15 REGULÄRE AUSDRÜCKE UND RECHTSLINEARE GRAMMATIKEN

Am Ende von Einheit 14 über endliche Automaten haben wir gesehen, dass manche formale Sprachen zwar von kontextfreien Grammatiken erzeugt, aber nicht von endlichen Akzeptoren erkannt werden können. Damit stellt sich für Sie vielleicht zum ersten Mal die Frage nach einer *Charakterisierung*, nämlich der der mit endlichen Akzeptoren erkennbaren Sprachen. Damit ist eine präzise Beschreibung dieser formalen Sprachen gemeint, die nicht (oder jedenfalls nicht offensichtlich) Automaten benutzt.

In den beiden Abschnitten dieser Einheit werden Sie zwei solche Charakterisierungen kennenlernen, die über reguläre Ausdrücke und die über rechtslineare Grammatiken. In Abschnitt 15.3 wird es unter anderem um eine bequeme Verallgemeinerung vollständiger Induktion gehen.

15.1 REGULÄRE AUSDRÜCKE

Der Begriff *regulärer Ausdruck* geht ursprünglich auf Stephen Kleene (1956) zurück und wird heute in unterschiedlichen Bedeutungen genutzt. In dieser Einheit führen wir kurz regulären Ausdrücken nach der „klassischen“ Definition ein.

Etwas anderes (nämlich allgemeineres) sind die Varianten der *Regular Expressions*, von denen Sie möglicherweise schon im Zusammenhang mit dem ein oder anderen Programm (emacs, grep, sed, ...) oder der ein oder anderen Programmiersprache (Java, Python, Perl) gelesen haben. Für Java gibt es das Paket java.util.regex. Regular expressions sind eine deutliche Verallgemeinerung regulärer Ausdrücke, auf die wir in dieser Vorlesung nicht eingehen werden. Alles was wir im folgenden über reguläre Ausdrücke sagen, ist aber auch bei regular expressions anwendbar.

Wir kommen direkt zur Definition regulärer Ausdrücke. Sie wird sie hoffentlich an das ein oder andere aus der Einheit 6 über formale Sprachen und die zugehörigen Übungsaufgaben erinnern.

Es sei A ein Alphabet, das keines der fünf Zeichen aus $Z = \{\mathtt{|}, \mathtt{(}, \mathtt{)}, \mathtt{*}, \emptyset\}$ enthält. Ein *regulärer Ausdruck* über A ist eine Zeichenfolge über dem Alphabet $A \cup Z$, die gewissen Vorschriften genügt. Die Menge der regulären Ausdrücke ist wie folgt festgelegt: *regulärer Ausdruck*

- $\emptyset$ ist ein regulärer Ausdruck.
- Für jedes $x \in A$ ist x ein regulärer Ausdruck.
- Wenn R_1 und R_2 reguläre Ausdrücke sind, dann sind auch $\mathtt{(}R_1\mathtt{|}R_2\mathtt{)}$ und $\mathtt{(}R_1R_2\mathtt{)}$ reguläre Ausdrücke.

- Wenn R ein regulärer Ausdruck ist, dann auch `(`R`*)`.
- Nichts anderes sind reguläre Ausdrücke.

Um sich das Schreiben zu vereinfachen, darf man Klammern auch weglassen. Im Zweifelsfall gilt „Stern- vor Punkt- und Punkt- vor Strichrechnung“, d. h. R_1`|`R_2R_3`*` ist z. B. als `(`R_1`|(`R_2`(`R_3`*)))` zu verstehen. Bei mehreren gleichen binären Operatoren gilt das als links geklammert; zum Beispiel ist R_1`|`R_2`|`R_3 als `((`R_1`|`R_2`)|`R_3`)` zu verstehen.

Man kann die korrekte Syntax regulärer Ausdrücke auch mit Hilfe einer kontextfreien Grammatik beschreiben: Zu gegebenem Alphabet A sind die legalen regulären Ausdrücke gerade die Wörter, die von der Grammatik

$$\begin{aligned} G &= (\{R\},\ \{\texttt{|},\texttt{(},\texttt{)},\texttt{*},\texttt{∅}\} \cup A,\ R,\ P) \\ \text{und } P &= \{R \to \texttt{∅},\ \ R \to \texttt{(}R\texttt{|}R\texttt{)},\ \ R \to \texttt{(}RR\texttt{)},\ \ R \to \texttt{(}R\texttt{*)}\} \\ &\quad \cup \{R \to x \mid x \in A\} \end{aligned}$$

erzeugt werden.

Die folgenden Zeichenketten sind alle reguläre Ausdrücke über dem Alphabet {`a`,`b`}:

- `∅`
- `a`
- `b`
- `(ab)`
- `((ab)a)`
- `(((ab)a)a)`
- `((ab)(aa))`
- `(∅|b)`
- `(a|b)`
- `((a(a|b))|b)`
- `(a|(b|(a|a)))`
- `(∅*)`
- `(a*)`
- `((ba)(b*))`
- `(((ba)b)*)`
- `((a*)*)`
- `(((((ab)b)*)*)|(∅*))`

Wendet man die Klammereinsparungsregeln an, so ergibt sich aus den Beispielen mit Klammern:

- `ab`
- `aba`
- `abaa`
- `ab(aa)`
- `∅|b`
- `a|b`
- `a(a|b)|b`
- `(a|(b|(a|a)))`
- `∅*`
- `a*`
- `bab*`
- `(bab)*`
- `a**`
- `(abb)**|∅*`

Die folgenden Zeichenketten sind dagegen auch bei Berücksichtigung der Klammereinsparungsregeln *keine* regulären Ausdrücke über {`a`,`b`}:

`(|b)`	vor `|` fehlt ein regulärer Ausdruck
`|∅|`	vor und hinter `|` fehlt je ein regulärer Ausdruck
`()ab`	zwischen `(` und `)` fehlt ein regulärer Ausdruck
`((ab)`	Klammern müssen „gepaart“ auftreten
`*(ab)`	vor `*` fehlt ein regulärer Ausdruck
`c*`	`c` ist nicht Zeichen des Alphabetes

Reguläre Ausdrücke werden benutzt, um formale Sprachen zu spezifizieren. Auch dafür bedient man sich wieder einer induktiven Vorgehensweise; man spricht auch von einer induktiven Definition:

durch R beschriebene Sprache $\langle R \rangle$

Die *von einem regulären Ausdruck R beschriebene formale Sprache* $\langle R \rangle$ ist wie folgt definiert:

- $\langle \texttt{∅} \rangle = \{\}$ (d. h. die leere Menge).
- Für $x \in A$ ist $\langle x \rangle = \{x\}$.
- Sind R_1 und R_2 reguläre Ausdrücke, so ist $\langle R_1 \texttt{|} R_2 \rangle = \langle R_1 \rangle \cup \langle R_2 \rangle$.
- Sind R_1 und R_2 reguläre Ausdrücke, so ist $\langle R_1 R_2 \rangle = \langle R_1 \rangle \cdot \langle R_2 \rangle$.
- Ist R ein regulärer Ausdruck, so ist $\langle R\texttt{*} \rangle = \langle R \rangle^*$.

Betrachten wir drei einfache Beispiele:

- $R = \texttt{a|b}$: Dann ist $\langle R \rangle = \langle \texttt{a|b} \rangle = \langle \texttt{a} \rangle \cup \langle \texttt{b} \rangle = \{\texttt{a}\} \cup \{\texttt{b}\} = \{\texttt{a}, \texttt{b}\}$.
- $R = \texttt{(a|b)*}$: Dann ist $\langle R \rangle = \langle \texttt{(a|b)*} \rangle = \langle \texttt{a|b} \rangle^* = \{\texttt{a}, \texttt{b}\}^*$.
- $R = \texttt{(a*b*)*}$: Dann ist $\langle R \rangle = \langle \texttt{(a*b*)*} \rangle = \langle \texttt{a*b*} \rangle^* = (\langle \texttt{a*} \rangle \langle \texttt{b*} \rangle)^* = (\langle \texttt{a} \rangle^* \langle \texttt{b} \rangle^*)^* = (\{\texttt{a}\}^* \{\texttt{b}\}^*)^*$.

Mehr oder weniger kurzes Überlegen zeigt übrigens, dass für die Sprachen des zweiten und dritten Beispiels gilt: $(\{\texttt{a}\}^* \{\texttt{b}\}^*)^* = \{\texttt{a}, \texttt{b}\}^*$. Man kann also die gleiche formale Sprache durch verschiedene reguläre Ausdrücke beschreiben — wenn sie denn überhaupt so beschreibbar ist.

Damit klingen (mindestens) die beiden folgenden Fragen an:

1. Kann man allgemein algorithmisch von zwei beliebigen regulären Ausdrücken R_1, R_2 feststellen, ob sie die gleiche formale Sprache beschreiben, d. h. ob $\langle R_1 \rangle = \langle R_2 \rangle$ ist?
2. Welche formalen Sprachen sind denn durch reguläre Ausdrücke beschreibbar?

Die Antwort auf die erste Frage ist *ja*. Allerdings hat das Problem, die Äquivalenz zweier regulärer Ausdrücke zu überprüfen, die Eigenschaft PSPACE-vollständig zu sein wie man in der Komplexitätstheorie sagt. Was das ist, werden wir in der Einheit über Turingmaschinen kurz anreißen. Es bedeutet unter anderem, dass alle *bisher bekannten* Algorithmen im allgemeinen *sehr sehr langsam* sind: die Rechenzeit wächst „stark exponentiell“ mit der Länge der regulären Ausdrücke (z. B. wie 2^{n^2} o.ä.). Es sei noch einmal betont, dass dies für alle bisher bekannten Algorithmen gilt. Man weiß nicht, ob es vielleicht doch signifikant schnellere Algorithmen für das Problem gibt, aber man sie „nur noch nicht gefunden“ hat.

Nun zur Antwort auf die zweite Frage. (Was rechtslineare Grammatiken sind, werden wir in nachfolgenden Abschnitt 15.2 gleich noch beschreiben. Es handelt sich um einen Spezialfall kontextfreier Grammatiken.)

15.1 Satz. *Für jede formale Sprache L sind die folgenden drei Aussagen äquivalent:*

1. *L kann von einem endlichen Akzeptor erkannt werden.*
2. *L kann durch einen regulären Ausdruck beschrieben werden.*
3. *L kann von einer rechtslinearen Grammatik erzeugt werden.*

reguläre Sprache

Eine formale Sprache, die die Eigenschaften aus Satz 15.1 hat, heißt *reguläre Sprache*. Da jede rechtslineare Grammatik eine kontextfreie Grammatik ist, ist jede reguläre Sprache eine kontextfreie Sprache.

Zwar werden wir Satz 15.1 nicht im Detail beweisen, aber wir wollen zumindest einige Dinge andeuten, insbesondere auch eine grundlegende Vorgehensweise.

Satz 15.1 hat folgende prinzipielle Struktur:

- Es werden drei Aussagen $\mathcal{A}$, $\mathcal{B}$ und $\mathcal{C}$ formuliert.
- Es wird behauptet:
  - $\mathcal{A} \Longleftrightarrow \mathcal{B}$
  - $\mathcal{B} \Longleftrightarrow \mathcal{C}$
  - $\mathcal{C} \Longleftrightarrow \mathcal{A}$

Man kann nun natürlich einfach alle sechs Implikationen einzeln beweisen. Aber das muss man gar nicht! Dann wenn man zum Beispiel schon gezeigt hat, dass $\mathcal{A} \Longrightarrow \mathcal{B}$ gilt und dass $\mathcal{B} \Longrightarrow \mathcal{C}$, dann folgt $\mathcal{A} \Longrightarrow \mathcal{C}$ automatisch. Das sieht man anhand der folgenden Tabelle:

	$\mathcal{A}$	$\mathcal{B}$	$\mathcal{C}$	$\mathcal{A} \Longrightarrow \mathcal{B}$	$\mathcal{B} \Longrightarrow \mathcal{C}$	$\mathcal{A} \Longrightarrow \mathcal{C}$
1				W	W	W
2			W	W	W	W
3		W		W		W
4		W	W	W	W	W
5	W				W	
6	W		W		W	W
7	W	W		W		
8	W	W	W	W	W	W

In allen Zeilen 1, 2, 4 und 8, in denen sowohl für $\mathcal{A} \Longrightarrow \mathcal{B}$ als auch für $\mathcal{B} \Longrightarrow \mathcal{C}$ ein W (für *wahr*) eingetragen ist, ist das auch für $\mathcal{A} \Longrightarrow \mathcal{C}$ der Fall. Statt *falsch* haben wir der besseren Übersicht wegen die entsprechenden Felder freigelassen.

Wenn man $\mathcal{A} \Longrightarrow \mathcal{B}$ und $\mathcal{B} \Longrightarrow \mathcal{C}$ schon bewiesen hat, dann muss man also $\mathcal{A} \Longrightarrow \mathcal{C}$ gar nicht mehr beweisen. Und beweist man nun zusätzlich noch $\mathcal{C} \Longrightarrow \mathcal{A}$, dann

- folgt mit $\mathcal{A} \Longrightarrow \mathcal{B}$ sofort $\mathcal{C} \Longrightarrow \mathcal{B}$ und
- mit $\mathcal{B} \Longrightarrow \mathcal{C}$ folgt sofort $\mathcal{B} \Longrightarrow \mathcal{A}$,

und man ist fertig.

Statt sechs Implikationen zu beweisen zu müssen, reichen also drei. Für einen Beweis von Satz 15.1 genügen daher folgende Konstruktionen:

- zu gegebenem endlichen Akzeptor A ein regulärer Ausdruck R mit $\langle R \rangle = L(A)$:

  Diese Konstruktion ist „mittel schwer“. Man kann z. B. einen Algorithmus benutzen, dessen Struktur und Idee denen des Algorithmus von Warshall ähneln.
- zu gegebenem regulären Ausdruck R eine rechtslineare Grammatik G mit $L(G) = \langle R \rangle$:

  Diese Konstruktion ist „relativ leicht“. Wir werden im nächsten Abschnitt noch etwas genauer darauf eingehen.
- zu gegebener rechtslinearer Grammatik G ein endlicher Akzeptor A mit $L(A) = L(G)$:

  Diese Konstruktion ist die schwierigste.

Wie wertvoll Charakterisierungen wie Satz 15.1 sein können, sieht man an folgendem Beispiel: Es sei L eine reguläre Sprache, z. B. die Sprache aller Wörter, in denen irgendwo das Teilwort `abbab` vorkommt. Aufgabe: Man zeige, dass auch das Komplement $L' = \{\mathtt{a}, \mathtt{b}\}^* \setminus L$, also die Menge aller Wörter, in denen nirgends das Teilwort `abbab` vorkommt, regulär ist.

Wüssten wir nur, dass reguläre Sprachen die durch reguläre Ausdrücke beschreibbaren sind, und hätten wir nur einen solchen für L, dann stünden wir vor einem Problem. Damit Sie das auch merken, sollten Sie einmal versuchen, einen regulären Ausdruck für L' hinzuschreiben.

Aber wir wissen, dass wir uns auch endlicher Akzeptoren bedienen dürfen. Und dann ist alles *ganz* einfach: Denn wenn A ein endlicher Akzeptor ist, der L erkennt, dann bekommt man daraus den für L', indem man einfach akzeptierende und ablehnende Zustände vertauscht.

15.2 RECHTSLINEARE GRAMMATIKEN

Mit beliebigen kontextfreien Grammatiken kann man jedenfalls zum Teil andere formale Sprachen erzeugen, als man mit endlichen Akzepteren erkennen kann. Denn die Grammatik $G = (\{X\}, \{\mathtt{a}, \mathtt{b}\}, X, \{X \to \mathtt{a}X\mathtt{b} \mid \varepsilon\})$ erzeugt $\{\mathtt{a}^k\mathtt{b}^k \mid k \in \mathbb{N}_0\}$ und diese Sprache ist nicht regulär.

Aber die folgende einfache Einschränkung tut „das Gewünschte“. Eine *rechtslineare Grammatik* ist eine kontextfreie Grammatik $G = (N, T, S, P)$, die der folgen- *rechtslineare Grammatik*

den Einschränkung genügt: Jede Produktion ist entweder von der Form $X \to w$ oder von der Form $X \to wY$ mit $w \in T^*$ und $X, Y \in N$. Auf der rechten Seite einer Produktion darf also höchstens ein Nichterminalsymbol vorkommen, und wenn dann nur als letztes Symbol.

Die oben erwähnte Grammatik $G = (\{X\}, \{\mathtt{a}, \mathtt{b}\}, X, \{X \to \mathtt{a}X\mathtt{b} \mid \varepsilon\})$ ist also *nicht* rechtslinear, denn in der Produktion $X \to \mathtt{a}X\mathtt{b}$ steht das Nichtterminalsymbol X nicht am rechten Ende.

Und da wir uns überlegt hatten, dass die erzeugte formale Sprache nicht regulär ist, kann es auch gar keine rechtslineare Grammatik geben, die $\{\mathtt{a}^k\mathtt{b}^k \mid k \in \mathbb{N}_0\}$ erzeugt.

Typ-3-Grammatiken
Typ-2-Grammatiken

Es sei auch noch die folgende Sprechweise eingeführt: Rechtslineare Grammatiken heißen auch *Typ-3-Grammatiken* und die schon eingeführten kontextfreien Grammatiken nennt man auch *Typ-2-Grammatiken*. Hier ahnt man schon, dass es noch weiter geht. Es gibt auch noch *Typ-1-Grammatiken* und *Typ-0-Grammatiken*.

Wenn für ein $i \in \{0, 1, 2, 3\}$ eine formale Sprache L von einer Typ-i-Grammatik erzeugt wird, dann sagt man auch, L sei eine *Typ-i-Sprache* oder kurz *vom Typ i*.

Zumindest einer der Vorteile rechtslinearer Grammatiken gegenüber deterministischen endlichen Akzeptoren, wie wir sie im vorangegangenen Kapitel eingeführt haben, ist, dass sie manchmal deutlich kürzer und übersichtlicher hinzuschreiben sind. Ein genaueres Verständnis dafür, warum das so ist, werden Sie bekommen, wenn Sie im dritten Semester auch etwas über sogenannte nichtdeterministische endliche Akzeptoren gelernt haben.

15.3 Kantorowitsch-Bäume und strukturelle Induktion

Kantorowitsch-Baum

Reguläre Ausdrücke kann man auch als sogenannte *Kantorowitsch-Bäume* darstellen. Für den regulären Ausdruck `((b|∅)a)(b*)` ergibt sich zum Beispiel der Graph aus Abbildung 15.1

Hier handelt es sich *nicht* um den Ableitungsbaum gemäß der Grammatik aus Abschnitt 15.1. Aber die Struktur des regulären Ausdruckes wird offensichtlich ebenso gut wiedergegeben und die Darstellung ist sogar noch kompakter.

Solche Bäume wollen wir im folgenden der Einfachheit halber Regex-Bäume nennen. Es sei A irgendein Alphabet. Dann ist ein Baum ein Regex-Baum, wenn gilt:

- Entweder ist es Baum dessen Wurzel zugleich Blatt ist und das ist mit einem $x \in A$ oder $\emptyset$ beschriftet,
- oder es ist ein Baum, dessen Wurzel mit $*$ beschriftet ist und die genau einen

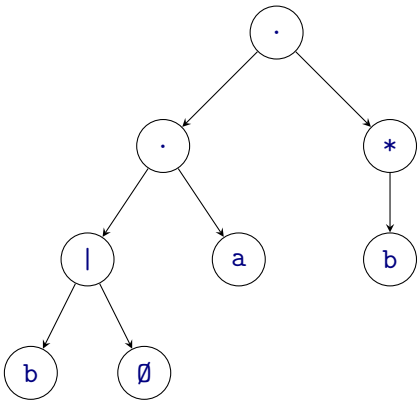

Abbildung 15.1: Der Kantorowitsch-Baum für den regulären Ausdruck `((b|∅)a)(b*)`

Nachfolgeknoten hat, der Wurzel eines Regex-Baumes ist

- oder es ist ein Baum, dessen Wurzel mit · oder mit | beschriftet ist und die genau zwei Nachfolgeknoten hat, die Wurzeln zweier Regex-Bäume sind.

Man beachte, dass linker und rechter Unter-Regex-Baum unterhalb der Wurzel unterschiedliche Höhe haben können.

Wichtig ist für uns im folgenden, dass erstens größere Bäume „aus kleineren zusammengesetzt" werden, und dass zweitens diese Zusammensetzung immer eindeutig ist. Außerdem kommt man bijektiv von regulären Ausdrücken zu Regex-Bäumen und umgekehrt.

Für das weitere definieren wir noch ein oft auftretendes Maß für Bäume, die sogenannte *Höhe* $h(T)$ eines Baumes T. Das geht so: *Höhe eines Baumes*

$$h(T) = \begin{cases} 0 & \text{falls die Wurzel Blatt ist} \\ 1 + \max_i h(U_i) & \text{falls die } U_i \text{ alle Unterbäume von } T \text{ sind} \end{cases}$$

Wenn man beweisen möchte, dass eine Aussage für alle regulären Ausdrücke gilt, dann kann man das dadurch tun, dass man die entsprechende Aussage für alle Regex-Bäume beweist. Und den Beweis für alle Regex-Bäume kann man durch vollständige Induktion über ihre Höhe führen. Bei naiver Herangehensweise tritt aber ein Problem auf: Beim Schritt zu Bäumen der Höhe $n+1$ darf man nur auf Bäume der Höhe n zurückgreifen. Auftretende Unterbäume können aber alle Höhen $i \leq n$ haben, und man möchte gerne für alle die Induktionsvoraussetzung benutzen. Das darf man auch! Wir wollen uns zunächst klar machen, warum das so ist

Dazu sehen wir uns eine einfache Verallgemeinerung vollständiger Induktion an. Es sei $\mathcal{B}(n)$ eine Aussage, die von einer Zahl $n \in \mathbb{N}_0$ abhängt. Wir wollen beweisen: $\forall n \in \mathbb{N}_0 : \mathcal{B}(n)$. Dazu definieren wir eine Aussage $\mathcal{A}(n)$ wie folgt: $\forall i \leq n : \mathcal{B}(i)$. Wenn wir beweisen können, dass $\forall n \in \mathbb{N}_0 : \mathcal{A}(n)$ gilt, dann sind wir fertig, denn aus $\mathcal{A}(n)$ folgt stets $\mathcal{B}(n)$.

Wie sieht ein Induktionsbeweis für $\forall n \in \mathbb{N}_0 : \mathcal{A}(n)$ aus?

Induktionsanfang: Es ist zu zeigen, dass $\mathcal{A}(0)$ gilt, also die Aussage $\forall i \leq 0 : \mathcal{B}(i)$. Das ist offensichtlich äquivalent zu $\mathcal{B}(0)$. Das ist zu zeigen.

Induktionsvoraussetzung: für beliebiges aber festes $n \in \mathbb{N}_0$ gilt: $\mathcal{A}(n)$, also die Aussage $\forall i \leq n : \mathcal{B}(i)$.

Induktionsschluss: Es ist zu zeigen, dass auch $\mathcal{A}(n+1)$ gilt, also die Aussage $\forall i \leq n+1 : \mathcal{B}(i)$. Diese Aussage ist äquivalent zu $\Big(\forall i \leq n : \mathcal{B}(i)\Big) \wedge \mathcal{B}(n+1)$. Wie zeigt man das?

- Der erste Teil $\forall i \leq n : \mathcal{B}(i)$ ist trivial, denn das ist ja gerade die Induktionsvoraussetzung.
- Daher bleibt nur zu zeigen: $\mathcal{B}(n+1)$. Zum Beweis dafür kann man aber ebenfalls auf die Induktionsvoraussetzung zurückgreifen, also auf *alle* Aussagen $\mathcal{B}(0), \mathcal{B}(1), \ldots, \mathcal{B}(n)$ und nicht nur die letzte von ihnen.

Diese Vorgehensweise wollen wir nun anwenden, um zu einen Beweis dafür zu skizzieren, dass es für jeden regulären Ausdruck R eine rechtslineare Grammatik G gibt mit $\langle R \rangle = L(G)$. Bei regulären Ausdrücken denkt man nun vorteilhafterweise an Regex-Bäume und wir machen nun zunächst eine „normale“ vollständige Induktion über die Höhe der Regex-Bäume. Im Schema von eben ist also $\mathcal{B}(n)$ die Aussage:

> Für jeden Regex-Baum R der Höhe n gibt es eine rechtslineare Grammatik G gibt mit $\langle R \rangle = L(G)$.

Wir wollen zeigen, dass $\forall n \in \mathbb{N}_0 : \mathcal{B}(n)$ gilt.

Induktionsanfang: Es ist $\mathcal{B}(0)$ zu zeigen. Man muss also rechtslineare Grammatiken angeben, die die formalen Sprachen $\{x\} = \langle x \rangle$ für $x \in A$ und die leere Menge $\{\} = \langle \emptyset \rangle$ erzeugen. Das ist eine leichte Übung.

Induktionsvoraussetzung: für beliebiges aber festes $n \in \mathbb{N}_0$ gelte die Aussage $\forall i \leq n : \mathcal{B}(i)$, d. h. für jeden Regex-Baum R' mit einer Höhe $i \leq n$ gibt es eine rechtslineare Grammatik G gibt mit $\langle R' \rangle = L(G)$.

Induktionsschluss: Es bleibt zu zeigen, dass auch $\mathcal{B}(n+1)$ gilt, dass also für jeden Regex-Baum R der Höhe $n+1$ eine rechtslineare Grammatik G mit $\langle R \rangle = L(G)$ existiert.

Sei daher R ein beliebiger Regex-Baum der Höhe $n+1$. Dann gibt es drei mögliche Fälle:

1. Die Wurzel von R ist ein $*$-Knoten und hat genau einen Unterbaum R' der Höhe n.
2. Die Wurzel von R ist ein $|$-Knoten und hat genau zwei Unterbäume R_1 und R_2. Da R Höhe $n+1$ hat, hat einer der beiden Unterbäume Höhe n, der andere hat eine Höhe $i \leq n$.
3. Die Wurzel von R ist ein „Konkatenations-Knoten“ und hat genau zwei Unterbäume R_1 und R_2. Da R Höhe $n+1$ hat, hat einer der beiden Unterbäume Höhe n, der andere hat eine Höhe $i \leq n$.

Der entscheidende Punkt ist nun: In den Fällen 2 und 3 darf man nach Induktionsvoraussetzung annehmen, dass für *beide* Unterbäume rechtslineare Grammatiken der gewünschten Art existieren.

In allen drei Fällen kann man dann aus den Grammatiken für den Unterbaum bzw. die Unterbäume die Grammatik für den Regex-Baum, also regulären Ausdruck, R konstruieren. Das wollen wir hier nicht allen Details durchführen und beschränken uns auf den einfachsten Fall, nämlich Fall 2: Seien also $G_1 = (N_1, A, S_1, P_1)$ und $G_2 = (N_2, A, S_2, P_2)$ Typ-3-Grammatiken, die $L(G_1) = \langle R_1 \rangle$ bzw. $L(G_2) = \langle R_2 \rangle$ erzeugen. Ohne Beschränkung der Allgemeinheit nehmen wir an, dass $N_1 \cap N_2 = \emptyset$ ist. Wir wählen ein „neues“ Nichtterminalsymbol $S \notin N_1 \cup N_2$ Damit können wir eine Typ-3-Grammatik G mit $L(G) = \langle R_1 | R_2 \rangle$ ganz leicht hinschreiben:

$$G = (\{S\} \cup N_1 \cup N_2, A, S, \{S \to S_1 \mid S_2\} \cup P_1 \cup P_2)$$

Als erstes muss man sich klar machen, dass auch die Grammatik rechtslinear ist. Tun Sie das; es ist nicht schwer. Etwas Arbeit würde es machen, zu beweisen, dass $L(G) = L(G_1) \cup L(G_2)$ ist. Das wollen wir uns an dieser Stellen sparen.

Sie können das aber ruhig selbst einmal versuchen. Und für die anderen beiden Fälle können Sie das auch einmal versuchen, geeignete (rechtslineare!) Grammatiken zu konstruieren, oder einfach glauben, dass es geht.

strukturelle Induktion

Um zu einer manchmal sogenannten *strukturellen Induktion* zu kommen, muss man nun nur noch das Korsett der vollständigen Induktion über die Höhe der Bäume „vergessen“. Was bleibt ist folgende prinzipielle Situation:

1. Man beschäfigt sich mit irgendwelchen „Gebilden“ (eben waren das reguläre Ausdrücke bzw. Bäume). Dabei gibt es kleinste „atomare“ oder „elementare“ Gebilde (eben waren das die regulären Ausdrücke x für $x \in A$ und $\emptyset$)

und eine oder mehrere Konstruktionsvorschriften, nach denen man aus kleineren Gebilden größere zusammensetzen kann (eben waren das `*`, `|` und Konkatenation).

2. Man möchte beweisen, dass alle Gebilde eine gewisse Eigenschaft haben. Dazu macht man dann eine strukturelle Induktion:
   - Im Induktionsanfang zeigt man zunächst für *alle* „atomaren“ Gebilde, dass sie eine gewünschte Eigenschaft haben und
   - im Induktionsschritt zeigt man, wie sich bei einem „großen“ Gebilde die Eigenschaft daraus folgt, dass schon alle Untergebilde die Eigenschaft haben, gleich nach welcher Konstruktionsvorschrift das große Gebilde gebaut ist.

15.4 AUSBLICK

Beweise für die Behauptungen aus Satz 15.1 werden Sie vielleicht in der Vorlesung „Theoretische Grundlagen der Informatik“ oder in „Formale Systeme“ sehen. Insbesondere ist es dafür nützlich, sich mit nichtdeterministischen endlichen Automaten zu beschäftigen, auf die wir am Ende von Einheit 14 schon hingewiesen haben.

Sie werden sehen, dass reguläre Ausdrücke bei der Verarbeitung von Textdateien des öfteren nützlich sind. Dabei kommen zu dem, was wir in Abschnitt 15.1 definiert haben, zum einen noch bequeme Abkürzungen hinzu, denn wer will schon z. B.

```
a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
```

schreiben müssen als regulären Ausdruck für einen einzelnen Kleinbuchstaben. Zum anderen gibt es aber auch noch Erweiterungen, die dazu führen, dass die resultierenden *regular expressions* mächtiger sind als reguläre Ausdrücke. Wer sich dafür (jetzt schon) genauer interessiert, dem sei das Buch von Friedl (2006) empfohlen.

LITERATUR

Friedl, Jeffrey (2006). *Mastering Regular Expressions*. 3rd edition. O'Reilly Media, Inc.

Kleene, Stephen C. (1956). "Representation of Events in Nerve Nets and Finite Automata". In: *Automata Studies*. Hrsg. von Claude E. Shannon und John McCarthy. Princeton University Press. Kap. 1, S. 3–40.
Eine Vorversion ist online verfügbar; siehe http://www.rand.org/pubs/research_memoranda/2008/RM704.pdf (8.12.08).

