwangsläufig zweimal in denselben Zustand geraten, obwohl er unterschiedlich viele a gesehen hat — und dann kann er die beiden Fälle nicht mehr unterscheiden.

Dasselbe gilt für die Klammersprache aus 5.4: Verschachtelte Klammern erfordern ein unbeschränktes Gedächtnis. Genau deshalb war jene Grammatik nicht regulär.

Eine Maschine, die $a^n b^n$ sehr wohl erkennt, kennen Sie schon: die **Turingmaschine** aus Abschnitt 1.2. Sie streicht Paare durch und schreibt sich die Zwischenstände aufs Band. Der Unterschied ist genau dieser Schreibplatz — der endliche Automat hat keinen.

Merke

Endliche Automaten erkennen genau die regulären Sprachen. Alles, was Zählen oder Schachteln erfordert, liegt außerhalb.

Für solche Sprachen braucht man mächtigere Maschinen. Ein **Kellerautomat** besitzt zusätzlich einen Stapelspeicher: Bei jedem a legt er eine Marke ab, bei jedem b nimmt er eine weg — und akzeptiert, wenn der Stapel am Ende leer ist. Damit erkennt er die kontextfreien Sprachen, zu denen auch die Klammerstrukturen der Programmiersprachen gehören. Ganz oben steht die Turingmaschine aus Abschnitt 1.2. Diese Staffelung heißt **Chomsky-Hierarchie**:

Typ	Sprachklasse	Maschine	Form der Produktionen	Beispiel
3	regulär	endlicher Automat	$A \to xB$ oder $A \to x$	Bezeichner, Zahlen
2	kontextfrei	Kellerautomat	$A \to \alpha$ (links ein Nichtterminal)	$a^n b^n$, Klammern
1	kontextsensitiv	linear beschränkter Automat	$\alpha \to \beta$ mit $\lvert\alpha\rvert \le \lvert\beta\rvert$	$a^n b^n c^n$
0	rekursiv aufzählbar	Turingmaschine	beliebig	haltende Programme

In Worten, von der schwächsten zur mächtigsten Stufe:

- **Typ 3, regulär.** Das Gedächtnis ist endlich. Die Maschine merkt sich eine von endlich vielen Lagen, sonst nichts. Sie kann erkennen, *dass* etwas vorkam, aber nicht, *wie oft*.
- **Typ 2, kontextfrei.** Dazu kommt ein Stapel. Damit lässt sich zählen und schachteln, aber immer nur **eine** Sache zur Zeit — was heruntergenommen ist, ist weg.
- **Typ 1, kontextsensitiv.** Der Speicher wächst mit der Eingabe. Jede Regel darf die Satzform nur verlängern, nie verkürzen; deshalb bleibt die Wortfrage entscheidbar.
- **Typ 0, rekursiv aufzählbar.** Keine Beschränkung mehr. Das ist genau das, was eine Turingmaschine aufzählen kann — und hier wird die Wortfrage unentscheidbar: Gehört das Wort zur Sprache, hält die Maschine irgendwann und sagt ja; gehört es nicht dazu, läuft sie möglicherweise für immer. Das ist genau die *partielle* Entscheidbarkeit aus Abschnitt 4.4. „Rekursiv aufzählbar" ist deshalb **nicht** dasselbe wie „berechenbar".

Die vier Klassen sind echt ineinander geschachtelt — jede Stufe kann alles, was die vorige kann, und mehr:

$$\text{Typ 3} \subsetneq \text{Typ 2} \subsetneq \text{Typ 1} \subsetneq \text{Typ 0}$$

Zwei Beispiele markieren die beiden Sprünge, die in der Schule am häufigsten gebraucht werden:

Merke

$L_1 = \{a^n b^n \mid n \geq 1\}$ ist **nicht regulär**, aber kontextfrei. Der Grund steht oben: Ein endlicher Automat kann die a nicht zählen, ein Kellerautomat schon.

$L_2 = \{a^n b^n c^n \mid n \geq 1\}$ ist **nicht kontextfrei**, aber kontextsensitiv. Hier reicht auch der Stapel nicht: Beim Abarbeiten der b wird er geleert — für die c ist die Zahl der a dann verloren. Man müsste **zweimal** dasselbe zählen, und dafür braucht es ein Band statt eines Stapels.

Das erklärt nebenbei den Aufbau eines Compilers: Der Lexer erkennt Token — die sind regulär, ein endlicher Automat genügt. Der Parser prüft die Struktur — die ist kontextfrei und braucht einen Stapel.

Arbeitsauftrag 10

41. Versuchen Sie, einen Automaten für $a^n b^n$ mit höchstens fünf Zuständen zu bauen. Beschreiben Sie, woran es scheitert, und geben Sie ein Wort an, das Ihr Automat fälschlich akzeptiert.
42. Begründen Sie, warum ein Automat mit 100 Zuständen die Sprache $a^n b^n$ für $n \leq 40$ korrekt entscheiden kann — und warum das nichts nützt.
43. Ordnen Sie ein: Telefonnummern, korrekt geklammerte Terme, Datumsangaben im Format TT.MM.JJJJ, vollständige Pascal-Programme.