Kapitel 3

Baum-Automaten

3.1 Baum-Automaten

Def. Ein Tupel $A = (\Sigma, Q, Q_F, \Delta)$ ist Baumautomat, wenn

1. Σ ist Signatur
2. Q ist endliche Menge (Zustände)
3. $Q_F \subseteq Q$ (akzeptierende Zustände)
4. $\Delta \subseteq \bigcup_k \Sigma_k \times Q^k \times Q$ (Übergangsrelation)

Bsp. Signatur $\Sigma = \{f, a, b\}$ mit f zweistellig und a, b nullstellig. Zustandsmenge $Q = \{0, 1\}$, akzeptierende Zustände $Q_F = \{0\}$. Übergangsrelation

$$\begin{aligned} a \to 1, b \to 0 \\ f(0,0) \to 0, f(1,1) \to 0, f(0,1) \to 1, f(1,0) \to 1 \end{aligned}$$

Def. Ein Lauf von A auf t ist eine Abbildung $r : \mathsf{Pos}(t) \to Q$ (von Position nach Zustand) mit

$$\forall p \in \mathsf{Pos}(t) : \text{wenn} t[p] = f \in \Sigma_k, \text{dann} f(r(p1), \ldots, r(pk)) \to r(p) \in \Delta$$

(der Lauf ist kompatibel mit der Übergangsrelation).

Bsp. Ein Lauf des vorigen Automaten auf $t = f(f(a, a), a)$:

$$\begin{aligned} r : 11 \mapsto 1, r : 12 \mapsto 1 \\ r : 1 \mapsto 0, r : 2 \mapsto 1, r : \epsilon \mapsto 1 \end{aligned}$$

Def. Ein akzeptierender Lauf ist ein Lauf r mit $r(\epsilon) \in Q_F$ (die Wurzel ist mit einem akzeptierenden Zustand beschriftet).

Bsp. der obige Lauf ist nicht akzeptierend.

Def. Ein Automat A akzeptiert einen Baum t, wenn ein akzeptierender Lauf von A auf t existiert.

Def. Die Menge $\mathsf{Lang}(A)$ aller von A akzeptierten Bäume.

Übg. beschreibe $\mathsf{Lang}(A)$ für das Beispiel.

Def. Eine Menge L von Bäumen heißt regulär, wenn ein A existiert, der L akzeptiert.

Def. Die Menge aller regulären Baumsprachen heißt REG.

3.2 Varianten von Automaten

3.2.1 Vollständige Automaten

Def. Ein Automat A heißt vollständig: $\forall t \in \mathsf{Tree}(\Sigma) : \exists$ Lauf r von A auf t.

Satz: Zu jedem Automaten A gibt es einen vollständigen Automaten A' mit $\mathsf{Lang}(A) = \mathsf{Lang}(A')$.

Beweis: einen neuen Zustand, der nicht akzeptiert.

Vorsicht: Das ist zwar nur ein Zustand mehr, aber es können sehr viele Regeln hinzukommen.

3.2.2 Schlanke Automaten

Def. Die vom Zustand $q \in Q$ eines Automaten $A = (\Sigma, Q, Q_F, \Delta)$ akzeptierte Sprache ist $\mathsf{Lang}(A, q) = \mathsf{Lang}(\Sigma, Q, \{q\}, \Delta)$.

Def. Im Automaten $A = (\Sigma, Q, Q_F, \Delta)$ heißt ein Zustand $q \in Q$ produktiv, wenn $\emptyset \neq \mathsf{Lang}(A, q)$.

D. h. der Zustand kann wenigstens einen Baum akzeptieren. Beachte: die Menge Q_F wird für die Definition nicht benutzt.

Satz: Die Menge der produktiven Zustände ist die kleinste Menge $M \subseteq Q$ mit $\forall f(q_1, \ldots, q_k) \to q \in \Delta$: wenn $\forall i : q_i \in M$, dann $q \in M$.

Def. Die Menge der erreichbaren Zustände des Automaten $A = (\Sigma, Q, Q_F, \Delta)$

ist die kleinste Menge $M \subseteq Q$ mit:

1. $Q_F \subseteq M$
2. $\forall f(q_1, \ldots, q_k) \to q \in \Delta$: wenn $q \in M$, dann $\forall i : q_i \in M$.

D. h. die Zustände in M sind von der Wurzel (den akzeptierenden Zuständen) aus erreichbar. Beachte: es wird nicht geprüft, ob die Zustände aus M produktiv sind.

Übg: Finde Automaten, bei denen nicht alle erreichbaren Zustände produktiv sind, und umgekehrt.

Def. Ein Automat ist schlank: alle Zustände sind erreichbar und produktiv.

Satz: Zu jedem Automaten A gibt es einen schlanken Automaten C mit $\mathsf{Lang}(A) = \mathsf{Lang}(C)$.

Beweis: Wir erhalten C aus A durch Löschen von Zuständen: Wir löschen in A die nicht produktiven Zustände (und die dazugehörigen Regeln), und erhalten B. In B löschen wir die nicht erreichbaren Zustände (und dazugehörende Regeln) und erhalten C.

Übung: ist es auch korrekt, erst die nicht erreichbaren Zustände zu löschen und dann die nicht produktiven? (Nein. Beispiel?)

3.2.3 Deterministische Automaten

Def. Ein Automat $A = (\Sigma, Q, Q_F, \Delta)$ heißt deterministisch: Δ ist eine partielle Funktion: $\bigcup_k \Sigma_k \times Q^k \hookrightarrow Q$.

Bem: Die Markierung einer Position ergibt sich eindeutig aus dem Funktionssymbol und den Markierungen der Argumentpositionen.

Satz: Wenn A vollständig ist, dann $\forall t \in \mathsf{Tree}(\Sigma) : \exists$ höchstens ein Lauf von A auf t.

Übg: sind beide Forderungen sogar äquivalent? Nein. Beispiel?

Satz: Zu jedem Automaten A gibt es einen vollständigen deterministischen Automaten A' mit $\mathsf{Lang}(A) = \mathsf{Lang}(A')$.

Beweis: Sei $A = (\Sigma, Q, Q_F, \Delta)$. Dann ist $A' = (\Sigma, Q', Q'_F, \Delta')$ mit $Q = 2^Q$. Der Rest ist Übung.

Bem: Die Zustandsmenge wird im schlimmsten Fall exponentiell größer. Solche Fälle kommen leider auch wirklich vor.

Clevere Implementierungen erzeugen von vornherein nur produktive Zustände aus 2^Q, aber das hilft manchmal nichts.

3.2.4 Top-Down-Automaten

Def. Ein Tupel $A = (\Sigma, Q, Q_F, \Delta)$ ist ein Top-Down-Automat:

1. Σ ist Signatur
2. Q ist endliche Menge (Zustände)
3. $Q_F \subseteq Q$ (akzeptierende Zustände)
4. $\Delta \subseteq \bigcup_k Q \times \Sigma_k \times Q^k$ (Übergangsrelation)

Bsp: die Regeln sehen jetzt so aus: $q \to f(q_1, \ldots, q_k)$.

Def. Der Top-Down-Automat $A = (\Sigma, Q, Q_F, \Delta)$ akzeptiert die gleiche Sprache wie der (Bottom-Up-)Automat $A' = (\Sigma, Q, Q_F, \Delta')$ mit

$$\Delta' = \{f(q_1, \ldots, q_k) \to q \mid q \to f(q_1, \ldots, q_k) \in \Delta\}.$$

Def. Ein Top-Down-Automat $A = (\Sigma, Q, Q_F, \Delta)$ heißt deterministisch: Δ ist eine partielle Funktion: $\bigcup_k Q \times \Sigma_k \hookrightarrow Q^k$.

Bem: Die Markierung der Argumentpositionen ergibt sich eindeutig aus dem Funktionssymbol und der Markierung der Position. Vergleiche mit deterministischen Bottom-Up-Automaten.

Es gibt Top-Down-Automaten A, für die kein deterministischer Top-Down-Automat A' existiert mit $\mathsf{Lang}(A) = \mathsf{Lang}(A')$. (Beispiel!)

Das ist ein Unterschied zu Wortsprachen. Dort können Automaten von links nach rechts und von rechts nach links deterministisch gemacht werden (Beispiel!)

3.2.5 ϵ-Automaten

Def. Ein ϵ-Automat ist ein Tupel $A = (\Sigma, Q, Q_F, \Delta, E)$ mit

1. Σ ist Signatur
2. Q ist endliche Menge (Zustände)

3. $Q_F \subseteq Q$ (akzeptierende Zustände)

4. $\Delta \subseteq \bigcup_k \Sigma_k \times Q^k \times Q$ (Übergangsrelation)

5. $E \subseteq Q \times Q$ (ϵ-Übergänge)

Def. Ein Lauf eines ϵ-Automaten $A = (\Sigma, Q, Q_F, \Delta, E)$ auf einem Baum t ist eine Abbildung $\mathsf{Pos}(t) \to Q^+$ mit

1. $\forall p \in \mathsf{Pos}(t)$: wenn $t[p] = f \in \Sigma_k$, dann

$$f(\mathsf{last}(r(p1)), \ldots, \mathsf{last}(r(pk))) \to \mathsf{first}(r(p)) \in \Delta.$$

2. $\forall p \in \mathsf{Pos}(t)$: wenn $r(p) = (q_1, \ldots, q_k)$, dann $\forall 1 \leq i < k : (q_i, q_{i+1}) \in E$.

An jeder Position steht nun eine nichtleere Liste von Zuständen.

Def. Ein akzeptierender Lauf eines ϵ-Automaten $A = (\Sigma, Q, Q_F, \Delta, E)$ auf einem Baum t ist ein Lauf r mit $\mathsf{last}(r(\epsilon)) \in Q_F$.

Def. Die Menge $\mathsf{Lang}(A)$ der von A akzeptierten Bäume.

Satz: Zu jedem Automaten $A = (\Sigma, Q, Q_F, \Delta)$ gibt es einen ϵ-Automaten $A' = (\Sigma, Q, Q_F, \Delta, E)$ mit $\mathsf{Lang}(A) = \mathsf{Lang}(A')$. Beweis: $E = \emptyset$.

Satz: Zu jedem ϵ-Automaten $A' = (\Sigma, Q, Q_F, \Delta, E)$ gibt es einen Automaten $A = (\Sigma, Q, Q_F, \Delta')$ mit $\mathsf{Lang}(A) = \mathsf{Lang}(A')$. Beweis: zu jedem Zustand die ϵ-Hülle bilden.

Manchmal ist ein ϵ-Automat leichter hinzuschreiben als zu schreiben und zu lesen als der äquivalente Automat ohne Epsilon-Übergänge.

Wir werden ϵ-Automaten aber nicht gesondert untersuchen, da sie auch genau die Sprachen aus REG akzeptieren.

3.3 Eigenschaften von Automaten

3.3.1 Membership

Satz: Für einen Automaten $A = (\Sigma, Q, Q_F, \Delta)$ und einen Baum $t \in \mathsf{Tree}(\Sigma)$ ist die Frage $t \in \mathsf{Lang}(A)$ entscheidbar.

Beweis: Wir müssen in der Menge aller Läufe von A auf t einen akzeptierenden finden. Wir beschriften, von den Blättern beginnend, jede Position p in t mit der Menge der Zustände, die ein Lauf r in p annehmen kann. D. h. wir führen die Potenzmengenkonstruktion "live" aus.

Übg: Laufzeit als Funktion der Eingabegröße.

3.3.2 Leere

Satz: Für einen Automaten $A = (\Sigma, Q, Q_F, \Delta)$ ist die Frage $\emptyset = \mathsf{Lang}(A)$ entscheidbar.

Beweis: Wir prüfen, ob Q_F wenigstens einen produktiven Zustand enthält.

Übg: Laufzeit.

3.3.3 Das Pumping-Lemma

Wie bei regulären Wortsprachen gibt es auch für endliche Baumautomaten eine Aussage der Art: jede genügend lange Berechnung muß einen Zustand mehrfach durchlaufen.

Satz: Für jede Sprache $L \in \mathsf{REG}$ in der Signatur Σ gilt: $\exists n : \forall t \in L : \mathsf{depth}(t) \geq n :$

$$\exists C, D \in \mathsf{Con}(\Sigma), E \in \mathsf{Tree}(\Sigma) : t = C[D[E]] \wedge \forall k \geq 0 : C[D^k[E]] \in L$$

Es gilt sogar $\mathsf{depth}(D[E])) \leq n$.

Beweis: L wird akzeptiert durch einen Automaten $A = (\Sigma, Q, Q_F, \Delta)$. Dann nehmen wir n als $|Q| + 1$.

Übg: Die Sprache

$$\{f(g^k(a), g^k(a)) \mid k \geq 0\}$$

ist nicht regulär.

3.3.4 Endlichkeit

Satz: Für einen Automaten $A = (\Sigma, Q, Q_F, \Delta)$ ist die Frage "ist $\mathsf{Lang}(A)$ endlich" entscheidbar.

Zwei Methoden. Zunächst:

Satz: $\mathsf{Lang}(A)$ ist unendlich $\iff \exists t \in \mathsf{Lang}(A)$ mit $|Q| < \mathsf{depth}(t) \leq 2|Q|$.

Beweis: Pumping-Lemma.

Damit müssen wir, um die Frage nach der Endlichkeit von $\mathsf{Lang}(A)$ zu entscheiden, für alle Bäume t mit $\mathsf{depth}(t) \leq 2|Q|$ prüfen, ob $t \in \mathsf{Lang}(A)$. Das sind nur endlich viele.

Allerdings sehr (exponentiell) viele, deswegen wollen wir ein besseres Verfahren. Wir suchen nach Automatenzuständen, die Schleifen produzieren. Solche gibt es nicht, wenn wir die Zustandsmenge ordnen können:

Wir nehmen an, daß $A = (\Sigma, Q, Q_F, \Delta)$ schlank ist, und bezeichnen mit $<$ den transitiven Abschluß der Relation

$$\text{wenn} f(q_1, \ldots, q_k) \to q \in \Delta, \text{dann} \forall 1 \leq i \leq k : q_i < q$$

Satz. Wenn r ein Lauf von A auf $t \in \mathsf{Tree}(\Sigma)$, dann

$$\forall p, q \in \mathsf{Pos}(t) : p \leq q \Rightarrow r(p) > r(q)$$

Satz. $\mathsf{Lang}(A)$ ist unendlich $\iff \exists q \in Q : q < q$.

Beweis. ($\Rightarrow$): wenn $\mathsf{Lang}(A)$ unendlich ist, dann $\exists t \in \mathsf{Lang}(A) : \mathsf{depth}(t) > |Q|$. Der akzeptierende Lauf von A auf t enthält einen Pfad, auf dem ein Zustand q wenigstens zweimal benutzt wird. Für diesen gilt $q < q$.

($\Leftarrow$): Weil A schlank ist, ist jeder Zustand produktiv und erreichbar. Es gibt also zu jedem $q \in Q$ ein $t \in \mathsf{Lang}(A)$ mit einem akzeptierenden Lauf r, der q benutzt. Bezeichne p diese Position mit $r(p) = q$ für unser q mit $q < q$. Wir gewinnen einen Kontext $C = t[p \mapsto \bullet]$ und einen Teilbaum $s = t[p]$. Wir haben $s \in \mathsf{Lang}(A, q)$.

Aus $q < q$ und daraus, daß alle Zustände von A produktiv sind, folgt andererseits: $\exists D \in \mathsf{Con}(\Sigma)$ mit $\forall s \in \mathsf{Lang}(A, q) : D[s] \in \mathsf{Lang}(A, q)$.

Damit gilt $\forall k \geq 0 : D^k[s] \in \mathsf{Lang}(A, q)$, also auch $C[D^k[s]] \in \mathsf{Lang}(A)$, und das sind unendlich viele Bäume.

3.4 Operationen mit Automaten

3.4.1 Durchschnitt

Satz: Für Automaten A, B gibt es einen Automaten C mit $\mathsf{Lang}(C) = \mathsf{Lang}(A) \cap \mathsf{Lang}(B)$.

Beweis: Kreuzprodukt der Automaten bilden.

Bem: Determinismus bleibt erhalten.

Übg: Vollständigkeit?

3.4.2 Vereinigung

Satz: Für Automaten A, B gibt es einen Automaten C mit $\mathsf{Lang}(C) = \mathsf{Lang}(A) \cup \mathsf{Lang}(B)$.

Beweis: Zustandsmengen disjunkt machen, dann alles vereinigen.

Bem: dabei geht Determinismus verloren (Beispiel!)

anderer Beweis (erhält Determinismus): Kreuzprodukt-Konstruktion und andere akzeptierende Menge (Übung: welche?):

Vorsicht: beide Automaten müssen vollständig sein (das Resultat ist es dann auch).

3.4.3 Komplement

Satz: Für Automaten $A = (\Sigma, Q, Q_F, \Delta)$ gibt es einen Automaten C mit $\mathsf{Lang}(C) = \mathsf{Tree}(\Sigma) \setminus \mathsf{Lang}(A)$.

Beweis: A deterministisch machen, ergibt $B = (\Sigma, Q', Q'_F, \Delta')$. Dann ist $C = (\Sigma, Q', Q \setminus Q'_F, \Delta')$.

Bem: hier läßt sich die Potenzmengenkonstruktion nicht vermeiden. Die Zustandsmenge von C ist im Allgemeinen viel größer als die von A. Beim Implementieren ist das deswegen die kritische Stelle.

3.4.4 Homomorphismen

Ist REG abgeschlossen unter Homomorphismen? Nein.

Bsp: $h : f(x) \mapsto g(x,x)$ und $L = f^*(a) \in \mathsf{REG}$. Dann ist $h(L) \notin \mathsf{REG}$.

Def. Ein Hom $h : \mathsf{Tree}(\Sigma) \to \mathsf{Tree}(\Sigma')$ heißt linear, wenn

$$\forall k : \forall f \in \Sigma_k : h : f(x_1, \ldots, x_k) \mapsto t \in \mathsf{Tree}(\Sigma', \{x_1, \ldots, x_k\}) \quad \text{mit } t \text{ linear}$$

Satz: Für jeden Automaten $A = (\Sigma, Q, Q_F, \Delta)$ und linearen Homomorphismus h gibt es einen Automaten B mit $\mathsf{Lang}(B) = h(\mathsf{Lang}(A))$.

Beispiel: Wenn $f(a,b) \to c \in \Delta$, und $h(f(x,y)) = d(g(y, d(x)))$, dann haben wir $d(g(b, d(a))) \to c$ in B. Wir bringen diese Regel durch neue Zustände in flache Form:

$$d(a) \to q_0, g(b, q_0) \to q_1, d(q_q) \to c.$$

Bem. Auch wenn h nicht linear ist, kann $h(\mathsf{Lang}(A))$ regulär sein.

Einfaches Beispiel: $h : f(x) \mapsto g(x,x)$ und $L = d^* f f a$. Dann ist $h(L) = d^*(g(g(a,a), g(a,a))) \in \mathsf{REG}$. Das liegt hier offenbar daran, daß der nicht-lineare Teil des Homomorphismus nur auf Bäume von beschränkter Tiefe paßt.

Schwereres Beispiel: h und L wie eben. Wir benutzen die reguläre Sprache $M = a \cup g(M, M)$. Wir haben $h(L) \subset M$ und $h(M) = M$, also $h(L \cup M) = M \in \mathsf{REG}$.

Es ist bis heute offen, ob die Frage "ist $h(\mathsf{Lang}(A))$ regulär" entscheidbar ist.

3.4.5 Inverse Homomorphismen

Für inverse Homomorphismen brauchen wir Linearität nicht:

Satz: Für jeden Automaten $A = (\Sigma, Q, Q_F, \Delta)$ und Homomorphismus h gibt es einen Automaten B mit $\mathsf{Lang}(B) = h^{-1}(\mathsf{Lang}(A))$.

3.4.6 Iterierte Homomorphismen

Bei transitiver Hüllenbildung gehen Abschlußeigenscahften verloren.

Satz: Es gibt eine Sprache $L \in \mathsf{REG}$ und einen linearen Homomorphismus h, für den $h^*(L) \notin \mathsf{REG}$.

Beispiel: $L = f(a,b), h : f(x,y) \mapsto f(g(x), g(y))$.

Frage: Wenn $L \in \mathsf{REG}$ und h ein Homomorphismus, ist dann $h^{-*}(L) \in \mathsf{REG}$?