9

Entscheidungsverfahren für ω-Automaten

Automaten sollen den algorithmischen Zugang zu logischen Problemen, insbesondere des Erfüllbarkeitsproblems liefern. In diesem Kapitel widmen wir uns nun den algorithmischen Fragestellungen bzgl. der verschiedenen Typen von Automaten auf unendlichen Wörtern. Wir betrachten im wesentlichen das Leerheitsproblem (ist $L(\mathcal{A}) = \emptyset$?) und das Universalitätsproblem (ist $L(\mathcal{A}) = \Sigma^\omega$?). Ersteres z.B. lässt sich dann zusammen mit einer äquivalenzerhaltenden Reduktion von Formeln in Automaten verwenden, um Erfüllbarkeit zu entscheiden. Dies wurde ja auch bereits bei den Entscheidungsverfahren für WMSO und MSO so verwendet. Andere Probleme wie das Wortproblem für endlich repräsentierte ω-Wörter etc. lassen sich normalerweise leicht auf diese reduzieren.

Die einfachsten Verfahren basieren auf Erreichbarkeitstests, die auf den Transitionsgraphen der Automaten arbeiten. Wir stellen aber noch zwei andere Verfahren vor, welche Universalität für NBA auf anderen Art und Weise lösen.

9.1 Verschiedene Leerheitsprobleme

Wir haben bereits bei der Entscheidbarkeit von MSO in Abschnitt 6.3 angedeutet, wie man Leerheit für NBA entscheiden kann—siehe Satz 6.10. An dieser Stelle betrachten wir zunächst das Leerheitsproblem für die allgemeineren Rabin-Automaten und erhalten dann ein stärkeres Resultat als Satz 6.10 für NBA. Ein wichtiges Hilfsmittel bei diesem und weiteren Verfahren ist die Zerlegung eines gerichteten Graphen in starke Zusammenhangskomponenten.

Definition 9.1. Sei $G = (V, E)$ ein gerichteter Graph. Die *reflexiv-transitive Hülle* E^* der Relation E ist definiert als

$$E^* \quad = \quad \bigcup_{i \in \mathbb{N}} E^i$$

wobei

$$\begin{aligned} E^0 &:= \{(v,v) \mid v \in V\} \\ E^{i+1} &:= \{(v,w) \mid \exists u \in V.(v,u) \in E^i \text{ und } (u,w) \in E\} \end{aligned}$$

Somit besteht E^i aus denjenigen Paaren (v,w), für die es einen Weg der Länge i von v nach w im Graphen G gibt. E^* besteht aus den Paaren (v,w) für die w von v aus überhaupt erreichbar ist.

Definition 9.2. Eine *starke Zusammenhangskomponente* (SCC) ist ein $C \subseteq V$, so dass für alle $v, w \in C$ gilt: $(v,w) \in E^*$. Eine SCC C ist *maximal*, falls jede Obermenge $C' \supseteq C$ keine SCC ist.

Eine SCC C heißt *nichttrivial*, wenn sie mindestens eine Kante enthält, wenn also $C \times C \cap E \neq \emptyset$ gilt.

Beachte, dass es genau die nichttrivialen SCCs sind, in denen es von jedem Zustand zu jedem Zustand einen Pfad gibt, der mindestens die Länge 1 hat.

Es ist klar, dass jeder gerichtete Graph eine eindeutige Zerlegung in maximale, starke Zusammenhangskomponenten besitzt.

Satz 9.3. *Sei $G = (V,E)$ ein gerichteter Graph. Dann lässt sich in Zeit $\mathcal{O}(|E|)$ eine Zerlegung in maximale SCCs berechnen.*

Bevor wir im Folgenden komplexitätstheoretischen Abschätzungen an gewisse Entscheidungsverfahren machen, müssen wir uns noch über die Größe einer Eingabe, bestehend aus einem endlichen Automaten im Klaren sein. Um möglichst präzise Aussagen machen zu können, die auf Satz 9.3 aufbauen, verwenden wir hier die Anzahl der *Transitionen* eines nichtdeterministischen Automaten als Maß für seine Größe. Beachte, dass dies genau der Anzahl der Kanten in seinem Transitionsgraphen entspricht. Wir können außerdem o.B.d.A. davon ausgehen, dass dies eine obere Schranke an die Anzahl der Zustände ist. Im Zweifelsfall muss man einem Automaten noch einen Dummy-Zustand hinzufügen, von dem aus nichts erkannt wird, so dass jeder Zustand mindestens eine ausgehende Kante haben kann. Somit gilt, falls man die Verfahren in der Anzahl der Zustände eines Automaten messen möchte, dass sich die Komplexität in der Anzahl der Transitionen dadurch quadrieren kann.

Satz 9.4. *Das Leerheitsproblem für nichtdeterministische Rabin-Automaten mit n Transitionen und Index k ist in Zeit $\mathcal{O}(nk)$ lösbar.*

Beweis. Sei $\mathcal{A} = (Q, \Sigma, q_0, \delta, \{(G_1,F_1), \ldots, (G_k,F_k)\})$ ein NRA. Dann gilt: $L(\mathcal{A}) \neq \emptyset$ genau dann, wenn es ein $i \in \{1,\ldots,k\}$ gibt, so dass es einen Pfad von q_0 zu einem $q \in G_i$ gibt und einen nichtleeren Pfad von q nach q, auf dem kein Zustand in F_i vorkommt.

Man kann nun folgendermaßen vorgehen. O.B.d.A. kann man davon ausgehen, dass nur noch Zustände vorhanden sind, die vom Initialzustand aus erreichbar sind. Danach geht man alle $i = 1,\ldots,k$ durch und entfernt zunächst

in Zeit $\mathcal{O}(n)$ alle Zustände aus G_i. Dann berechnet man die SCC-Zerlegung des Transitionsgraphen in Zeit $\mathcal{O}(n)$ und prüft für jeden Zustand in F_i, ob dieser in einer nichttrivialen SCC liegt. Letzteres ist in konstanter Zeit möglich. Ist solch ein i und Zustand in F_i gefunden, so ist die Sprache des NRA nichtleer, anderenfalls ist sie leer. Das Verfahren läuft insgesamt in Zeit $\mathcal{O}(nk)$. □

Korollar 9.5. *Das Leerheitsproblem für nichtdeterministische Büchi-Automaten mit n Transitionen ist in Zeit $\mathcal{O}(n)$ lösbar.*

Beweis. Ein NBA mit Endzustandsmenge F ist ein Spezialfall eines NRA mit Endzustandskomponente $\mathcal{F} = \{(F, \emptyset)\}$ und somit vom Index 1. □

Ebenso kann man leicht einen NPA als NRA auffassen, wodurch sich die Schranke aus Satz 9.4 ebenfalls leicht auf NPA überträgt.

Korollar 9.6. *Das Leerheitsproblem für nichtdeterministische Paritätsautomaten mit n Transitionen und Index k ist in Zeit $\mathcal{O}(nk)$ lösbar.*

Beweis. Sei $\mathcal{A} = (Q, \Sigma, q_0, \delta, \Omega)$ ein nichtdeterministischer Paritätsautomat vom Index k. Sei $k' := \lfloor \frac{k}{2} \rfloor$. Es ist leicht zu sehen, dass der Rabinautomat $\mathcal{A}' = (Q, \Sigma, q_0, \delta, \{(G_0, F_0), \ldots, (G_{k'}, F_{k'})\})$ mit

$$\begin{aligned} G_i &:= \{q \in Q \mid \Omega(q) = 2i\} \\ F_i &:= \{q \in Q \mid \Omega(q) > 2i\} \end{aligned}$$

dieselbe Sprache erkennt wie $\mathcal{A}$. Außerdem ist sein Index $\mathcal{O}(k)$. □

Nicht jedoch leicht zu übertragen ist dieses Resultat auf nichtdeterministische Streett-Automaten. Zur Erinnerung: Diese sind syntaktisch genauso wie Rabin-Automaten definiert, in ihrer Akzeptanzbedingung wird jedoch universell statt existentiell über die Paare von Endzustandsmengen quantifiziert. Bevor wir dafür ein Verfahren vorstellen, beweisen wir noch ein kleines technisches Lemma.

Definition 9.7. Sei $\mathcal{A}$ ein Automat mit Zustandsmenge Q und $\rho = q_0 q_1 q_2 \ldots$ ein Lauf des Automaten auf einem beliebigen Wort. Dieser heißt *ultimativ-periodisch*, falls es $i \geq 0$, $n \geq 1$ gibt, so dass für alle $j \geq i$ gilt: $q_j = q_{j+n}$.

Ein ultimativ-periodischer Lauf verfängt sich also in einer Schleife.

Lemma 9.8. *Sei $\mathcal{A}$ ein NSA. Dann gilt $L(\mathcal{A}) \neq \emptyset$ genau dann, wenn es einen akzeptierenden und ultimativ-periodischen Lauf in $\mathcal{A}$ gibt.*

Beweis. "⇐" Trivial.

"⇒" Sei $w \in L(\mathcal{A})$. Dann gibt es einen akzeptierenden Lauf ρ von $\mathcal{A}$ auf w. Angenommen, dieser ist nicht ultimativ-periodisch. Da es nur endlich viele Zustände gibt, enthält dieser ein kleinstes Teilstück der Form $p_1, \ldots, p_m, p_1$, so dass für alle $i = 1, \ldots, k$ gilt: $G_i \cap \{p_1, \ldots, p_m\} = \emptyset$ oder $F_i \cap \{p_1, \ldots, p_m\} \neq \emptyset$. Dann erfüllt auch der Lauf $(p_1 \ldots p_m)^\omega$ die Streett-Bedingung. Schließlich sieht man leicht, dass das Hinzufügen endlicher Präfixe nichts daran ändert. □

Als nächstes betrachten wir einen Algorithmus StAux, welcher als Eingabe einen Graphen $\mathcal{G} = (V, E)$ und eine Menge $\mathcal{S} = \{(G_1, F_1), \ldots, (G_k, F_k)\}$ von Paaren von Knotenmenge erhält und entscheidet, ob es in $\mathcal{G}$ eine nichttriviale starke Zusammenhangskomponente C gibt, so dass für alle $i = 1, \ldots, k$ folgendes gilt.

$$G_i \cap C \neq \emptyset \implies F_i \cap C \neq \emptyset$$

Dieser arbeitet folgendermaßen. Zunächst wird der Graph in SCCs zerlegt und man betrachtet nur die nichttrivialen davon. Gibt es eine, die nichtdisjunkt mit allen F_i ist, so sind wir fertig. Anderenfalls führt man für jede SCC C folgendes durch. Man geht alle Paare (G_i, F_i) der Reihe nach durch. Ist der Schnitt aus C und F_i leer, so muss erstens auch der Schnitt aus C und G_i leer sein und C "gut" bzgl. der noch verbleibenden Paare sein. Dies überprüft man, indem man G_i aus dem Graphen entfernt und den Algorithmus rekursiv mit den noch verbleibenden Paaren auf den Teilgraphen anwendet. In Pseudo-Code liest sich das folgendermaßen.

```
StAux(G = (V,E), {(G_1,F_1),...,(G_k,F_k)})
    seien C_1,...,C_m die nichttrivialen SCCs von G
    if ∃i ∈ {1,...,m}.∀j = 1,...,k. C_i ∩ F_j ≠ ∅ then return true
    for i = 1,...,m do
        j := min{h | C_i ∩ F_h = ∅}
        G' := (V \ G_j, E ∩ (V \ G_j) × (V \ G_j))
        S' := {(G_{j+1},F_{j+1}),...,(G_k,F_k)}
        if StAux(G',S') = true then return true
    done
    return false
```

Lemma 9.9. *Algorithmus* StAux *liefert true auf Eingabe $\mathcal{G} = (V, E)$ und $\mathcal{S}$ genau dann, wenn es eine nichttriviale SCC C gibt, so dass für alle $(G, F) \in S$ gilt: $G \cap C = \emptyset$ oder $F \cap C \neq \emptyset$.*

Beweis. Übung.

Lemma 9.10. *Algorithmus* StAux *terminiert auf Eingabe $\mathcal{G} = (V, E)$ und $\mathcal{S} = \{(G_1, F_1), \ldots, (G_k, F_k)\}$ in $\mathcal{O}(nk + k^2)$ vielen Schritten, wobei $n = |E|$.*

Beweis. Nach Satz 9.3 ist die SCC-Zerlegung in $\mathcal{O}(n)$ möglich. O.B.d.A. gehen wir wieder davon aus, dass es höchstens n Knoten im Graphen gibt. Sei $T(n, k)$ die worst-case Laufzeit des Algorithmus. Offensichtlich gilt $T(n, 0) = \mathcal{O}(1)$. Desweiteren gilt für $k > 0$:

$$T(n,k) \;=\; \mathcal{O}(n) + \sum_{i=1}^{m} \mathcal{O}(k+n) + T(|C_i|, k-1) \;\leq\; \mathcal{O}(n+k) + T(n, k-1)$$

Die Ungleichung gilt, weil wir bereits annehmen können, dass $T(n, k)$ mindestens linear im ersten Argument ist. Man vergewissere sich, dass $T(n, k) = \mathcal{O}(nk + k^2)$ eine Lösung dieser Rekurrenz ist. □

Satz 9.11. *Das Leerheitsproblem für nichtdeterministische Streett-Automaten mit n Transitionen und Index k ist in Zeit $\mathcal{O}(nk^2)$ lösbar.*

Beweis. Sei $\mathcal{A} = (Q, \Sigma, q_0, \delta, \{(G_1, F_1), \ldots, (G_k, F_k)\})$ ein NSA. Aufgrund von Lemma 9.8 reicht es aus, nach ultimativ-periodischen Läufen zu suchen, um auf Nichtleerheit zu testen. Deswegen gilt $L(\mathcal{A}) \neq \emptyset$ genau dann, wenn es eine SCC $C \subseteq Q$ gibt, die von q_0 aus erreichbar ist, so dass der NSA, den man aus $\mathcal{A}$ gewinnt, indem alle Zustände, die nicht zu C gehören, gelöscht werden, eine nichtleere Sprache akzeptiert. Die Wahl des Anfangszustands ist dabei irrelevant, da C eine SCC ist und somit entweder von allen oder von keinem Zustand aus ein Wort akzeptiert wird. Somit gilt, dass $L(\mathcal{A})$ nichtleer ist genau dann, wenn Algorithmus StAux den Transitionsgraphen von $\mathcal{A}$, eingeschränkt auf die Zustände, die vom Anfangszustand aus erreichbar sind, akzeptiert. Die Behauptung folgt dann also aus Lemma 9.10. □

Es ist wichtig, dass Algorithmus StAux nicht direkt auf den Transitionsgraphen des NSA angewandt wird, sondern nur auf den vom Anfangszustand aus erreichbaren Teil. Beachte, dass der Algorithmus selbst sich nicht um Erreichbarkeit von irgendwelchen Zuständen aus kümmert. So könnte es eine SCC geben, die dazu führt, dass StAux die Ausgabe *true* liefert, die aber selbst vom Anfangszustand aus nicht erreichbar ist. Somit muss die Sprache des NSA auch nicht unbedingt nichtleer sein. Andererseits kann man nicht im Algorithmus StAux selbst verlangen, dass die Eingabe aus einem zusammenhängenden Graphen besteht, da diese Eigenschaft vor den rekursiven Aufrufen verletzt werden kann.

9.2 Universalitätsproblem für Büchi-Automaten

Wenden wir uns nun dem komplementären Problem zu—dem Universalitätsproblem. Beachte, dass dies auf Seite der Logik dem Allgemeingültigkeitsproblem entspricht, sowie Leerheit (oder besser gesagt Nichtleerheit) dem Erfüllbarkeitsproblem entspricht.

9.2.1 Universalität durch Komplementierung

Aus den obigen Resultaten können wir gleich ein Verfahren für das Universalitätsproblem für Büchi-Automaten konstruieren.

Korollar 9.12. *Das Universalitätsproblem für nichtdeterministische Büchi-Automaten mit n Zuständen ist in Zeit $2^{\mathcal{O}(n \log n)}$ lösbar.*

Beweis. Sei $\mathcal{A}$ ein NBA mit n Zuständen. Laut Korollar 8.14 existiert ein deterministischer Rabin-Automat $\mathcal{A}'$ mit höchstens $2^{\mathcal{O}(n \log n)}$ vielen Zuständen und Index höchstens $2n$, so dass $L(\mathcal{A}') = L(\mathcal{A})$ gilt. Nach Satz 7.7 lässt sich

dieser ohne Blow-Up in Zustandszahl und Index zu einem DSA $\overline{\mathcal{A}}$ komplementieren; also gilt $L(\overline{\mathcal{A}}) = \Sigma^\omega \setminus L(\mathcal{A})$.

Die Behauptung folgt dann sofort aus Satz 9.11, da $L(\mathcal{A}) = \Sigma^\omega$ genau dann, wenn $L(\overline{\mathcal{A}}) = \emptyset$ gilt. Beachte, dass der eventuelle quadratische Blow-Up im Vergleich zwischen Anzahl der Zustände und Anzahl der Transitionen von der Konstante im Exponenten verschluckt wird. □

9.2.2 Universalität mit dem Satz von Ramsey

Es gibt noch ein alternatives Verfahren für das Universalitätsproblem für Büchiautomaten, welches auf der in Abschnitt 6.2 definierten Äquivalenzrelation für Büchi-Automaten und dem Satz von Ramsey beruht.

Sei für den Rest dieses Abschnitts ein Büchiautomat $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ fixiert. Zur Erinnerung: Seien $u, v \in \Sigma^*$. Es gilt

$$u \sim v \iff \forall q.\forall q'.(q \xrightarrow{u} q' \Leftrightarrow q \xrightarrow{v} q') \ \wedge \ (q \xrightarrow{u}_{\mathsf{fin}} q' \Leftrightarrow q \xrightarrow{v}_{\mathsf{fin}} q')$$

Eine Äquivalenzklasse L dieser Relation ist durch zwei Mengen von Zustandspaaren gekennzeichnet: Die Menge V_L aller Paare (q, q'), so dass $q \xrightarrow{w} q'$ für alle $w \in L$, sowie die Menge V_L^{fin} aller Paare (q, q'), so dass $q \xrightarrow{w}_{\mathsf{fin}} q'$ für alle $w \in L$.

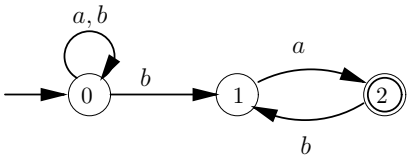
Wie üblich bezeichnet man mit $[w]$ die Äquivalenzklasse des Wortes $w \in \Sigma^*$. Für Mengen von Paaren (Relationen) $R, S \subseteq Q \times Q$ definiert man wie üblich $R; S = \{(q, q') \mid \exists \check{q}.(q, \check{q}) \in R, (\check{q}, q') \in S\}$.

Lemma 9.13. *a) Es gilt* $V_{[\varepsilon]} = \{(q, q) \mid q \in Q\}$ *und* $V_{[\varepsilon]}^{\mathsf{fin}} = \{(q, q) \mid q \in F\}$.
b) Für $a \in \Sigma$ *gilt* $V_{[a]} = \{(q, q') \mid q' \in \delta(a, q)\}$ *und* $V_{[a]}^{\mathsf{fin}} = \{(q, q') \mid q' \in \delta(a, q) \wedge (q \in F \vee q' \in F)\}$.
c) Für $u, v \in \Sigma^*$ *gilt:* $V_{[uv]} = V_u; V_v$ *und* $V_{[uv]}^{\mathsf{fin}} = V_{[u]}^{\mathsf{fin}}; V_{[v]} \cup V_{[u]}; V_{[v]}^{\mathsf{fin}}$.

Beweis. Übung.

Man kann sich nun mithilfe dieses Lemmas eine Übersicht über die endlich vielen Äquivalenzklassen verschaffen: Ausgehend von $[a]$ für $a \in \Sigma$ konstruiert man durch sukzessive Anwendung des Lemmas Repräsentationen für weitere Wörter, bis keine neuen Klassen mehr entstehen.

Beispiel 9.14. Wir betrachten den folgenden Automaten.

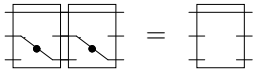


Wir repräsentieren Äquivalenzklassen grafisch als "Schaltboxen", z.B.:

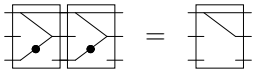
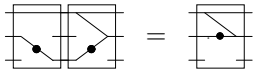
$[\varepsilon]$ =  $[a]$ =  $[b]$ = 

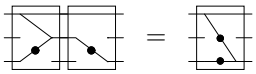
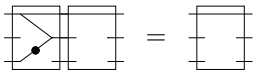
Die Zustände sind als Ein- und Ausgabedrähte repräsentiert. Hier stellen die Drähte oben den Zustand 0, die in der Mitte den Zustand 1 und die unten den Zustand 2 dar. Ein Zustand q auf der Eingabeseite (links) wird mit einem Zustand q' auf der Ausgabeseite (rechts) verbunden, wenn $q \xrightarrow{u} q'$. Gilt zusätzlich $q \xrightarrow{u}_{\mathsf{fin}} q'$, so wird die Verbindung mit einem Kreis gekennzeichnet.

Hat man Äquivalenzklassen $[u]$ und $[v]$ in dieser Form gegeben, so kann die Äquivalenzklasse $[uv]$ durch Hintereinanderschalten abgelesen werden:

$[aa]$ = 

Ebenso erhalten wir folgendes.

$[bb]$ =  $[ab]$ = 

$[ba]$ =  $[baa]$ = 

Aus der letzten Gleichung ergibt sich $[baa] = [aa]$. Indem wir so weiterverfahren, erhalten wir folgende Multiplikationstabelle:

	a	b	ab	ba	bb	aa	bba	aba	bab	$bbab$	$abab$
a	aa	ab	bb	aba	bb	aa	bba	bba	$abab$	$bbab$	$bbab$
b	ba	bb	bab	bba	bb	aa	bba	ba	$bbab$	$bbab$	bab
ab	aba	bb	$abab$	bba	bb	aa	bba	aba	$bbab$	$bbab$	$abab$
ba	aa	bab	bb	ba	bb	aa	bba	bba	bab	$bbab$	$bbab$
bb	bba	bb	$bbab$	bba	bb	aa	bba	bba	$bbab$	$bbab$	$bbab$
aa	aa	bb	bb	bba	bb	aa	bba	bba	$bbab$	$bbab$	$bbab$
bba	aa	$bbab$	bb	bba	bb	aa	bba	bba	$bbab$	$bbab$	$bbab$
aba	aa	$abab$	bb	aba	bb	aa	bba	bba	$abab$	$bbab$	$bbab$
bab	ba	bb	bab	bba	bb	aa	bba	ba	$bbab$	$bbab$	bab
$bbab$	bba	bb	$bbab$	bba	bb	aa	bba	bba	$bbab$	$bbab$	$bbab$
$abab$	aba	bb	$abab$	bba	bb	aa	bba	aba	$bbab$	$bbab$	$abab$

Viele der Gleichungen sind rein algebraische Konsequenzen von anderen. So kann man etwa $[ba][abab] = [ba][ab][ab] = [bb][ab] = [bbab]$ folgern.

Mithilfe des folgenden Satzes lässt sich dann die Universalität unmittelbar entscheiden.

Satz 9.15. *Es gilt $L(\mathcal{A}) = \Sigma^\omega$ genau dann, wenn für alle Klassen $[u], [v]$ mit $[uv] = [u]$ und $[vv] = [v]$ mit $v \neq \varepsilon$ gilt: es gibt einen Zustand q mit $(q_0, q) \in V_{[u]}$ und $(q, q) \in V^{\mathsf{fin}}_{[v]}$.*

Beweis. "$\Rightarrow$" Nehmen wir zunächst an, dass $L(\mathcal{A}) = \Sigma^\omega$. Seien zwei Klassen $[u], [v]$ mit $[uv] = [u]$ und $[vv] = [v]$ mit $v \neq \varepsilon$ vorgelegt. Betrachte einen akzeptierenden Lauf von $\mathcal{A}$ auf uv^ω. Wir finden einen Zustand q und Zahlen $i_0 \geq 0, i_1 > 0$ etc., so dass $q_0 \xrightarrow{uv^{i_0}} q$ und $q \xrightarrow{v^{i_1}}_{\mathsf{fin}} q$. Wegen $[uv] = [u]$ und $[vv] = [v]$ bedeutet das aber, dass $(q_0, q) \in V_{[u]}$ und $(q, q) \in V^{\mathsf{fin}}_{[v]}$.

"$\Leftarrow$" Sei umgekehrt die Bedingung an die Klassen $[u], [v]$ erfüllt und $w \in \Sigma^\omega$ beliebig. Wie im Beweis von Lemma 6.7 finden wir mithilfe des Satzes von Ramsey Äquivalenzklassen $[u], [v]$, sowie eine Indexfolge $i_1 < i_2 < i_3 < \ldots$, so dass $[w_{0\ldots i_0}] = [u]$ und $[w_{i_j+1,\ldots i_{j'}}] = [v]$ für $j' > j$. Daraus folgt unmittelbar $[vv] = [v]$ und, indem wir u ggf. durch uv ersetzen, auch $[uv] = [u]$. Also gibt es nach Annahme einen Zustand q mit $(q_0, q) \in V_{[u]}$ und $(q, q) \in V^{\mathsf{fin}}_{[v]}$. Dies liefert einen akzeptierenden Lauf von $\mathcal{A}$ auf w. □

In der Praxis prüft man für jede idempotente Äquivalenzklasse, also $[v]$ mit $[vv] = [v]$, ob es eine Klasse u gibt mit $[uv] = [u]$ und $(q_0, q) \in V_{[u]} \Rightarrow (q, q) \notin V^{\mathsf{fin}}_{[v]}$. Findet man solche Klassen $[u], [v]$, so akzeptiert $\mathcal{A}$ nicht alle Wörter, nämlich insbesondere nicht uv^ω. Findet man keine solchen Beispiele, dann akzeptiert $\mathcal{A}$ jedes beliebige Wort.

Der Automat aus Beispiel 9.14 akzeptiert *nicht* alle Wörter: Die *idempotenten* Äquivalenzklassen, also v mit $[v][v] = [v]$, sind $[ba]$, $[bb]$, $[aa]$, $[bba]$, $[bbab]$, $[abab]$. Alle außer $[abab]$ verletzen die Bedingung aus dem Satz, insbesondere ist sie für $u = [b]$ und $v = [aa]$ verletzt und in der Tat wird $b(aa)^\omega$ nicht akzeptiert.

Man hätte hier die Konstruktion der Multiplikationstabelle gleich nach der Entdeckung von $[aa]$ abbrechen können. Akzeptiert der Automat jedoch alle Wörter, so muss die vollständige Tabelle generiert werden, da man ja sonst nicht weiß, ob nicht doch noch Klassen $[u]$ und $[v]$ wie beschrieben, gefunden werden.

9.2.3 Reduktion von Subsumption auf Universalität

Schließlich wenden wir uns der etwas allgemeineren Frage zu, ob $L(\mathcal{A}) \subseteq L(\mathcal{A}')$ für vorgelegte Büchi-Automaten $\mathcal{A}$ und $\mathcal{A}'$ gilt. Dies lässt sich recht einfach auf das bereits behandelte Universalitätsproblem reduzieren:

Satz 9.16. *Seien $\mathcal{A}$ und $\mathcal{A}'$ NBAs über demselben Alphabet Σ. Dann kann in linearer Zeit ein NBA $\mathcal{A}''$ über einem Alphabet Δ konstruiert werden, so dass gilt: $L(\mathcal{A}'') = \Delta^\omega$ genau dann, wenn $L(\mathcal{A}) \subseteq L(\mathcal{A}')$.*

Beweis. Seien $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ und $\mathcal{A}' = (Q', \Sigma, q_0', \delta', F')$. Es ist $L(\mathcal{A}) \subseteq L(\mathcal{A}')$ genau dann, wenn für jedes Wort $w \in \Sigma^\omega$ und jeden akzeptierenden Lauf $r \in Q^\omega$ von $\mathcal{A}$ auf w ein akzeptierender Lauf von $\mathcal{A}'$ auf w existiert.

Die universelle Quantifizierung über akzeptierende Läufe von $\mathcal{A}'$ wird nun über die Alphabeterweiterung zu $\Delta = \Sigma \times Q$ der universellen Quantifizierung über Wörter zugeschlagen.

Genauer gesagt sind $Q'', \delta'', q_0'', F''$ so zu wählen, dass ein Wort $(w, r) \in (\Sigma \times Q)^\omega$ genau dann akzeptiert wird, wenn entweder r kein akzeptierender Lauf von $\mathcal{A}$ auf w ist, oder aber $w \in L(\mathcal{A})$ ist.

Hierzu muss zum einen ein Lauf von $\mathcal{A}'$ auf w zu raten und zum anderen zu prüfen, ob der gegebene Lauf r der Übergangstafel δ folgt. Ist dies irgendwann nicht der Fall, so kann sofort akzeptiert werden. Ansonsten wird geraten, ob entweder r nicht akzeptierend ist (durch Raten eines Zeitpunktes ab dem keine Endzustände mehr kommen), oder aber der geratene Lauf von $\mathcal{A}'$ akzeptierend ist. Die Details verbleiben als Übung. □

Um also das Subsumptionsproblem $L(\mathcal{A}) \subseteq L(\mathcal{A}')$ zu lösen, konstruiert man diesen Automaten $\mathcal{A}''$ und wendet dann eines der Entscheidungsverfahren für das Universalitätsproblem an.

9.3 Size-change Termination

In diesem Abschnitt betrachten wir eine Anwendung von Büchi-Automaten auf die Frage, ob ein gegebenes rekursives Programm terminiert.

9.3.1 Rekursive Programme

Gegeben sei eine Menge F von Funktionssymbolen verschiedener Stelligkeit. Ein (rekursives) *Programm* enthält für jedes Funktionssymbol $f \in F$ der Stelligkeit n genau eine Gleichung der Form

$$f(x_1, \ldots, x_n) = f_1(\mathbf{t}_1), \ldots, f_l(\mathbf{t}_l)$$

Hierbei sind $f_1, \ldots f_l$ beliebige Funktionssymbole in F (einschließlich f selbst). Die Terme $\mathbf{t}_i$ sind aufgebaut aus den Variablen $x_1, \ldots, x_n$ und der Vorgängerfunktion $x - 1$. Solche Programme abstrahieren von konkreten, rekursiven Programmen im Hinblick auf ihr Verhalten bzgl. Termination.

Beispiel 9.17. Ein funktionales, rekursives Programm zur Multiplikation zwei natürlicher Zahlen kann z.B. folgendermaßen modelliert werden. Man nimmt ein zweistelliges Symbol p und ein dreistelliges Symbol m.

$$\begin{aligned} \mathsf{p}(x, y) &= \mathsf{p}(x, y - 1) \\ \mathsf{m}(x, y, u) &= \mathsf{p}(x, u), \mathsf{m}(x, y - 1, u) \end{aligned}$$

Es sollte klar sein, dass dieses rekursive Programm keine Multiplikation selbst berechnet—wir haben den Programmen ja auch noch keine formale Semantik gegeben. Aber man kann sich vorstellen, dass damit ein Programm abstrahiert wird, welches für beliebige, natürliche Zahlen x, y, u rekursiv den Wert $x \cdot y + u$ berechnet. Insbesondere sollte das konkrete (hier nicht näher ausgeführte) Programm zur Multiplikation immer terminieren, falls ein Aufruf von $m(x, y, u)$ für jede Setzung der Variablen x, y, u terminiert. Was Termination genau bedeuten soll, wird weiter unten ausgeführt.

Beispiel 9.18. Die Ackermann-Funktion ist leicht mit einem dreistelligen Funktionssymbol a zu modellieren.

$$\mathsf{a}(x,y,u) \;=\; \mathsf{a}(x-1,u,u), \mathsf{a}(x,y-1,u)$$

Beispiel 9.19. Wir bringen noch zwei künstliche Beispiele, die nicht unbedingt aus der Abstraktion eines konkreten Programms entstammen.

$$\begin{aligned}
\mathsf{f}(x,y,z) &= \mathsf{g}(x,x,y)\\
\mathsf{g}(x,y,z) &= \mathsf{h}(x,y-1,z-1)\\
\mathsf{h}(x,y,z) &= \mathsf{i}(x,y-1,z-1)\\
\mathsf{i}(x,y,z) &= \mathsf{k}(x,y-1,z-1)\\
\mathsf{k}(x,y,z) &= \mathsf{f}(x,y-1,z-1)
\end{aligned}$$

und

$$\mathsf{t}(x,y,z,w) \;=\; \mathsf{t}(x,x,z,w-1), \mathsf{t}(x-1,z,w-1,y-1), \mathsf{t}(z,x-1,y,w-1)$$

Die Idee ist, dass durch solch ein Programm partielle Funktionen $\mathbb{N}^k \to \{0\}$ definiert werden. Per Definition ist $f(x_1,\ldots,x_n) = 0$, falls $x_i = 0$ für ein i (Striktheit). Für andere Werte ergibt sich der Funktionswert aus der definierenden Gleichung

$$f(\mathbf{x}) \;=\; u_1(\mathbf{x}_1),\ldots,u_k(\mathbf{x}_k)\ .$$

Sind *alle* $u_i(\mathbf{x}_i)$ gleich 0 und damit insbesondere definiert, so ist auch $f(\mathbf{x})$ definiert und damit gleich 0. Ist auch nur einer der Terme $u_q(\mathbf{x}_1),\ldots,u_k(\mathbf{x}_k)$ undefiniert, so ist auch die linke Seite undefiniert.

Die Frage, die uns hier besonders interessiert, ist: Sind alle Instanzen $f(u_1,\ldots,u_n)$ in einem gegebenen Programm definiert? Man kann dies wie oben angedeutet als Terminationsproblem ansehen. Ein Funktionsaufruf terminiert, wenn eines der Argumente 0 ist, ansonsten terminiert es nur, wenn alle rekursiven Aufrufe terminieren. Die Frage ist also, ob eine rekursive Funktion für alle Eingaben terminiert.

Bei den Beispielsfunktionen $\mathsf{p},\mathsf{m},\mathsf{a}$ oben sind alle Instanzen definiert. Bei f haben wir jedoch

$$\begin{aligned}
&\mathsf{f}(10,10,10) = \mathsf{g}(10,10,10) = \mathsf{h}(10,9,9) = \mathsf{i}(10,8,8) = \mathsf{k}(10,7,7) =\\
&\mathsf{f}(10,6,6) \;\; = \mathsf{g}(10,10,6) \;\; = \mathsf{h}(10,9,5) = \mathsf{i}(10,8,4) = \mathsf{k}(10,7,3) =\\
&\mathsf{f}(10,6,2) \;\; = \mathsf{g}(10,10,6) \;\; = \ldots
\end{aligned}$$

Man sieht, dass $\mathsf{f}(10,10,10)$ und größere Instanzen nicht definiert sind; bestimmt kleinere Instanzen dagegen sind es schon.

Um das zweite künstliche Beispiel von oben zu analysieren, schreiben wir $f(\mathbf{t}) \rightsquigarrow g(\mathbf{u})$ zum Zeichen, dass der Aufruf $f(\mathbf{t})$ den Aufruf $g(\mathbf{u})$ unmittelbar nach sich zieht. Dann gilt

$$\mathsf{t}(3,10,11,4) \leadsto \mathsf{t}(2,11,3,9) \leadsto \mathsf{t}(3,1,11,8) \leadsto \mathsf{t}(3,3,11,7) \leadsto \\ \mathsf{t}(11,2,3,6) \leadsto \mathsf{t}(11,11,3,5) \leadsto \mathsf{t}(3,10,11,4) \leadsto \ldots$$

und damit ist $\mathsf{t}(3,10,11,4)$ nicht definiert. Hingegen ist $\mathsf{t}(10,10,10,10)$ definiert, wie man mithilfe eines Rechners feststellt. Es empfiehlt sich, dies über dynamische Programmierung zu lösen.

Es sollte klar sein, dass in vielen Fällen ein reales funktionales Programm P, dessen Terminationsverhalten nicht vom Wertverlauf abhängt, zu einem Programm P' im obigen Sinne abstrahiert werden kann in einer solchen Weise, dass P' genau dann für alle Eingaben terminiert, wenn P es tut. Natürlich ist das nicht immer möglich, denn für unsere Programme ist die Frage nach der Termination für alle Eingaben entscheidbar, wie wir gleich sehen werden.

Vorher bemerken wir noch, dass solch eine automatische Terminationsanalyse nicht nur für Programme, sondern gerade auch für Beweise sehr nützlich ist. Induktive Beweise stellt man sich gern rekursiv vor; man verwendet die zu zeigende Aussage einfach und geht dabei davon aus, dass solche Verwendungen bezüglich irgendeines Maßes kleiner sind, als die gerade aktuelle. Ein solcher "rekursiver" Beweis ist natürlich nur dann gültig, wenn jeder "Aufruf" tatsächlich terminiert; somit ist ein rechnergestützter Beweisprüfer, der solche Beweisstrategien anbietet, auf eine Terminationsanalyse angewiesen.

9.3.2 Termination als Sprachinklusion

Für den Rest dieses Kapitels sei ein Programm P mit Funktionssymbolen F vorgegeben. Es sei n die maximale Zahl von Funktionsaufrufen in der rechten Seite einer Gleichung.

Wir betrachten das Alphabet $\Sigma = F \times \{1,\ldots,n\}$ und definieren die ω-Sprache $L_{\mathsf{call}} \subseteq \Sigma^\omega$ als die Menge aller unendlichen Aufrufsequenzen: ein Wort $(f_0,d_0)(f_1,d_1)(f_2,d_2)\ldots$ ist in L_{call} genau dann, wenn ein Aufruf der Form $f_{i+1}(\mathbf{t})$ in der rechten Seite der definierenden Gleichung von f_i an d_i-ter Stelle vorkommt. Wir lassen Klammern und Kommas weg und schreiben so ein Wort als $f_0 d_0 f_1 d_1 \ldots$.

Im Beispiel mit $F = \{\mathsf{f},\mathsf{g},\mathsf{h},\mathsf{i},\mathsf{k}\}$ enthält L_{call} fünf Wörter, nämlich

$$\begin{aligned} w_{\mathsf{f}} &= (\mathsf{f1g1h1i1k1})^\omega \\ w_{\mathsf{k}} &= \mathsf{k1}w_{\mathsf{f}} \\ w_{\mathsf{i}} &= \mathsf{i1}w_{\mathsf{k}} \\ w_{\mathsf{h}} &= \mathsf{h1}w_{\mathsf{i}} \\ w_{\mathsf{g}} &= \mathsf{g1}w_{\mathsf{h}} \end{aligned}$$

Im Beispiel mit $F = \{\mathsf{t}\}$ ist $L_{\mathsf{call}} = (\mathsf{t1} \cup \mathsf{t2} \cup \mathsf{t3})^\omega$ Im Beispiel mit $F = \{\mathsf{m},\mathsf{p}\}$ schließlich wäre $L_{\mathsf{call}} = ((\mathsf{m1} \cup \mathsf{m2})\mathsf{p1})^\omega \cup (\mathsf{p1}(\mathsf{m1} \cup \mathsf{m2}))^\omega$.

Eine Aufrufsequenz terminiert, wenn es eine Variable gibt, die immer wieder dekrementiert wird. Denn dann kann ja ein noch so hoher Startwert irgendwann zu Null reduziert werden und damit die Termination herbeiführen.

Natürlich muss man verlangen, dass jede Aufrufsequenz in diesem Sinne terminiert und die dies bezeugende Variable kann immer jeweils eine andere sein. Es sei L_{term} die Sprache der in diesem Sinne terminierenden Aufrufsequenzen.

Es gilt im Beispiel "Multiplikation", dass $(\mathsf{m2})^\omega \in L_{\mathsf{term}}$, da hier die Variable y immer wieder dekrementiert wird. Im zweiten künstlichen Beispiel gilt $(\mathsf{t1})^\omega \in L_{\mathsf{term}}$, ja sogar $(\mathsf{t1} \cup \mathsf{t3})^\omega \subseteq L_{\mathsf{term}}$. Auf der anderen Seite ist $(\mathsf{t2t3t1t3t1t3})^\omega \notin L_{\mathsf{term}}$.

Das gesamte Programm P wird für alle Aufrufe terminieren genau dann, wenn $L_{\mathsf{call}} \subseteq L_{\mathsf{term}}$ gilt. Dies ist also z.B. beim zweiten künstlichen Beispiel nicht der Fall.

9.3.3 Terminationsanalyse mit Büchi-Automaten

Die Sprache L_{term} ist Büchi-erkennbar, also ω-regulär. Sei m die maximale Stelligkeit eines Funktionssymbols in F. Der Einfachheit halber nehmen wir an, dass alle Funktionssymbole die Stelligkeit m haben. Dies lässt sich natürlich leicht durch Hinzunahme von Variablen, die in rekursive Aufrufe einfach unverändert weitergereicht werden, erreichen.

Wir definieren einen NBA $\mathcal{A} = (Q, \Sigma, I, \delta, F)$ mit mehreren Anfangszuständen wie folgt.

- Zustandsmenge ist $Q = \{1, \ldots, m\} \times \{0, 1\}$,
- Initialzustände sind alle Zustände: $I = Q$.
- Die Übergangsfunktion δ ergibt sich wie folgt: Sei $g(y_1, \ldots, y_m)$ der d-te Aufruf in der definierenden Gleichung für $f(x_1, \ldots, x_m)$. Ist $y_j = x_i$, so nehmen wir $(j, 0)$ in $\delta((i, \_), (f, d))$ auf. Ist $y_j = x_i - 1$, so nehmen wir $(j, 1)$ in $\delta((i, \_), (f, d))$ auf.
- Endzustände sind alle Zustände der Form $(i, 1)$.

$\mathcal{A}$ errät also die Variable, die immer wieder in dem als Wort vorliegenden Aufruf dekrementiert wird. Diese merkt er sich in der Zustandsmenge. Außerdem wird in der Zustandsmenge signalisiert, wenn diese dekrementiert wird.

Im Beispiel der Abstraktion der Multiplikationsfunktion haben wir konkret die folgende Transitionsfunktion.

$$\begin{aligned}
\delta((1, _), (\mathsf{p}, 1)) &= \{(1, 0)\} \\
\delta((2, _), (\mathsf{p}, 1)) &= \{(2, 1)\} \\
\delta((1, _), (\mathsf{m}, 1)) &= \{(1, 0)\} \\
\delta((2, _), (\mathsf{m}, 1)) &= \emptyset \\
\delta((3, _), (\mathsf{m}, 1)) &= \{(2, 0)\} \\
\delta((1, _), (\mathsf{m}, 2)) &= \{(1, 0)\} \\
\delta((2, _), (\mathsf{m}, 2)) &= \{(2, 1)\} \\
\delta((3, _), (\mathsf{m}, 2)) &= \{(3, 1)\}
\end{aligned}$$

Im ersten künstlichen Beispiel haben wir einen nichtdeterministischen Übergang:

$$\delta((1, \_), (\mathsf{f}, 1)) = \{(1, 0), (2, 0)\}$$

Die vollständige Übergangsfunktion geben wir hier nicht an.

Es sollte klar sein, dass $\mathcal{A}$ genau die terminierenden Aufrufe erkennt, da er akzeptiert, wenn es eine Variable gibt, die in einer vorliegenden Aufrufsequenz immer wieder dekrementiert wird. Somit kann die Belegung dieser Variablen mit einer natürlichen Zahl noch so hoch sein; sie wird ultimativ irgendwann zu Null. Also gilt $L_{\mathsf{term}} = L(\mathcal{A})$.

Satz 9.20. *Sei P ein rekursives Programm, und L_{call} und L_{term} die dazugehörigen Sprachen aller und aller terminierender Aufrufe. Es gilt $L_{\mathsf{call}} \subseteq L_{\mathsf{term}}$ genau dann, wenn $f(\mathbf{x})$ für alle Funktionen f in P und alle Belegungen für $\mathbf{x}$ definiert ist.*

Beweis. Übung.

Die Termination solch eines Programms kann daher mit Satz 9.16 entschieden werden. Alternativ kann man auch gleich einen Büchiautomaten $\mathcal{A}'$ konstruieren, der die Sprache $\overline{L_{\mathsf{call}}} \cup L_{\mathsf{term}}$ erkennt. Das geht auch ohne Komplementierung von Büchiautomaten, da ein Büchiautomat für $\overline{L_{\mathsf{call}}}$ direkt angegeben werden kann. Dieser ist deterministisch und hat nur Endzustände, kann jedoch steckenbleiben. Man testet den so entstandenen Büchi-Automaten auf Universalität.