4 Automaten auf endlichen Bäumen

Am Anfang des vorherigen Kapitels haben wir gesehen, dass sich ein Wort als eine Menge von Positionen mit einer Nachfolgerfunktion auffassen lässt. Dies kann man in natürlicher Weise auf mehrere Nachfolgerfunktionen erweitern. So erhalt man eben Bäume, die in der Informatik mindestens so eine wichtige Rolle wie Wörter spielen, siehe z.B. Parse-Bäume kontext-freier Grammatiken, abstrakte Datentypen, XML-Dokumente, etc.

4.1 Top-down vs. Bottom-Up

Definition 36
Ein endlicher Baum ist eine präfix-abgeschlossene, endliche Teilmene T von $\mathbb{N}^*$, d.h. ist $wn \in T$ für ein $w \in \mathbb{N}^*$ und ein $n \in \mathbb{N}$, dann ist auch $w \in T$.

Im folgenden ist Σ wie üblich ein endliches Alphabet, aber mit einer Funktion $\sigma : \Sigma \to \mathbb{N}$, die jedem Symbol eine *Stelligkeit* zuordnet. Ein Σ-Baum ist eine partielle Abbildung $t : T \to \Sigma$ für einen Baum T, so dass für alle $w \in \mathbb{N}^*$, $a \in \Sigma$ und $n \in \mathbb{N}$ gilt: $w \in T$ und $t(w) = a$ und $\sigma(a) = n$ gdw. für alle $i \in \mathbb{N}$ gilt: $wi \in T$ gdw. $i < n$. .

Sei $\Sigma_n := \{a \in \Sigma \mid \sigma(a) = n\}$ und $\mathcal{T}_\Sigma$ die Menge aller Σ-Bäume.

Beachte: Bäume sind hier endlich-verzweigend, nicht nur, weil sie sowieso nur endlich sind, sondern auch, weil die Stelligkeitsfunktion des Alphabets auch nur endlich viele Nachfolgerknoten zulässt.

Beispiel 10
Sei $\Sigma = \{+, *, -, 0, 1, \cdot\}$ mit $\sigma(+) = \sigma(*) = \sigma(\cdot) = 2$, $\sigma(-) = 1$ und $\sigma(0) = \sigma(1) = 0$. Der arithmetische Ausdruck $-(7 * 3) + (-4 + 9)$ lässt sich in binärer Kodierung z.B. wie in Abbildung 4.1 als Σ-Baum modellieren.

Definition 37
Ein *Bottom-Up-Baumautomat* (BUBA) ist ein Tupel $\mathcal{A} = (Q, \Sigma, \delta, F)$ mit

- endlicher Zustandsmenge Q,
- Alphabet Σ mit Stelligkeitsfunktion σ,
- Transitionsrelationen $\delta = \{\delta_a \mid a \in \Sigma\}$ wobei $\delta_a : Q^n \to 2^Q$, falls $\sigma(a) = n$,
- Endzustandsmenge $F \subseteq Q$.

Ein Lauf von $\mathcal{A}$ auf einem Σ-beschrifteten Baum t ist eine Abbildung $r : dom(t) \to Q$, so dass für alle $w \in dom(t)$ gilt:

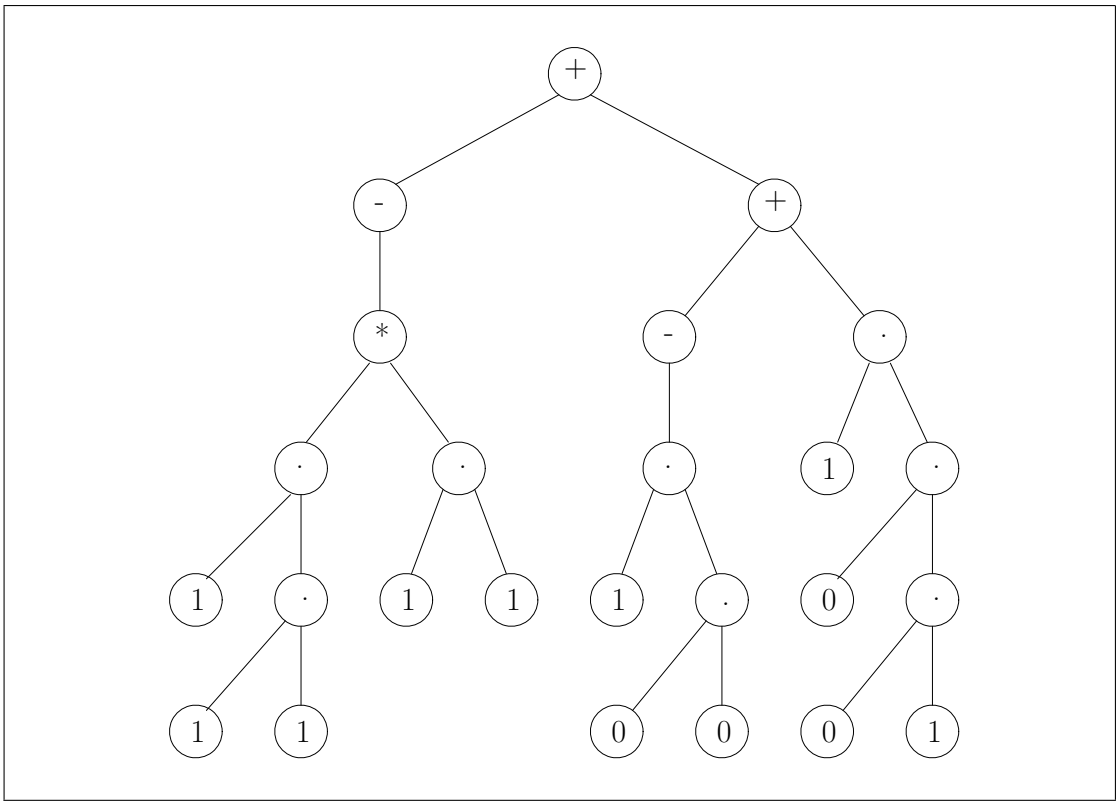

Abbildung 4.1: Ein arithmetischer Ausdruck als endlicher Baum.

- $r(w) \in \delta_a(r(w0), \ldots, r(w(n-1)))$ mit $a = t(w)$ und $n = \sigma(a)$ sonst.

Solch ein Lauf heißt *akzeptierend*, falls $r(\epsilon) \in F$. Wie üblich definieren wir $L(\mathcal{A}) \subseteq \mathcal{T}_\Sigma$ als die von $\mathcal{A}$ akzeptierte (Baum-)Sprache, also die Menge aller Σ-Bäume, auf den es einen akzpetierenden Lauf von $\mathcal{A}$ gibt.

Ein BUBA beschriftet also die Knoten eines Baums mit Zuständen. Er fängt bei den Blättern an und hört an der Wurzel auf.

Beispiel 11
Sei $\Sigma = \{\vee, \wedge, \neg, 0, 1\}$ mit $\sigma(\vee) = \sigma(\wedge) = 2$, $\sigma(\neg) = 1$ und $\sigma(0) = \sigma(1) = 0$. Die Sprache aller booleschen Ausdrücke, die zu 1 auswerten, ist BUBA-erkennbar. Dies wird z.B. von dem BUBA $\mathcal{A} = (\{0, 1\}, \Sigma, \delta, \{1\})$ mit

- $\delta_0() = \{0\}$,
- $\delta_1() = \{1\}$,
- $\delta_\neg(0) = \{1\}$, $\delta_\neg(1) = \{0\}$,
- $\delta_\vee(q_1, q_2) = \{q_1 \vee q_2\}$,

- $\delta_\wedge(q_1, q_2) = \{q_1 \wedge q_2\}$

getan.

Dieser Automat hat sogar noch eine besondere Eigenschaft: Er ist deterministisch.

Definition 38
Ein BUBA $\mathcal{A} = (Q, \Sigma, \delta, F)$ ist *deterministisch* (DBUBA), wenn für alle $a \in \Sigma$ und alle $q_1, \ldots, q_n \in Q$ mit $n = \sigma(a)$ gilt: $|\delta_a(q_1, \ldots, q_n)| = 1$.

Satz 50
Eine Baumsprache wird von einem BUBA erkannt, gdw. sie von einem DBUBA erkannt wird.

Beweis Richtung $\Leftarrow$ wird durch eventuelles Hinzufügen eines Zustands, aus dem nichts erkannt wird, erreicht. Richtung $\Rightarrow$ wird wie bei endlichen Wörtern durch eine einfache Potenzmengenkonstruktion gezeigt. ■

Definition 39
Ein *Top-Down-Baumautomat* (TDBA) ist ein Tupel $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ mit

- endlicher Zustandsmenge Q,
- Alphabet Σ mit Stelligkeitsfunktion σ,
- Anfangszustand $q_0 \in Q$,
- Transitionsfunktionen $\delta = \{\delta_a \mid a \in \Sigma\}$, wobei $\delta_a : Q \to 2^{Q^n}$ falls $\sigma(a) = n$,
- Endzustandszuweisung $F : \Sigma_0 \to 2^Q$.

Ein TDBA heißt *deterministisch*, falls $|\delta_a(q)| = 1$ für alle $a \in \Sigma$ und alle $q \in Q$.
Ein Lauf eines TDBA auf einem Baum t ist ein $r : dom(t) \to Q$ mit

- $r(\epsilon) = q_0$,
- $(r(w0), \ldots, r(w(n-1))) \in \delta_a(r(w))$ für alle $a \in \Sigma$ und alle $w \in dom(t)$ mit $t(w) = a$, $\sigma(a) = n$.

Solch ein Lauf heisst *akzeptierend*, falls $r(w) \in F(a)$ für alle Blätter w mit $t(w) = a$.

Ein TDBA beschriftet einen Baum also beginnend mit der Wurzel und endend in den Blättern.

Beispiel 12
Sei $\Sigma = \{a, b, c\}$ mit $\sigma(a) = \sigma(b) = 2$, $\sigma(c) = 0$. Die Sprache L aller Bäume, in denen mindestens ein b vorkommt, ist TDBA-erkennbar, z.B. von dem Automaten $\mathcal{A} = (\{-, +\}, \Sigma, -, \delta, F)$ mit

- $\delta_a(-) = \{(-, +), (+, -)\}$, $\delta_a(+) = \{(+, +)\}$,

- $\delta_b(-) = \{(+,+)\}$, $\delta_b(+) = \{(+,+)\}$,
- $\delta_b(-) = \{(+,+)\}$, $\delta_b(+) = \{(+,+)\}$,

und $F(c) = \{+\}$.

Satz 51
Die folgenden Aussagen sind für endliche Baumsprachen L über einem Alphabet Σ äquivalent.

1. *L wird von einem BUBA erkannt.*
2. *L wird von einem TDBA erkannt.*

Beweis Übung. ∎

Die Äquivalenz von TDBA und BUBA beruht darauf, dass Nichtdeterminismus zur Verfügung steht. Sei $a \in \Sigma_n$ für ein $n \in \mathbb{N}$. Dann ist die Transitionsrelation δ_a eines BUBA vom Typ $Q^n \times Q$. Beachte, dass $Q^n \times Q \simeq Q \times Q^n$ ist, welches der Typ einer Transitionsrelation δ_a eines TDBA ist. Bei einem DTDBA hat δ_a jedoch den Typ $Q \to Q^n$, und i.A. gilt $Q \to Q^n \not\simeq Q^n \to Q$.

Insbesondere gilt i.A.

$$|Q \to Q^n| \;=\; (|Q|^n)^{|Q|} \;=\; |Q|^{n\cdot|Q|} \;<\; |Q|^{(|Q|^n)} \;=\; |Q^n \to Q|$$

Satz 52
Es gibt reguläre Baumsprachen, die nicht von einem DTDBA erkannt werden.

Beweis Sei $\Sigma = \{a, b, c\}$ mit $\sigma(a) = 2$ und $\sigma(b) = \sigma(c) = 0$. Betrachte die endliche Baumsprache $L := \{a(b, c), a(c, b)\}$. Es ist leicht zu sehen, dass diese von einem TDBA erkannt wird.

Angenommen, $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ ist ein DTDBA mit $L(\mathcal{A}) = L$. Da $a(b, c) \in L(\mathcal{A})$ gibt es Zustände $q_1, q_2 \in Q$ mit $\delta_a(q_0) = \{(q_1, q_2)\}$, und es muss auch noch $q_1 \in F(b)$ und $q_2 \in F(c)$ gelten. Da aber auch $a(c, b) \in L(\mathcal{A})$ ist, gilt ebenfalls $q_1 \in F(c)$ und $q_2 \in F(b)$. Dann ist aber auch $\{a(b, b), a(c, c)\} \subseteq L(\mathcal{A})$. ∎

4.2 Reguläre Baumsprachen, Abschlusseigenschaften

Eine Baumsprache (Menge von Bäumen über dem gleichen Alphabet) ist *erkennbar*, wenn sie die Sprache eines BUBA ist.

Satz 53
Erkennbare Baumsprachen sind abgeschlossen unter den Operationen Vereinigung, Durchschnitt, Komplement.

Beweis wie bei Wörtern. Übung. ∎

Definition 40 (Baumkonkatenation)
Sei x ein 0-stelliges Element aus dem Alphabet Σ. Seien A und B Baumsprachen über Σ. Die Baumsprache $A[x{:=}B]$ (alternative Notation $A \cdot_x B$) ist wie folgt definiert:

$$\begin{aligned} A[x{:=}B] &:= \bigcup_{t\in A} t[x{:=}B] \\ x[x{:=}B] &:= B \\ y[x{:=}B] &:= y \quad \text{wenn } y \neq x \\ f(t_1 \dots t_k)[x{:=}B] &:= f(t_1[x{:=}B] \dots t_k[x{:=}B]) \end{aligned}$$

Intuitiv besteht $A[x{:=}B]$ aus allen Bäumen, die man erhält, indem man in einem Baum t aus A alle mit x beschrifteten Blätter durch Bäume aus B ersetzt (es dürfen unterschiedliche Bäume für verschiedene Vorkommen von x sein.).

Analog definiert man $A[x_1{:=}B_1, \dots, x_n{:=}B_n]$:

$$\begin{aligned} A[x_1{:=}B_1, \dots, x_n{:=}B_n] &:= \bigcup_{t\in A} t[x_1{:=}B_1, \dots, x_n{:=}B_n] \\ x_i[x_1{:=}B_1, \dots, x_n{:=}B_n] &:= B_i \\ y[x_1 = B_1, \dots, x_n = B_n] &:= y \quad \text{, wenn } y \notin \{x_1, \dots, x_n\} \\ f(t_1, \dots, t_k)[x_1{:=}B_1, \dots, x_n{:=}B_n] &:= f(t_1[s], \dots, t_k[s]), \text{ wobei } s = t_1{:=}B_1, \dots, t_k{:=}B_k \end{aligned}$$

Bemerkung: $A[x{:=}B, y{:=}C]$ ist im allgemeinen nicht dasselbe wie $A[x{:=}B][y{:=}C]$. Z.B.: $x[x{:=}y, y{:=}b] = y$, aber $x[x{:=}y][y{:=}b] = b$.

Satz 54
Sind $A, B_1, \dots, B_n$ erkennbar, so auch $A[x_1{:=}B_1, \dots, x_n{:=}B_n]$.

Beweis durch Konstruktion eines TDBA. ∎

Definition 41 (Baumstern)
Sei x ein einzelnes Symbol, L eine Baumsprache. Wir definieren die Baumsprache L^{*x} durch $L^{*x} = \bigcup_k L_k$, wobei $L_0 = \{x\}, L_{k+1} = L_k \cup L[x{:=}L_k]$.

Satz 55
Ist L erkennbar, so auch L^{*x}.

Beweis durch Konstruktion eines TDBA. ∎

Definition 42 (reguläre Baumsprache)
Eine Baumsprache L ist regulär, wenn sie aus den endlichen Sprachen mit Vereinigung, Baumkonkatenation, Baumstern erzeugt werden kann. Man schreibt $L \in \mathit{Reg}(\Sigma)$.

Satz 56
Eine Baumsprache L über Σ ist erkennbar genau dann, wenn eine endliche Menge 0-stelliger Hilfssymbole Z existiert, sodass $L \in \mathit{Reg}(\Sigma \cup Z)$. Beachte: die Symbole von Z kommen in L selbst nicht vor.

BEWEIS L werde von TDBA $\mathcal{A} = (\dots Q \dots)$ erkannt. Wir nehmen $Z = Q$ und definieren einen Automaten $\mathcal{A}'$ auf dem erweiterten Alphabet $\Sigma \cup Q$ (disjunkte Vereinigung) durch $F'(x) = F(x)$, falls $x \in \Sigma$ und $F'(q) = \{q\}$, falls $q \in Z$.

O.B.d.A. sei $Q = \{1, \dots, k\}$.

Sei K eine Teilmenge von Q, $h \leq k$ (möglicherweise $h = 0$), $i \in Q$. Wir bezeichnen mit $L(K, h, i)$ die Sprache über $\Sigma + K$, die von $\mathcal{A}'$ beginnend in i erkannt wird durch einen Lauf mit der Eigenschaft, dass innere Knoten (weder Wurzel noch Blatt) mit Zuständen $\leq h$ beschriftet sind. Insbesondere bedeutet das, dass Blätter $x \in \Sigma$ mit Endzuständen aus $F(x)$ beschriftet sind und dass Zustandsblätter $q \in K$ mit sich selbst beschriftet sind (wg. $F'(q) = \{q\}$).

Die Sprache $L(K, 0, q)$ ist endlich und somit erkennbar. Außerdem ist

$$L(K, h{+}1, q) = L(K, h, q) \cup L(K \cup \{h{+}1\}, h, q)[h{+}1 := U]$$
$$\text{wobei } U = L(K \cup \{h{+}1\}, h, h{+}1)^{*(h+1)}[h{+}1 := L(K, h, h{+}1)]$$

Dies sieht man dadurch, dass man in einem Element von $L(K, h{+}1, q)$ alle Vorkommnisse des Zustands $h + 1$ herauspräpariert.

Somit sind alle $L(K, h, q)$ regulär und damit auch $L(\mathcal{A}) = L(\emptyset, k, q_0)$. ∎

Satz 57
Für reguläre Baumsprachen L_1, L_2 (präsentiert durch Automat oder Ausdruck) sind folgende Probleme entscheidbar: $L_1 = \emptyset, L_1 = L_2, L_1 \subseteq L_2$.

BEWEIS Es genügt zu entscheiden, ob ein DBUBA die leere Sprache erkennt. Alles andere lässt sich darauf zurückführen. Sei also ein DBUBA $\mathcal{A}$ gegeben. Wir bestimmen iterativ die Menge M aller Zustände q, sodass es einen Baum t gibt mit $\mathcal{A}(t) = q$. Sei U_0 die Menge aller Zustände, die als Anfangsbeschriftung in Erscheinung treten. Ist U_n schon definiert, so sei U_{n+1} die Menge U_n zusammen mit allen Zuständen q, derart, dass es ein Symbol f gibt und Zustände $q_1, \dots q_k \in U_n$ mit $\delta(f, q_1, \dots q_k) = q$. Die gesuchte Menge M ist die Vereinigung aller U_n. Nach spätestens $|Q|$-Schritten ist aber diese Vereinigung erreicht, sodass sie in polynomialer Zeit berechnet werden kann. Der Automat akzeptiert nun die leere Sprache, genau dann, wenn $M \cap F = \emptyset$. ∎

Dieses Verfahren läuft in polynomialer Zeit, wenn die Ausgangssprache als DBUBA gegeben ist. Ist er hingegen als regulärer Ausdruck (möglicherweise gar mit Negation) oder als nichtdet. Automat gegeben, so muss zunächst determinisiert werden. Man kann zeigen [Seidl90], dass das Leerheitsproblem für in diesen Formen gegebene Sprachen EXPTIME vollständig ist (zum Vergleich PSPACE vollständig bei Wörtern).

4.3 Komplementierung von Topdown-Baumautomaten

Deterministische BUBA sind leicht zu komplementieren durch Vertauschen von End- und Nicht-Endzuständen. Für Topdown-Automaten ist es nicht so leicht, da sie erstens i.a. nichtdeterministisch sind und zweitens die Akzeptanzbedingung lautet: jedes Blatt ist mit einem passenden Endzustand beschriftet; davon ist das Komplement aber "es

gibt ein Blatt, das mit einem unpassenden Endzustand beschriftet ist", was nicht von derselben Form ist.

Es ist nützlich, sich zu überlegen, wie man Topdown-Automaten direkt, ohne den Umweg über Bottomup-Automaten komplementieren kann, da sich nur die Topdown-Automaten auf unendliche Bäume verallgemeinern lassen.

Sei also ein nichtdeterministischer Topdownautomat $\mathcal{A}$ und ein Baum $t = (T, t)$ gegeben. Sei n die maximale Stelligkeit und $D = \{0, \ldots, n-1\}$ die Menge der möglichen Richtungen, in die man in einem Knoten verzweigen kann. Positionen im Baum lassen sich so als bestimmte Wörter über D^* auffassen. Wenn $w \in D^*$, so bezeichnen wir wie üblich mit $t(w)$ die Beschriftung des durch w adressierten Knotens in t. Wenn $t(w) = a$ und $\sigma(a) = k$, dann ist $k \leq n$ und wenn $\{i \mid wi \in T\} = \{0, \ldots, k-1\}$.

Wir betrachten folgendes Zweipersonenspiel zwischen den Spielern A ("Automat") und P ("Pfadfinder").

1. Positionen des Spiels sind Paare der Form (w, q) wobei $w \in D^* \cap T$ und $q \in Q$, sowie Paare der Form $(w, \vec{q})$, wobei $w \in D^* \cap T$ und $\vec{q} \in Q^{\sigma(t(w))}$.

   Bei Positionen der Form (w, q) ist A am Zug; bei Positionen der Form $(w, \vec{q})$ ist P am Zug.

2. Startposition ist (ϵ, i), wobei i der Anfangszustand ist.

3. In der Position (w, q) wählt A ein Tupel $\vec{q} \in \delta_{t(w)}(q)$ und erreicht die Position $(w, \vec{q})$.

4. In der Position $(w, \vec{q})$ wählt P eine Richtung $i < m$, wobei $m = \sigma(t(w))$ die Stelligkeit des Symbols $t(w)$ ist und es wird die Position (wi, q') eingenommen, wobei q' die i-te Komponente des Vektors $\vec{q}$ ist.

5. Das Spiel endet, wenn eine Position (w, q) erreicht wird, wobei $t(w)$ ein Blatt ist, also $\sigma(t(w)) = 0$. Es gewinnt dann A, falls $q \in F(t(w))$ und es gewinnt P, falls $q \notin F(t(w))$.

Satz 58
Ein Baum wird genau dann von $\mathcal{A}$ akzeptiert, wenn Spieler A eine Gewinnstrategie besitzt.

Beweis Übung. ∎

Satz 59
Hat A keine Gewinnstrategie, so besitzt P eine positionale Gewinnstrategie, d.h. es gibt eine Funktion

$$s : D^* \times Q^{\leq n} \to D$$

sodass P gewinnt, falls er in der Position $(w, \vec{q})$ die Richtung $s(w, q, \vec{q})$ wählt.

Beweis Durch Induktion über die Tiefe des Baumes. ∎

Unendliche Spiele Die Existenz positionaler Gewinnstrategien gilt bei allen endlichen Spielen und kann durch Induktion über die Spieldauer nachgewiesen werden (ist man in einer Position von der aus man gewinnen kann, so ziehe man auf eine Position von der aus das auch noch der Fall ist). Bei unendlichen Spielen kann es erforderlich sein, sich gewisse Daten über die Spielhistorie zu merken und in die Entscheidungen über den nächsten Zug einfließen zu lassen. Ein Beispiel bildet das Dziembowski-Jurdziński-Walukiewicz-Spiel, bei dem zwei Spieler abwechselnd und ad infinitum Buchstaben in $\{A, B, C, D\}$ (Buchstabenspieler) und Zahlen aus $\{1, 2, 3, 4\}$ (Zahlenspieler) nennen. Der Zahlenspieler gewinnt, wenn die größte von ihm unendlich oft gespielte Zahl mit der Anzahl der vom Buchstabenspieler unendlich oft gespielten Buchstaben übereinstimmt. Man kann sich überlegen, dass der Zahlenspieler hier eine Gewinnstrategie hat, aber keine positionale.

Anwendung von Currying Die Gewinnstrategie in unserem Spiel kann als Funktion aufgefasst werden, die zu jeder Baumposition $w \in D^*$ eine Funktion $s = s_w : Q^{\leq n} \to D$ liefert.

Wir definieren nun $S := D^{Q^{\leq n}}$ als die *endliche* Menge aller solcher Funktionen.

Nun wird also t nicht von A akzeptiert, wenn P eine Gewinnstrategie besitzt, es also eine Beschriftung des Baumes t mit Elementen von S gibt, derart, dass für alle (von A gespielten) Transitionsfolgen $\vec{m} \in (Q^{\leq n})^*$ der von $\vec{m}$ und der Strategiebeschriftung induzierte Pfad w in einer Gewinnposition für P landet. Wir können nun die Quantifikation über die Pfade nach außen ziehen:

Der Baum t wird nicht von A akzeptiert, genau dann wenn für alle Pfade w in t folgendes gilt:

Für alle Transitionsfolgen $\vec{m} \in (Q^{\leq n})^*$, die erstens legal sind und zweitens (zusammen mit den Strategiebeschriftungen) den gegebenen Pfad w induzieren, gilt, dass eine Gewinnposition für P erreicht wird.

Dies soll durch einen nichtdeterministischen Topdownbaumautomaten ausgedrückt werden. Die Form sieht schon mal ganz gut aus: Eine universelle Quantifizierung über Pfade steht ganz außen. Wir gehen wie folgt vor. Zunächst bauen wir einen Automaten, der auf mit $\Sigma \times S$ beschrifteten Bäumen arbeitet. Intuitiv muss dieser Automat für jeden Pfad w abprüfen, ob für jedes gleichlange Wort aus Transitionsfolgen welches diesen Pfad induziert, die induzierte Zustandsfolge in A in einem Zustand q mit $q \notin F(x)$ landet (x die Blattbeschriftung am Ende des Pfades).

Diese Sprache über Pfaden ist regulär und kann deshalb mit einem endlichen Wortautomaten, den man auf jedem Pfad mitlaufen lässt überprüft werden. Genauer gesagt, sei $\mathcal{M}$ ein DFA über $\Sigma \times S \times D$, der folgende Sprache erkennt:

$$\begin{aligned} L(\mathcal{M}) = \{&(a_1, s_1, d_1) \dots (a_l, s_l, d_n) \mid \forall \vec{q_1} \dots \vec{q_l}. \forall q_1 \dots q_n. \\ &q_1 = i \Rightarrow \forall 1 \leq i \leq l. \vec{q_i} \in \delta_{a_i}(q_i) \Rightarrow \forall 1 < i \leq l. d_i = s_i(\vec{q_{i-1}}) \Rightarrow \\ &q_i = (\vec{q_{i-1}})_{d_i} \Rightarrow \sigma(a_l) = 0 \Rightarrow q_l \notin F(a_l)\} \end{aligned}$$

Wir bauen als nächstes einen Automaten $\mathcal{A}'$, der einen Baum beschriftet mit $\Sigma \times S$ erkennt, genau dann, wenn alle Pfade in $L(\mathcal{M})$ sind. Dieser Automat hat dieselben Zustände wie $\mathcal{M}$ und die Übergangsfunktion (der Automat ist deterministisch!)

$$\delta^{\mathcal{A}'}_{(a,s)}(q) = (\delta^{\mathcal{M}}((a,s,0),q),\ldots,\delta^{\mathcal{M}}((a,s,\sigma(a)-1),q))$$

Den ersehnten Automaten $\mathcal{A}''$ über Σ für das Komplement von $\mathcal{A}$ erhalten wir aus $\mathcal{A}'$, indem wir die Strategiebeschriftung raten. Die Zustände von $\mathcal{A}''$ sind dieselben, wie die von $\mathcal{A}'$, die Übergangsrelation ist definiert durch

$$\delta^{\mathcal{A}''}_a(q) = \{\vec{q} \mid \exists s.\vec{q} = \delta^{\mathcal{A}'}_{(a,s)}(q)\}$$

Dieser Automat erkennt also wie gewünscht das Komplement von $L(\mathcal{A})$.

Wir können die verwendeten Konstruktionen etwas abstrakter fassen:

Lemma 31
Sei $L \in Reg(\Sigma \times D)$ eine reguläre Wortsprache. Die assoziierte Baumsprache L^t besteht aus allen Bäumen t, sodass jeder Pfad in t geschrieben als Wort über $\Sigma \times D$ in L ist. Die Sprache L^t ist reguläre Baumsprache.

Beweis Übung ■

Lemma 32
Sei $p : \Sigma \to \Sigma'$ stelligkeitserhaltende Funktion und $L \in Reg(\Sigma)$ reguläre Baumsprache. Die Baumsprache $p(L) = \{t \mid \exists t' \in L.p(t') = t\}$ ist regulär.

Hier bezeichnet $p(t')$ den Baum, den man aus t' erhält, indem man die Beschriftungen in t' gemäß p ersetzt.

Beweis Übung. ■

4.4 Higher-order matching

Wir betrachten den einfach typisierten Lambdakalkül (funktionale Programme ohne Rekursion mit Typen

$$\tau ::= o \mid \tau_1,\ldots,\tau_n \to \tau$$

Wir schreiben $\tau_1^n \to \tau_2$ als Abkürzung für $\underbrace{\tau_1,\ldots,\tau_1}_{n \text{ Stück}} \to \tau_2$. Wir setzen ein Baumalphabet Σ voraus, wobei wir f mit Stelligkeit n identifizieren mit einer Funktion des Typs $o^n \to o$. Insbesondere identifizieren wir ein Blattsymbol mit einer Konstanten des Typs o. Wir wollen nun Matchinggleichungen dritter Ordnung lösen, insbesondere interessieren wir uns für die Lösungsmenge einer Gleichung

$$x(s_1,s_2,\ldots,s_n) = t$$

wobei s_i variablenfreie Terme von Typen der Form $\tau_i = o,\ldots,o \to o$ sind und t variablenfreier Term vom Typ o ist. Demgemäß ist x eine Variable vom Typ $\tau_1,\ldots,\tau_n \to o$. Die Symbole aus Σ gelten nicht als Variablen.

Beispiel 13
Man bestimme alle Terme, die die Gleichung

$$x(\lambda y_1, y_2.y_1, \lambda y_3.f(y_3, y_3)) = f(a, a)$$

lösen. Zwei mögliche Lösungen sind $x = \lambda x_1, x_2.x_2(a)$ und $x = \lambda x_1, x_2.x_1(f(a, a), a)$. Eine andere Lösung ist $x = \lambda x_1, x_2.x_1(x_2(x_1(x_1(a, \Box), \Box)), \Box)$ wobei für jedes Vorkommen von $\Box$ ein beliebiger Term eingesetzt werden kann.

Fassen wir die λ-gebundenen Variablen als 0-stellige Symbole auf und das Präfix $\lambda x_1, \ldots, x_k.$ als einstelliges Symbol, so können wir λ-Terme mit Baumautomaten verarbeiten. Hier ist ein Baumautomat mit dem zusätzlichen Symbol $\Box$, der gerade alle schematischen Lösungen der gegebenen Gleichungen erkennt.

Es gibt vier Zustände, A, B, F, E

Die Übergänge sind:

$$\begin{array}{ll} a \to A & f(A, A) \to F \\ \Box \to B & x_1(A, B) \to A \\ x_1(F, B) \to F & x_2(A) \to F \\ \lambda x_1, x_2.F \to E & \end{array}$$

Endzustand ist der Zustand E.

Wir verarbeiten $\lambda x_1, x_2.x_1(f(a, a), \Box)$ wie folgt:

$$\begin{array}{l} \lambda x_1, x_2.x_1(f(a, a), \Box) \to \\ \lambda x_1, x_2.x_1(f(A, A), B) \to \\ \lambda x_1, x_2.x_1(F, B) \to \\ \lambda x_1, x_2.F \to E \end{array}$$

Wir verarbeiten $\lambda x_1, x_2.x_1(x_2(x_1(x_1(a, \Box), \Box)), \Box)$ wie folgt:

$$\begin{array}{l} \lambda x_1, x_2.x_1(x_2(x_1(x_1(a, \Box), \Box)), \Box) \to \\ \lambda x_1, x_2.x_1(x_2(x_1(x_1(A, B), B)), B) \to \\ \lambda x_1, x_2.x_1(x_2(x_1(A, B)), B) \to \\ \lambda x_1, x_2.x_1(x_2(A), B) \to \\ \lambda x_1, x_2.x_1(F, B) \to \\ \lambda x_1, x_2.F \to E \end{array}$$

Versuchen wir dagegen $\lambda x_1, x_2.x_2(x_1(f(a, a), \Box))$ zu verarbeiten, so erhalten wir

$$\lambda x_1, x_2.x_2(x_1(f(a, a), \Box)) \to \lambda x_1, x_2.x_2(x_1(f(A, A), B)) \to \lambda x_1, x_2.x_2(x_1(F, B)) \to \lambda x_1, x_2.x_2(F)$$

und wir bleiben stecken. Die Zustände entsprechen Teiltermen der rechten Seite: $A = a$, $F = f(a, a)$. Außerdem haben wir den Sonderzustand B für $\Box$ und E für Ende. Die Idee ist, dass ein Term t mit zwei Variablen x_1 und x_2 im Automaten den Zustand A (bzw. F) liefert, gdw $t[x_1 := \lambda y_1, y_2.y_1, x_2 := \lambda y_3.f(y_3, y_3)]$ sich zu a (bzw. $f(a, a)$) reduziert (per $\beta\eta$-Reduktion). Außerdem ist der Wert von t der Zustand B, falls sich $t[x_1 := s_1, x_2 := s_2]$ zu $\Box$ reduziert.

Dies beweist man durch Induktion über t (in $\beta\eta$-Normalform vorausgesetzt.)

Betrachten wir nun den allgemeinen Fall einer Matchinggleichung $x(s_1, s_2, \ldots, s_n) = t$.

Für jeden Teilterm u der rechten Seite t führen wir einen Zustand q_u ein und dazu noch zwei Sonderzustände $q_\Box$ und E. Die Regeln sind wie folgt:

$\lambda x_1, \ldots, x_n.q_t \to E$
$\Box \to q_\Box$
$f(q_{u_1}, \ldots, q_{u_n}) \to q_v$, falls $v = f(u_1, \ldots, u_n)$ ein Teilterm der rechten Seite ist. Hier $u_i, v \neq \Box$
$x_i(q_{u_1}, \ldots, q_{u_n}) \to q_v$, falls $s_i(u_1, \ldots, u_n) =_{\beta\eta} v$. Hier $v \neq \Box$, aber vielleicht $u_i = \Box$

Diese Methode lässt sich auf beliebige Matchinggleichungen dritter Ordnung verallgemeinern, also Gleichungen der Form a(x1,...,xn) = b, wobei x_i Variablen dritter Ordnung sind, etwa mit Typen wie $(o \to o) \to o$, und a, b variablenfrei sind.

Die Entscheidbarkeit des Matchingproblems für beliebige Ordnung ist ein berühmtes offenes Problem. In 2006 hat C. Stirling eine Arbeit vorgelegt, in der die Entscheidbarkeit gezeigt wird. Sollte diese Arbeit Bestand haben, so wäre dieses Problem gelöst.

4.5 Baumautomaten und XML

Weitere wichtige Anwendungen für Baumautomaten ergeben sich im Bereich semistrukturierter Daten (XML). Sie werden hier insbesondere zur Typüberprüfung (Typen = XML Schemas) von Dokumenten und Transformationsskripten eingesetzt. Sind solche Transformationsskripten in funktionalem Stil geschrieben, so kann man eine Kombination von ML-Typüberprüfung und Baumautomaten einsetzen um eine approximative Typprüfung im folgenden Sinne durchzuführen:

Gegeben seien Schemas `In` und `Out`, sowie ein Transformationsskript P. Bestätigt die Typüberprüfung für P den Typ `In` $\to$ `Out`, so liefert P für Eingaben des Typs `In` stets Ausgaben des Typs `Out`. Es gibt aber Skripten P, die diese semantische Eigenschaft haben und dennoch von der Typüberprüfung zurückgewiesen werden.

Der Vorteil dieses Ansatzes ist, dass die Typüberprüfung recht einfach ist und gleichzeitig die einsetzbare Programmiersprache universell ist.

Alternativ kann man eine exakte Analyse erreichen, indem man die erlaubten Skripten einschränkt, etwa auf Baumautomaten mit Ausgabe und gleichzeitig die Analyse komplizierter macht.

Als inzwischen etwas veraltetes konkretes Beispiel zeigen wir hier die von Pierce und Hosoya entwickelte Programmiersprache Xduce.

Xduce ist eine funktionale Programmiersprache zur Verarbeitung von XML-Dokumenten. XML-Daten können sowohl als Ein- als auch als Ausgabe einer Funktion erscheinen. Es gibt aber auch noch andere Datentypen wie float, String etc. Für XML-Daten gibt es benutzerdefinierte Typen, die im wesentlichen den XML-Schemas entsprechen.

Dies ist eine Xduce Typdefinition:

```
type Addrbook = addrbook[Person*]
type Person = person[(Name,Tel?,Email* )]
type Name = name[String]
```

```
type Tel = tel[String]
type Email = email[String]

type Addrbook2 = addrbook[Person*]
type Person = person[(Name,Tel*,Email* )]
type Name = name[String]
type Tel = tel[String]
type Email = email[String]
```

Hier ist ein XML-Dokument, das zu ihr passt, also den Typ `Addrbook` hat:

```
<addrbook>
  <person>
    <name>Haruo Hosoya</name>
    <email>hahosoya</email>
    <email>haruo</email>
  </person>
  <person>
    <name>Jerome Vouillon</name>
    <tel>123-456-789</tel>
    <email>vouillon</email>
  </person>
  <person>
    <name>Benjamin Pierce</name>
    <email>pierce</email>
  </person>
</addrbook>
```

Und hier ist dieselbe Typdefinition als DTD (Document Type Definition).

```
<!DOCTYPE Addrbook
 [ <!ELEMENT addrbook( person*) >
   <!ELEMENT person (name, tel?, email*) >
   <!ELEMENT name (\# PCDATA)>
   <!ELEMENT tel (\# PCDATA)>
   <!ELEMENT email (\# PCDATA)>
]>
```

und natürlich können wir auch einen Baumautomaten verwenden, dessen Sprache gerade die XML-Dokumente des Typs Addrbook sind. Ein kleines Problem hierbei ist, dass die Symbole in XML, wie `addrbook`, variable Stelligkeit besitzen. Man kann das dadurch umgehen, dass man alle Symbole nur ein oder zweistellig erlaubt und ggf. neue künstliche Typabkürzungen einführt, vgl. Chomsky Normalform kontextfreier Grammatiken. In Xduce werden XML-Dokumente *intern* in die feststellige Notation umgerechnet und verarbeitet. Alternativ kann man auch die Baumautomaten auf die variable Stelligkeit erweitern.

Die folgende Xduce Funktion übersetzt nun eine Folge von Adresseinträgen in ein Telefonbuch. Die gegebenen Typannotate werden automatisch überprüft.

```
fun mkTelList (val e as (Name,Addr,Tel?)*) : (Name,Tel)* =
```

```
match e with
  name[val n], addr[val a], tel[val t], val rest
    -> name[n], tel[t], mkTelList(rest)
| name[val n], addr[val a], val rest
    -> mkTelList(rest)
| ()
    -> ()
```

Startpunkte für weiterführende Lektüre zu diesen Themen sind die WWW-Seiten von Haruo Hosoya (approximative Analyse für beliebige Programme), Thomas Schwentick (exakte Analyse für Transducer), Frank Neven (allgemeines zu XML und Automatentheorie), Michael Schwartzbach (allgemeines zu praktischen Aspekten von XML aus Sicht eines Theoretikers).