Institut für Formale Methoden der Informatik

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Studienarbeit Nr.

Gewisse Eigenschaften deterministischer Automaten und ihre Komplexität

Moritz Rathgeber

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. V. Diekert
Betreuer/in:	Dr. M. Kufleitner
Beginn am:	2013-11-18
Beendet am:	2014-05-20
CR-Nummer:	

Kurzfassung

In dieser Arbeit wird untersucht, wie sich die Ergebnisse von Cho und Huynh [CH91] zur Komplexität der Probleme „gegeben ein deterministischer endlicher Automat: ist die von diesem Automat akzeptierte Sprache sternfrei bzw. *piecewise testable* bzw. *dot-depth-one*“ auf andere Klassen von Sprachen übertragen lassen. Es kann gezeigt werden, dass alle Klassen, die durch Omega-Gleichungen charakterisierbar sind, in PSPACE entscheidbar und NL-schwer sind. Wir geben Verfahren an, dass zu jeder Gleichung, die eine gewisse Bedingung erfüllt, ein Verbotsmuster erzeugt, das in NL entscheidbar ist, und zeigen, dass das Schnittproblem bereits für die Klasse der deterministische endliche Automaten, die *locally testable* Sprachen akzeptieren, PSPACE-vollständig ist, wodurch der PSPACE-Vollständigkeitsbeweis für die Klasse der sternfreien Spachen vereinfacht werden kann.

Inhaltsverzeichnis

1 Einleitung

Zu jeder Unterklasse der regulären Sprachen gibt es das Entscheidungsproblem, ob ein deterministischer endlicher Automat eine Sprache aus dieser Klasse akzeptiert. In [Ste85b] wurde gezeigt, dass dieses Entscheidungsproblem für die Klasse der sternfreien Sprachen in PSPACE liegt und Co-NP-schwer ist und für die Klassen *dot-depth one* (definiert in [CB71]) und *piecewise testable* (definiert in [Sim75]) in P liegt. In [CH91] wurde die Charakterisierung der Platzkomplexität zu PSPACE-vollständig und NL-vollständig verbessert. Im Folgenden verwenden wir eine Sprachklasse auch als Kurzform für „Das Entscheidungsproblem ob ein Automat eine Sprache aus dieser Klasse akzeptiert".

Bei diesen drei Klassen handelt es sich um Varietäten von Sprachen, die durch Omega-Gleichungen, die im syntaktischen Monoid der Sprache gelten müssen, definiert werden können. Da viele Unterklassen der regulären Sprachen durch Omega-Gleichungen beschrieben werden können, stellt sich die Frage nach der Komplexität anderer derartiger Varietäten.

Als Obergrenze für beliebige Omega-Gleichungen kann PSPACE bestimmt werden, indem der PSPACE-Algorithmus aus [Ste85b] verallgemeinert wird (Kapitel 1). Als Untergrenze für nichttriviale Fälle ergibt sich NL-schwer durch Abwandlung des Beweises aus [CH91] (Kapitel 3). In Kapitel 4 wird ein Verfahren angegeben, welches es ermöglicht aus einer Menge von Gleichungen, die eine gewisse Bedingung erfüllen, einen NL-Algorithmus zu konstruieren. Die NL-Algorithmen für – und damit die NL-Vollständigkeit von – *piecewise testable* und *dot-depth one* folgen als Sonderfälle. In Kapitel 5 wird eine leichte Abwandlung des PSPACE-Schwere-Beweis für die sternfreien Sprachen aus [CH91] gegeben.

Im Rest dieses Kapitels werden die verwendeten Begriffe definiert und die Notation festgelegt.

1.1 Sprachen

Ein *Alphabet* ist eine endliche Menge von Zeichen. Ein *Wort* über einem Alphabet Σ ist eine beliebig aber endlich lange Folge $a_0 a_1 \cdots a_{n-1}$ von Zeichen a_i aus Σ. Für das leere Wort wird ϵ geschrieben. Die Menge aller Worte über Σ wird mit $\Sigma^\star$ (gesprochen: „Sigma Stern"), die aller nicht leeren Worte mit Σ^+ bezeichnet. $|w|$ bezeichnet die Länge des Wortes w und kann als $|\epsilon| = 0$ und $|wa| = |w| + 1$ für alle $w \in \Sigma^\star$, $a \in \Sigma$ definiert werden.

Das Produkt zweier Worte $w = a_0 a_1 \cdots a_{|w|-1}$ und $v = b_0 b_1 \cdots b_{|v|-1}$ ist das Wort $w \cdot v = a_0 a_1 \cdots a_{|w|-1} b_0 \cdots b_{|v|-1}$. Der Operator „$\cdot$" wird im allgemeinen nicht geschrieben, da in allen Fällen, in denen dadurch nicht klar ist, ob eine Verknüpfung von Zeichen zu einem Wort oder ein Produkt von Worten zu lesen ist, für beide Interpretationen das gleiche Ergebnis herauskommt. Das iterierte

Produkt eines Wortes mit sich selbst wird als Potenz $w^0 = \epsilon$, $w^{n+1} = ww^n$ geschrieben. Bei einem Wort $w = pfs$ bezeichnen wir das Teilworte p als *Präfix*, f als *Faktor* und s als *Suffix* von w. Ein Präfix (Faktor, Suffix) p von w heißt *echtes Präfix* (Faktor, Suffix) wenn $p \neq w$.

Eine *Sprache* über einem Alphabet Σ ist eine Menge von Worten über Σ. Die leere Sprache wird mit ϕ bezeichnet.

Das Produkt zweier Sprachen $L_1, L_2 \subseteq \Sigma^\star$ ist die Sprache $L_1 \cdot L_2 = \{uv \in \Sigma^\star \mid u \in L_1 \wedge v \in L_2\}$. Auch dieser Produktoperator wird im Allgemeinen nicht geschrieben. Sprachen über dem gleichen Alphabet können mit den bekannten Mengenoperationen verknüpft werden. Die Differenz $\Sigma^\star \setminus L$ wird als Komplement von L bezeichnet und $\overline{L}$ geschrieben. Das iterierte Produkt wird erneut als Potenz $L^0 = \{\epsilon\}$, $L^{n+1} = LL^n$ geschrieben. Der Abschluss einer Sprache unter Produkt ist $L^\star = \bigcup_{i \in \mathbb{N}} L^i$.

Das Produkt einer Sprache mit einem Wort lesen wir als Produkt mit der Sprache, die nur dieses Wort enthält; auch hierbei führen Mehrdeutigkeiten, ob ein Produkt von Worten oder ein Produkt von Sprachen zu lesen ist, am Ende zur gleichen Sprache, wodurch auf Klammern verzichtet werden kann.

Die Klasse der *regulären Sprachen* ist der Abschluss der Klasse der endlichen Sprachen unter Produkt, Stern und booleschen Operationen.

1.2 Monoide

Eine *algebraische Struktur* $(M, \cdot)$ ist eine Menge M mit einer inneren Verknüpfung $\cdot \in M \times M \to M$, die Paare von Elementen aus M in M abbildet. Eine *Halbgruppe* ist eine algebraische Struktur, in der das assoziative Gesetz gilt, d. h. für alle $x, y, z \in M$ gilt $(x \cdot y) \cdot z = x \cdot (y \cdot z)$. Ein *Monoid* ist eine Halbgruppe $(M, \cdot)$ mit einem *neutralen Element* $1 \in M$, d. h. für alle $x \in M$ gilt $x \cdot 1 = x = 1 \cdot x$. Die wiederholte Verknüpfung eines Elements mit sich selbst wird durch die Potenzschreibweise $x^0 = 1, x^{n+1} = x^n x$ ausgedrückt. Ein Monoidhomomorphismus $\phi \in M \to N$ von einem Monoid $(M, \cdot)$ in ein Monoid $(N, \circ)$ ist eine Funktion, die mit den Monoidoperationen verträglich ist, d. h. $\phi(x \cdot y) = \phi(x) \circ \phi(y)$.

Sei $S \subseteq M$ eine beliebige Teilmenge von M. Wir sagen, ein Homomorphismus $\phi \in M \to N$ *erkennt* S, wenn es eine Menge $T \subseteq N$ gibt, so dass $x \in S \leftrightarrow \phi(x) \in T$. Ein Monoid N erkennt S, wenn es ein Homomorphismus von M nach N gibt, der S erkennt. Insbesondere erkennt ein Monoid N eine Sprache $L \subseteq \Sigma^\star$, wenn es einen Homomorphismus $\phi \in \Sigma^\star \to N$ gibt, der L erkennt. Die regulären Sprachen sind genau die Sprachen, die von endlichen Monoiden erkannt werden.

Die *syntaktische Kongruenz* $\sim_L$ einer Sprache L ist definiert als $u \sim_L v$ g. d. w. $\forall\, p, s \in \Sigma^\star : pus \in L \leftrightarrow pvs \in L$. Diese Relation ist eine mit dem Produkt verträgliche Kongruenz. Die Menge $\Sigma^\star / \sim_L$ der Äquivalenzklassen $[w]_{\sim_L} = \{u \mid u \sim_L w\}$ und ihr Produkt $[w]_{\sim_L}[v]_{\sim_L} = [wv]_{\sim_L}$ bilden das *syntaktische Monoid* $M(L)$ der Sprache L. Dieses erkennt L mittels des *syntaktischen Homomorphismus* $\eta_L = w \mapsto [w]_{\sim_L}$.

1.3 Automaten

Ein *nichtdeterministischer endlicher Automat* $\mathcal{A}$, kurz NFA (nondeterministic finite automaton), ist ein Quintupel $(Q, \Sigma, q_0, F, \delta)$ einer endlichen Menge Q von *Zuständen*, eines Alphabets Σ, eines *Startzustandes* $q_0 \in Q$, einer Menge von *Endzuständen* $F \in 2^Q$ und einer *Überführungsfunktion* $\delta \in Q \times \Sigma \to 2^Q$ von Paaren aus Zuständen des Automaten und Zeichen des Alphabets in Mengen von Zuständen. Ein Automat akzeptiert ein Wort $w = a_1 a_2 \cdots a_n$ g. d. w. es eine Folge von Zuständen $q_1, q_2, \ldots q_n$ gibt, so dass $q_1 \in \delta(q_0, a_0)$, $q_{i+1} \in \delta(q_i, a_i)$, und $q_n \in F$ gilt. Die Sprache $L(\mathcal{A})$ ist die Sprache aller Worte, die der Automat $\mathcal{A}$ akzeptiert. Die Sprachen, die von endlichen Automaten akzeptiert werden, sind genau die regulären Sprache. Zur Vereinfachung der Notation kürzen wir die Relation $\{(q, q') \in Q \times Q \mid q' \in \delta(q, a)\}$ mit $a^{\mathcal{A}}$ ab; statt $q' \in \delta(q, a)$ schreiben wir also $q\ a^{\mathcal{A}}\ q'$, oder, wenn der Automat aus dem Kontext bekannt ist, einfach $q\ a\ q'$.

Ein Automat ist vollständig, wenn es für jeden Zustand q und jedes Zeichen a einen Folgezustand q' gibt, so dass gilt $q\ \ a\ \ q'$. Zu jedem Automat kann ein vollständiger Automat, der die gleiche Sprache akzeptiert, konstruiert werden. Dafür wird ein neuer *toter Zustand* d hinzugefügt und die Überführungsfunktion um $q\ a\ d \leftrightarrow q = d \vee \neg \exists\, q' \in Q : q\ a\ q'$ erweitert. Im Folgenden nehmen wir an, dass alle Automaten vollständig sind.

Die Komposition von zwei Relationen $R \in M \times N$ und $S \in N \times O$ ist definiert als $RS = \{(x, z) \mid \exists\, y \in N : x\ R\ y \wedge y\ S\ z\} \in M \times O$. Wir erweitern den $^{\mathcal{A}}$-Operator auf Worte $w \in \Sigma^\star$ durch $\epsilon^{\mathcal{A}} = \{(x, x) \mid x \in Q\}$, $(wa)^{\mathcal{A}} = w^{\mathcal{A}} a^{\mathcal{A}}$ für alle $a \in \Sigma$, $w \in \Sigma^\star$. Der Abschluss der Relationenmenge $\{a^{\mathcal{A}} \in Q \times Q \mid a \in \Sigma\}$ unter Komposition ist das *Transitionsmonoid* $M(\mathcal{A})$ des Automaten $\mathcal{A}$. Wir haben $w \in L(\mathcal{A})$ g. d. w. es einen Zustand $q \in F$ gibt, so dass $q_0\ w^{\mathcal{A}}\ q$. Mit anderen Worten, $\phi = w \mapsto w^{\mathcal{A}}$ ist ein Monoidhomomorphismus von $\Sigma^\star$ in $2^{Q \times Q}$, der die Sprache $L(\mathcal{A})$ erkennt. $\phi(L(\mathcal{A}))$ sind genau diejenigen Relationen, die den Startzustand mit einem Endzustand verbinden, d. h. $w \in L(\mathcal{A})$ g. d. w. $\exists\, q \in F : q_0\, w^{\mathcal{A}}\, q$.

Wenn die Relation $a^{\mathcal{A}}$ für alle $a \in \Sigma$ rechtseindeutig ist, ist $\mathcal{A}$ ein *deterministischer endlicher Automat*, kurz DFA (deterministic finite automaton). Die Klasse der Sprachen, die von einem DFA akzeptiert werden, ist ebenfalls die Klasse der regulären Sprachen Es ist einfach zu zeigen, dass die auf Worte erweiterten Überführungsrelationen ebenfalls rechtseindeutig sind. Wir definieren für DFA folgende Abkürzung: $q\, w = \iota\, q' : q\ w\ q'$.

1.4 Idempotente Elemente

Ein Element e eines Monoids $(M, \cdot)$ heißt *idempotent*, wenn $e \cdot e = e$. In einem endlichen Monoid M gilt: Zu jedem Element $x \in M$ gibt es eine Potenz x^k die idempotent ist; wir sagen: x generiert das idempotente Element x^k.

Beweis. Die Folge $x^0, x^1, ..., x^{|M|}$ enthält $|M| + 1$ Terme, M selbst aber nur $|M|$ Elemente, also muss es zwei Exponenten $0 \leq i < j \leq |M|$ geben, so dass $x^i = x^j$. Setze $p = j - i$, dann erhalten wir, durch wiederholtes Einsetzen dieser Gleichung in sich selbst, $x^i = x^i x^p = x^i x^p x^p = \cdots = x^i x^{mp}$ für alle m und $x^l x^i = x^l x^i x^{mp}$ für alle m und l. Sei k das kleinste Vielfache von p, so dass $i \leq mp = k$ (hierbei

gilt $k < i+p \leq |M|$). Das gesuchte idempotente Element ist nun: $x^k = x^{k-i}x^i = x^{k-i}x^ix^{mp} = x^kx^k$. Die Potenz x^{k-i} kann gebildet werden, da $i \leq k$. □

Das von x generierte idempotente Element ist eindeutig.

Beweis. Angenommen $x^k \neq x^{k'}$ wären zwei unterschiedliche von x generierte Idempotente. Dann gilt $x^k = x^kx^k = x^{3k} = \cdots = x^{k'k} = \cdots = x^{3k'} = x^{k'}x^{k'} = x^{k'}$ im Widerspruch zur Annahme. □

Für jedes Monoid M gibt es eine natürliche Zahl ω, so dass für alle $x \in M$ gilt: $x^\omega = x^\omega x^\omega$ und $\omega \leq (|M|-1)!$.

Beweis. Wie zuvor gezeigt gibt es für jedes Element $x \in M$ einen Exponent $k_x < |M|$, so dass $x^{k_x} = x^{k_x}x^{k_x} = \cdots = x^{nk_x}$ für alle Vielfachen nk_x von k_x. Ein Vielfaches aller $k_x, x \in M$ erfüllt diese Bedingung für alle x. $(|M|-1)!$ ist ein Vielfaches aller k_x, daher $\omega \leq (|M|-1)!$ □

1.5 Varietäten

Eine Klasse endlicher Monoiden, die unter Untermonoiden, surjektiver homomorpher Abbildung und endlichem Produkt abgeschlossen ist, heißt eine *Varietät endlicher Monoide.* Varietäten werden häufig durch eine Menge von *Omega-Gleichungen* definiert:

Definition 1 (Omega-Gleichungen)**.** Sei X ein endlicher Vorrat an Variablenzeichen, $(M, \cdot)$ ein Monid mit neutralem Element 1 und $g \in X \to M$ eine Variablenbelegung. Wir definieren rekursiv *Omega-Terme* über die Variablen X und erweitern die Belegung g auf beliebige Omega-Terme.

i. 1 ist ein Omega-Term, $g(1) = 1$

ii. Jedes $x \in X$ ist ein Omega-Term, $g(x)$ ist bereits definiert.

iii. Wenn e und f Omega-Terme sind, so ist auch $e \cdot f$ ein Omega-Term, $g(ef) = g(e) \cdot g(f)$.

iv. Wenn e ein Omega-Term ist, so ist auch e^ω ein Omega-Term, $g(e^\omega)$ ist das eindeutige von $g(e)$ erzeugte idempotente Element $g(e)^\omega$.

v. nichts sonst ist ein Omega-Term

Wenn e und f Omega-Terme sind, so ist $e = f$ eine Omega-Gleichung.

Ein Monoid M erfüllt eine Gleichung $e = f$, wenn für alle Belegungen $g \in X \to M$ gilt: $g(e) = g(f)$. M erfüllt eine Menge $e_1 = f_1, \ldots, e_n = f_n$, wenn M alle Gleichungen aus der Menge erfüllt. Wir schreiben dafür $M \in [\![e_1 = f_1, \ldots, e_n = f_n]\!]$.

Jeder Varietät von Monoiden $\mathbf{V}$ entspricht eine *Varietät von Sprachen* $\mathcal{V}$, die diejenigen Sprachen enthält, deren syntaktisches Monoid zu $\mathbf{V}$ gehört. Da es sich hierbei um eine echte Klasse handelt (schon die Klasse aller Alphabete, d. h. die Klasse aller endlichen Mengen, ist eine echte Klasse), wird meistens ein beliebiges Alphabet Σ vorgegeben und die Menge $\mathcal{V}(\Sigma^\star)$ der Sprachen über Σ, deren syntaktisches Monoid zu $\mathbf{V}$ gehört, verwendet.

2 PSPACE-Algorithmen

In diesem Kapitel geben wir ein Algorithmenschema an, das einen PSPACE-Algorithmus für beliebige Omega-Gleichungen erzeugt, und zeigen dadurch eine Verallgemeinerung von Theorem 2.i in [Ste85b].

Theorem 1. *Für jede Varietät $\mathcal{V} = [\![l_1 = r_1, \ldots, l_n = r_n]\!]$, die durch Omega-Gleichungen definiert ist gilt: Das Entscheidungsproblem „gegeben ein DFA $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$: Gehört die von $\mathcal{A}$ akzeptierte Sprache $L(\mathcal{A})$ zu $\mathcal{V}(\Sigma^\star)$“ ist in PSPACE entscheidbar.*

Im Folgenden wird ein nichtdeterministischer PSPACE-Algorithmus für das Entscheidungsproblem $L(\mathcal{A}) \notin [\![l = r]\!]$ für beliebige Omega-Terme l und r angegeben. Die Existenz des für Theorem 1 benötigten Algorithmus folgt aus dem Abschluss von PSPACE unter Schnitt und Komplement.

Lemma 2. *Seien l und r Omega-Terme über eine Menge X von Variablen, $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$ ein DFA, $L(\mathcal{A})$ die von $\mathcal{A}$ akzeptierte Sprachen, $M(L(\mathcal{A}))$ und $\eta_{L(\mathcal{A})}$ ihr syntaktisches Monoid und der syntaktische Morphismus und $M(\mathcal{A})$ das Transitionsmonoid von $\mathcal{A}$. Dann sind die folgenden Aussagen äquivalent:*

$$L(\mathcal{A}) \notin [\![l = r]\!] \tag{2.1}$$
$$\exists\, g \in X \to M(L(\mathcal{A})) : g(l) \neq g(r) \tag{2.2}$$
$$\exists\, g' \in X \to \Sigma^\star : \eta_{L(\mathcal{A})}(g'(l)) \neq \eta_{L(\mathcal{A})}(g'(r)) \tag{2.3}$$
$$\exists\, g' \in X \to \Sigma^\star, p, s \in \Sigma^\star : pg'(l)s \in L(\mathcal{A}) \not\leftrightarrow pg'(r)s \in L(\mathcal{A}) \tag{2.4}$$
$$\exists\, g' \in X \to \Sigma^\star, p, s \in \Sigma^\star : q_0\, pg'(l)s^{\mathcal{A}} \in F \not\leftrightarrow q_0\, pg'(r)s^{\mathcal{A}} \in F \tag{2.5}$$
$$\exists\, g'' \in X \to M(\mathcal{A}), p, s \in \Sigma^\star : q_0\, p^{\mathcal{A}}\, g''(l)\, s^{\mathcal{A}} \in F \not\leftrightarrow q_0\, p^{\mathcal{A}}\, g''(r)\, s^{\mathcal{A}} \in F \tag{2.6}$$

Beweis. Die Äquivalenzen (2.1)↔(2.2), (2.3)↔(2.4) und (2.4)↔(2.5) folgen aus den Definitionen der von einer Omega-Gleichung definierten Varietät, des syntaktischen Monoids und der von einem Automaten akzeptierten Sprache.

Für (2.2)↔(2.3) und (2.5)↔(2.6) wähle $g(x) = \eta_{L(\mathcal{A})}(g'(x))$ und $g''(x) = g'(x)^{\mathcal{A}}$. Dass g und g'' für ein gegebenes g' existieren ist klar. Die Existenz von g' für gegebenes g (g'') folgt aus der Definition des syntaktischen (Transitions-) Monoids als Bildmenge von $\Sigma^\star$ unter dem entsprechenden Homomorphismus. $g(t) = \eta_{L(\mathcal{A})}(g'(t))$ und $g''(t) = g'(t)^{\mathcal{A}}$ für einen beliebigen Omega-Term t wird durch Induktion über den Aufbau der Omega-Terme bewiesen. □

Ob Bedingung (2.6) in einem Automaten gilt, kann durch einen nichtdeterministischen *guess and check*-Algorithmus geprüft werden. Dieser errät eine Variablenbelegung $g'' = x \mapsto q \mapsto q\, g'(x)^{\mathcal{A}} \in$

$X \to Q \to Q$, berechnet die Werte $g''(l)$ und $g''(r)$ der Omega-Terme l und r unter dieser Belegung und rät schließlich ein Prefix und ein Suffix, welche zeigen, dass die Belegung einem Beispiel für $g(l) \neq g(r)$ im syntaktischen Monoid entspricht.

Da kein fester Algorithmus angegeben wird sondern ein Algorithmenschema für beliebige Variablenmengen X und Omega-Terme l und r, muss die Definition des Algorithmus von seiner Ausführung unterschieden werden. Wir schreiben, zur besseren Unterscheidung, den Algorithmus selbst in Pseudocode und die Anweisungen zum Erzeugen des Algorithmus auf Deutsch. Parameter und Indizes werden, wenn für die Ausführung bestimmt, in Klammern und, wenn für das Schema, als Subskripte geschrieben. Um die Omega-Terme rekursiv auswerten zu können und auch der Übersichtlichkeit wegen wird der Algorithmus aus Teilalgorithmen zusammengesetzt. Hierbei handelt es sich um eine rekursive Definition, die durch Einsetzen konkreter l und r aufgelöst werden kann, und nicht um Unterprogrammaufrufe. Wir verwenden jedoch die üblichen Sichtbarkeitsregeln von Unterprogrammaufrufen, um nicht explizit eindeutige Bezeichner für die Programmvariablen erzeugen zu müssen. Bei der Bewertung des Platzbedarfs zählen wir den für „Parameter“ benötigten Platz beim „Aufrufer“.

Algorithmus 1 : Algorithmus $Guess_X(\mathcal{A}, \mathsf{g})$ zum Raten einer Belegung $\mathsf{g} \in X \to M(\mathcal{A})$

Input : $\mathcal{A} = (\mathsf{Q}, \Sigma, \mathsf{q}_0, \mathsf{F}, \delta)$ ein DFA
Output : $\mathsf{g} \in X \to \mathsf{Q} \to \mathsf{Q}$ eine Variablenbelegung
Data : $\mathsf{a} \in \Sigma, \mathsf{stop} \in \{0, 1\}$
Für alle $x \in X$
3 **forall the** $\mathsf{q} \in \mathsf{Q}$ **do** $\mathsf{g}_x[\mathsf{q}] \leftarrow \mathsf{q}$;/* $g(x) \leftarrow \epsilon^{\mathcal{A}}$ */
guess stop;
while $\mathsf{stop} = 0$ **do**
guess stop; **guess** a;
8 **forall the** $\mathsf{q} \in Q$ **do** $\mathsf{g}_x[\mathsf{q}] \leftarrow \delta(\mathsf{g}_x[\mathsf{q}], \mathsf{a})$;/* $g(x) \leftarrow g(x)\, a^{\mathcal{A}}$ */

Lemma 3. *Algorithmus 1 ist ein PSPACE-Algorithmus und errät alle Belegungen g'' aus $X \to M(\mathcal{A})$.*

Beweis. (Korrektheit) In Zeile 3 wird g_x mit $q \mapsto q = \epsilon^{\mathcal{A}} \in M(\mathcal{A})$ initialisiert. Wenn vor der Ausführung von Zeile 8 $\mathsf{g}_x = f \in M(\mathcal{A})$, dann ist nach ihrer Ausführung $\mathsf{g}_x = q \mapsto f(q)\, a^{\mathcal{A}}$ ebenfalls in $M(\mathcal{A})$. Da Zeile 3 für alle g_x ausgeführt wird und g_x sonst nur in Zeile 8 verändert wird, ist nach Ablauf von Algorithmus 1 $\mathsf{g}_x \in M(\mathcal{A})$ für alle $x \in X$.

(Vollständigkeit) Da $g''(x) \in M(\mathcal{A})$, gibt es eine Folge von Zeichen $a_1 a_2 \cdots a_n$, so dass $g''(x) = a_1 a_2 \cdots a_n{}^{\mathcal{A}}$. $\mathsf{g}_x = g''(x)$ wird durch diejenige Berechnung geladen, die bei Ausführung der Instanzierung des Algorithmenrumpfs für x, n mal $\mathsf{stop} = 0$ rät, beim $n+1$-ten Mal $\mathsf{stop} = 1$ rät und im i-ten Schleifendurchlauf $a = a_i$ rät. Eine solche Berechnung gibt es für jedes $g''(x)$ und jedes x, also auch für jedes g''.

(PSPACE) Um keine Annahmen über die Codierung, in der $\mathcal{A}$ gegeben ist, treffen zu müssen, kann a als Position eines Vorkommens dieses Zeichens auf dem Eingabeband gespeichert werden. Bei einer Eingabe der Länge n werden dafür $\lceil \log n \rceil$ Bit benötigt. □

Algorithmus 2 : Algorithmus $Eval_{X,e}(\mathsf{g},\mathsf{e})$ zur Berechnung des Wertes $\mathsf{e} = \mathsf{g}(e)$ des Omega-Terms e unter der Belegung g.

```
   Input : g ∈ X → Q → Q eine Belegung
   Output : e ∈ Q → Q Wert von e unter der Belegung g
   Wenn e = 1
3  |  forall the q ∈ Q do e[q] ← q;
   Wenn e ∈ X
   |  forall the q ∈ Q do e[q] ← g_e[q];
   Wenn e = lr
   |  Data : l, r ∈ Q → Q
8  |  Eval_{X,l}(g, l); Eval_{X,r}(g, r);
10 |  forall the q ∈ Q do  e[q] ← r[l[q]] ;
   Wenn e = l^ω
   |  Data : l ∈ Q → Q
   |  Data : done ∈ {0, 1} = 0
13 |  Eval_{X,l}(g, l);
15 |  forall the q ∈ Q do e[q] ← q;
17 |  repeat
   |  |  if ¬done then
20 |  |  |  forall the q ∈ Q do  e[q] ← e[l[q]] ;
   |  |  done ← 1;
   |  |  forall the q ∈ Q do
   |  |  |  if e[q] ≠ e[e[q]] then  done ← 0 ;
   |  until done;
```

Lemma 4. *Algorithmus 2 berechnet aus einer Variablenbelegung* $\mathsf{g} \in x \to \mathsf{Q} \to \mathsf{Q}$ *den Wert* $\mathsf{e} = \mathsf{g}(e)$ *eines beliebigen Omega-Terms unter Belegung* g*. Der benötigte Speicherplatz ist polynomiell in der Größe des Automaten.*

Beweis. Für $e = 1$ ist $g(e) = \epsilon^{\mathcal{A}} = q \mapsto q$, in Zeile 3 wird e auf diesen Wert gesetzt. Wenn $e \in X$, kann e einfach aus der Belegung kopiert werden.

Wenn $e = lr$, dann ist nach Induktionsannahme nach Ausführung von $Eval$ in Zeile 8 $\mathsf{l} = \mathsf{g}(l)$ und $\mathsf{r} = \mathsf{g}(r)$. Nach Definition ist $\mathsf{g}(lr) = \mathsf{g}(l)\mathsf{g}(r) = q \mapsto \mathsf{g}(r)(\mathsf{g}(l)(q))$ was in Zeile 10 berechnet wird.

Wenn $e = l^\omega$, dann ist nach Induktionsannahme nach Ausführung von Zeile 13 $\mathsf{l} = \mathsf{g}(l)$. Die Variable e wird in Zeile 15 mit $\epsilon^{\mathcal{A}}$ initialisiert und in Zeile 20 mit $\mathsf{g}(l)$ verknüpft, e hat also nach Ende der Ausführung den Wert $\mathsf{g}(l)^k$ für ein $k \geq 1$. Da die Schleife in Zeile 17 erst verlassen wird wenn $\mathsf{e}(q) = \mathsf{e}(\mathsf{e}(q))$, ist e am Ende idempotent, und damit das eindeutige von $\mathsf{g}(l)$ erzeugte Idempotente $\mathsf{g}(l)^\omega$

(PSPACE) Es wird nur Platz zum Speichern der Zwischenergebnisse l und r benötigt. Diese sind Abbildungen der Menge der Zustände des Automaten auf sich selbst, und können als Liste von geordneten Paaren $(q, \mathsf{l}(q))$ gespeichert werden. Ein Zustand kann wieder als Index eines Vorkommens des Zustands auf dem Eingabeband gespeichert werden, wofür, wenn die Eingabe die Länge n hat,

$\lceil \log n \rceil$ Bit benötigt werden. Da die Liste rechtseindeutig ist, kann sie höchstens $|Q| < n$ Einträge enthalten, also ist der gesamte Platzbedarf in $O(n \log n)$ □

Algorithmus 3 : Algorithmus $Test\_Equation_{X,l,r}(\mathcal{A})$ zum Test, ob $L(\mathcal{A}) \in [\![l = r]\!]$, l und r Omega-Terme über die Variablen X.

```
   Input : A = (Q, Σ, q_0, F, δ) ein DFA
   Data : g ∈ X → Q → Q
   Data : l, r ∈ Q → Q
   Guess_X(A, g);
   Eval_{X,l}(g, l); Eval_{X,r}(g, r);
   Data : o, p, q ∈ Q
   Data : a ∈ Σ
   Data : stop ∈ {0, 1}
   q ← q_0;
   guess stop;
   while stop = 0 do
7  |   guess a;guess stop;
   |   q ← q a^A;
   o ← l[q]; p ← r[q];
   guess stop;
   while stop = 0 do
13 |   guess a;guess stop;
   |   o ← o a^A; p ← p a^A;
   if o ∈ F ⇎ p ∈ F then
   |   accept; /*  A ∉ [[l = r]]                                  */
   else
   |   reject;
```

Lemma 5. *Algorithmus 3 ist ein PSPACE-Algorithmus und akzeptiert auf Eingabe $\mathcal{A}$ g. d. w. $L(\mathcal{A}) \notin [\![e = f]\!]$.*

Beweis. (Korrektheit) Bei einer akzeptierenden Rechnung entspricht nach Lemma 3 der Wert von g einer Variablenbelegung $g'' \in X \to Q \to Q$. Ebenso entsprechen die in Zeilen 7 und 13 geratenen Zeichen zwei Worten p und s. Nach Lemma 4 ist $\mathsf{l} = g''(l)$ und $\mathsf{r} = g''(r)$. Da der Algorithmus akzeptiert, muss gelten, dass $q_0\, p^{\mathcal{A}}\, g''(l)\, s^{\mathcal{A}} \in F \not\leftrightarrow q_0\, p^{\mathcal{A}}\, g''(r)\, s^{\mathcal{A}} \in F$ was nach Lemma 2 äquivalent ist zu $L(\mathcal{A}) \notin [\![e = f]\!]$.

(Vollständigkeit) Wenn $L(\mathcal{A}) \notin [\![e = f]\!]$, dann gibt es nach Gleichung 2.6 Worte p, s und eine Belegung g'', so dass $q_0\, p^{\mathcal{A}}\, g''(l)\, s^{\mathcal{A}} \in F \not\leftrightarrow q_0\, p^{\mathcal{A}}\, g''(r)\, s^{\mathcal{A}} \in F$. Nach Lemma 3 gibt es eine Rechnung, die diese Belegung rät. Ebenso gibt es eine Rechnung, die die Zeile 7 $|p|$ mal ausführt und beim i-ten Durchlauf das i-te Zeichen von p rät und Zeile 13 $|s|$ mal ausführt und beim i-ten Durchlauf das i-te Zeichen von s rät. Diese Rechnung endet mit $\mathsf{o} = q_0\, p^{\mathcal{A}}\, g''(l)\, s^{\mathcal{A}}$ und $\mathsf{p} = q_0\, p^{\mathcal{A}}\, g''(r)\, s^{\mathcal{A}}$ und nach Annahme muss der Algorithmus akzeptieren.

(PSPACE) Wie bei den Teilalgorithmen benötigen die Zustände $\mathsf{o}, \mathsf{p}, \mathsf{q}$ und das Zeichen a maximal $\lceil \log n \rceil$ Bit. Die Elemente $\mathsf{l}, \mathsf{r}, \mathsf{g}_x$ des Transitionsmonoiden können wieder in $O(n \log n)$ Bit gespeichert werden, die Anzahl der verschiedenen g_x hängt hierbei nicht von der Größe der Eingabe ab. Da die Teilalgorithmen $Eval$ und $Guess$ ebenfalls $O(n \log n)$ Bit benötigen, ist auch der Platzbedarf des kompletten Algorithmus in $O(n \log n)$. □

3 NL Schwere

Theorem 6. *Für jede Varietät $\mathcal{V} = [\![l_1 = r_1, \ldots, l_n = r_n]\!]$, die durch Omega-Gleichungen definiert ist und die nicht alle regulären Sprachen enthält ($\mathcal{V}(\Sigma^\star) \subsetneq \mathcal{R}eg(\Sigma^\star)$), und für jedes Alphabet das mindestens zwei Zeichen enthält gilt: Das Entscheidungsproblem „gegeben ein DFA $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$: gehört die von $\mathcal{A}$ akzeptierte Sprache $L(\mathcal{A})$ zu $\mathcal{V}(\Sigma^\star)$" ist NL-schwer bezüglich log-space-Reduktionen.*

Der Beweis erfolgt durch Logspace-Reduktion des NL-vollständigen *graph reachability problem* [HU79]. Dies ist das Entscheidungsproblem: „Gegeben ein Digraph $(V, E), E \subseteq V \times V$ und zwei Knoten $s, t \in V$: Gibt es einen Pfad von s nach t," d. h. gibt es eine Folge von Knoten $v_1, v_2, \ldots, v_k \in V$, so dass $(s, v_1) \in E$, $(v_k, t) \in E$ und $(v_i, v_{i+1}) \in E$ für alle $1 \leq i < k$. Wir können uns auf Instanzen beschränken, in denen jeder Knoten höchstens zwei Nachfolger hat. Um das allgemeine Problem darauf zu reduzieren, muss lediglich jeder Knoten durch eine Kette von Knoten ersetzt und die ausgehenden Kanten auf diese verteilt werden.

Wir geben nun eine Logspace-Reduktion einer Instanz $((V, E), s, t)$ des *reachability*-Problems auf einen Automaten $\mathcal{A}$ mit der Eigenschaft, dass $L(\mathcal{A}) \in \mathcal{V}(\Sigma^\star)$ g. d. w. $((V, E), s, t) \in$ *reachability*. Da wir im Gegensatz zu [CH91] nicht fordern, dass der Automat, der bei der Reduzierung entsteht minimal ist, kann der Beweis problemlos für eine beliebige Varietät geführt werden.

Da $\mathcal{V}(\Sigma^\star) \subsetneq \mathcal{R}eg(\Sigma^\star)$, gibt es einen DFA $\mathcal{A}' = (Q', \Sigma, q_0', F', \delta')$ mit $L(\mathcal{A}') \notin \mathcal{V}(\Sigma^\star)$. Seien $a \neq b \in \Sigma$ zwei beliebige Zeichen. Die Reduktion erzeugt nun einen DFA, der erst eine Codierung für eine Lösung des *reachability*-Problems als Wort über $\{a, b\}$ liest und dann in eine Kopie von $\mathcal{A}'$ übergeht. Wir können eine beliebige totale Ordnung $\leq$ auf den Knoten V voraussetzen (z. B. Reihenfolge des ersten Vorkommens auf dem Eingabeband).

Der DFA $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$ ist wie folgt definiert:

i. Die Zustände $Q = \{0\} \times V \cup \{1\} \times Q' \cup \bot$ sind die disjunkte Vereinigung der Knoten des *reachability problem* mit den Zuständen von $\mathcal{A}'$, sowie ein toter Zustand.

ii. Das Alphabet Σ wird unveränder übernommen.

iii. Der Startzustand $q_0 = (0, s)$ entspricht dem Startknoten des *reachability problem*.

iv. Die Endzustände $F = \{(1, q') \mid q' \in F'\}$ entsprechen den Endzuständen von $\mathcal{A}'$.

v. Die Überführungsfunktion δ codiert das *reachability problem* und bildet A' nach.

   a) $(0, v)\, a^{\mathcal{A}} = (0, v')$, wenn $(v, v') \in E$, $v \neq t$ und v genau eine ausgehenden Kante hat.

   b) $(0, v)\, a^{\mathcal{A}} = (0, v')$ und $(0, v)\, b^{\mathcal{A}} = (0, v'')$, wenn $v \neq t$, $(v, v') \in E$, $(v, v'') \in E$ und $v' < v''$.

c) $(0,t)\ a^{\mathcal{A}} = (1, q_0')$

d) $(1,q')\ c^{\mathcal{A}} = (1, q'\ c^{\mathcal{A}'})$ für alle $q' \in Q', c \in \Sigma$

e) $q\ c^{\mathcal{A}} = \bot$ sonst

Dass der Automat $\mathcal{A}$ deterministisch ist, folgt daraus, dass $\mathcal{A}'$ ebenfalls deterministisch ist und dass wir ausgeschlossen haben, dass es drei Kanten $(v,v'),(v,v''),(v,v''') \in E$ gibt, die vom gleichen Knoten ausgehen. Um zu zeigen, dass die Reduktion von einer deterministischen Turingmaschine mit logarithmisch begrenztem Arbeitsband berechnet werden kann, skizzieren wir die Berechnung der Übergänge, die *reachability problem* codieren. Die Turingmaschine durchläuft in zwei verschachtelten Schleifen alle Knoten und alle Kanten und gibt beim ersten Antreffen einer Kante, die vom aktuell betrachteten Knoten v ausgeht, den Übergang $(0,v)\ a^{\mathcal{A}}$ und beim zweiten $(0,v)\ b^{\mathcal{A}}$ aus. Am Ende der Kantenliste werden alle Übergänge von $(0,v)$ in den toten Zustand geschrieben. Die Turingmaschine muss hierzu nur speichern, ob bisher keine, eine oder zwei von v ausgehende Kanten angetroffen wurden (konstant) und welcher Knoten und welche Kante momentan bearbeitet werden (jeweils logarithmish). Das Alphabet Σ und der Automat $\mathcal{A}'$ sind im endlichen Programm des reduzierenden Automaten gespeichert und tragen nicht zu dessen Bandkomplexität bei.

Lemma 7. $((V,E),s,t) \notin$ reachability *g. d. w.* $L(\mathcal{A}) \in \mathcal{V}(\Sigma^\star)$

Beweis. ($\rightarrow$) Wenn $((V,E),s,t) \notin$ *reachability*, d. h. wenn es keinen Pfad von s nach t gibt, dann gibt es auch in $\mathcal{A}$ keinen Pfad von $(0,s)$ nach $(0,t)$. Damit sind alle $(1,q)$-Zustände, also insbesondere alle akzeptierenden Zustände, nicht vom Startzustand aus erreichbar und damit ist $L(\mathcal{A}) = \phi$. Das syntaktische Monoid der leeren Sprache ist aber das triviale Monoid $\{1\}$ und in jeder Varietät enthalten, also $L(\mathcal{A}) \in \mathcal{V}(\Sigma^\star)$.

($\leftarrow$) Wenn $((V,E),s,t) \in$ *reachability*, so entspricht jeder Lösung des *reachability problem* ein Wort $w \in \{a,b\}^\star$, so dass $(0,s)\ w^{\mathcal{A}} = (0,t)$.

Da $L(\mathcal{A}') \notin \mathcal{V}(\Sigma^\star)$, gibt es, mit Lemma 2, für eine der Gleichungen $l = r$, die $\mathcal{V}$ charakterisieren, eine Belegung $g' \in X \to \Sigma^\star$ und Präfix und Suffix $p, s \in \Sigma^\star$, so dass $q_0'\, pg'(l)s^{\mathcal{A}'} \in F' \not\leftrightarrow q_0'\, pg'(r)s^{\mathcal{A}'} \in F'$. Durch den Aufbau von $\mathcal{A}$ folgt daraus $(1,q_0')\, pg'(l)s^{\mathcal{A}} \in F \not\leftrightarrow (1,q_0')\, pg'(r)s^{\mathcal{A}} \in F$ und mit $(0,s)\ w^{\mathcal{A}} = (0,t)$ und $(0,t)\ a^{\mathcal{A}} = (1,q_0')$ folgt $q_0\ wapg'(l)s^{\mathcal{A}} \in F \not\leftrightarrow q_0\ wapg'(r)s^{\mathcal{A}} \in F$. wap ist also ein geeignetes Präfix, um, wiederum mit Lemma 2, zu zeigen, dass $L(\mathcal{A}) \notin \mathcal{V}(\Sigma^\star)$. □

4 Gleichungen in NL

In Algorithmus 3 wird überlogarithmischer Platz nur benötigt, um die Elemente des Transistionsmonoiden zu speichern. Die Elemente des Transitionsmonoid werden außerdem, außer beim Berechnen von Termen der Form t^ω, nur an einer konstanten Anzahl an Zuständen ausgewertet. Sobald auf die Auswertung von Termen der Form t^ω verzichtet werden kann, lässt sich eine Gleichung in NL testen, indem der Urbildbereich der Elemente des Transitionsmonoids auf die konstante (d. h. nur von der Gleichung abhängige) Menge der im Algorithmus abgefragten Zustände beschränkt wird. Trotz der Einschränkung t^ω nicht berechnen zu können, lassen sich auch Gleichungen mit Idempotenten in NL entscheiden. Dies ist möglich, wenn die Gleichungen es zulassen, direkt eine Belegung zu raten in der bei allen Termen der Form t^ω das erzeugende Element t bereits idempotent ist. Die Charakterisierungen und NL-Algorithmen aus der Literatur für *piecewise testable* [Sim75] und *dot depth one* [Ste85a] beruhen im Endeffekt auf diesem Vorgehen.

Theorem 8. *Für jede Varietät $\mathcal{V}$ gilt: das Entscheidungsproblem „gegeben ein DFA $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$: gehört die von $\mathcal{A}$ akzeptierte Sprache $L(\mathcal{A})$ zu $\mathcal{V}(\Sigma^\star)$“ ist in NL, g. d. w. es eine endliche Menge von Gleichungen gibt, so dass $\mathcal{V} = [\![l_1 = r_1, \ldots, l_n = r_n]\!]$ und alle Gleichungen $l_i = r_i$, $i \leq i \leq n$ gilt: Für jede Belegung $g \in X \to M$ gibt es eine Belegung $\hat{g}$, so dass $\hat{g}(l_i) = g(l_i)$, $\hat{g}(r_i) = g(r_i)$ und $\hat{g}(t) = g(t^\omega)$ für alle Teilausdrücke t^ω aus l_i und r_i.*

Da NL unter Schnitt abgeschlossen ist, betrachten wir im Folgenden nur eine Gleichung. Wir zeigen zunächst, dass die angegebene Bedingung dazu führt, dass alle in der Gleichung vorkommenden Idempotente geraten werden können.

Lemma 9. *Für die Belegungen g und $\hat{g}$ aus Theorem 8 gilt: $\hat{g}(t) = \hat{g}(t)^\omega$ für alle Teilausdrücke t^ω, die in l oder r vorkommen.*

Beweis. Nach der Annahme aus Theorem 8 ist $\hat{g}(t) = g(t^\omega)$, also idempotent, also $\hat{g}(t) = \hat{g}(t)^2 = \hat{g}(t)^3 = \cdots = \hat{g}(t)^\omega$. □

Definition 2. Für einen Omega-Term t sei $F(t)$ der Omega-Term, der durch Ersetzen aller Teilausdrücke u^ω durch u entsteht. Das heißt $F(1) = 1$, $F(x) = x$ für $x \in X$, $F(lr) = F(l)F(r)$ und $F(l^\omega) = F(l)$.

Lemma 10. *Für die Belegungen g und $\hat{g}$ aus Theorem 8 gilt: $\hat{g}(F(t^\omega)) = \hat{g}(t)$ für alle Teilausdrücke t^ω die in l oder r vorkommen.*

Beweis. Durch Induktion über die Tiefe, zu der Teilausdrücke der Form u^ω innerhalb von t verschachtelt sind.

Wenn t keine Teilausdrücke der Form u^ω enthält, ist $t = F(t)$ und damit gilt $\hat{g}(F(t^\omega)) = \hat{g}(t)$.

Wenn $t \neq F(t)$, dann hat t die Form $u_0 v_1^\omega u_1 \cdots v_n^\omega u_n$ mit $u_i \in X^\star$ (d. h. $u_i = F(u_i)$) und Ausdrücken v_i, in denen Omega-Potenzen weniger tief verschachtelt sind. Nach Induktionsannahme gilt $\hat{g}(F(v_i^\omega)) = \hat{g}(v_i)$ und mit Lemma 9 gilt $\hat{g}(F(v_i^\omega)) = \hat{g}(v_i)^\omega$. Also ist $\hat{g}(F(t^\omega)) = \hat{g}(F(u_0))\hat{g}(F(v_0^\omega))\hat{g}(F(u_1)) \cdots \hat{g}(F(u_n)) = \hat{g}(u_0)\hat{g}(v_0)^\omega \cdots \hat{g}(u_n) = \hat{g}(t)$. □

Lemma 11. *Für die Belegungen g und $\hat{g}$ aus Theorem 8 gilt: $\hat{g}(F(l)) = \hat{g}(l)$ und $\hat{g}(F(r)) = \hat{g}(r)$.*

Beweis. Mit Lemmata 10 und 9 wie Induktionsschritt von Lemma 10 □

Wir nutzen aus, dass NL unter Komplement abgeschlossen ist (Satz von Immerman und Szelepcsényi) und betrachten im Folgenden das konträre Problem.

Lemma 12. *Für alle l, r, die die Bedingung aus Theorem 8 erfüllen, sind die folgenden Bedingungen äquivalent*

$$\exists\, g : g(l) \neq g(r) \tag{4.1}$$

$$\exists\, \hat{g} : \hat{g}(F(e)) = \hat{g}(F(ee)) \text{ für alle } e^\omega \text{ in } l \text{ und } r \wedge \hat{g}(F(l)) \neq \hat{g}(F(r)) \tag{4.2}$$

Beweis. ($\rightarrow$) Zu jedem g gibt es jenes $\hat{g}$, dessen Existenz in Theorem 8 vorausgesetzt wird. In diesem gilt nach Lemma 10: $\hat{g}(F(e)) = \hat{g}(F(e^\omega)) = \hat{g}(e)^\omega = \hat{g}(e)^\omega \hat{g}(e)^\omega = \hat{g}(F(ee))$ für alle e^ω in l und r, und nach Lemma 11: $\hat{g}(F(l)) = g(l) \neq g(r) = \hat{g}(F(r))$.

($\leftarrow$) $\hat{g}$ hat bereits die gewünschte Eigenschaft: Aus $\hat{g}(F(e)) = \hat{g}(F(ee))$ folgt $\hat{g}(F(e)) = \hat{g}(F(e))^\omega$, womit, wie in Lemmata 10 und 11, gezeigt werden kann, dass $\hat{g}(l) = \hat{g}(F(l)) \neq \hat{g}(F(r)) = \hat{g}(r)$. □

Der Test von $\hat{g}(e) = \hat{g}(ee)$ in Gleichung 4.2 beinhalten eine Allquantifikation über alle Zustände. Es genügt jedoch zu überprüfen, dass $q\,\hat{g}(e)^{\mathcal{A}} = q\,\hat{g}(ee)^{\mathcal{A}}$ für alle q, an denen $g(e)$ auch ausgewertet wird. Wir definieren diese Überprüfung nun rekursiv für alle Omega-Terme:

Definition 3.

i. Wenn $e = 1$, dann $C_{q,e}(g) = \top$

ii. Wenn $e \in X$, dann $C_{q,e}(g) = \top$

iii. Wenn $e = lr$, dann $C_{q,e}(g) = C_{q,l}(g) \wedge C_{q\,g(F(l))^{\mathcal{A}},r}(g)$

iv. Wenn $e = l^\omega$, dann $C_{q,e}(g) = (q\,g(F(l))^{\mathcal{A}} = q\,g(F(l))^{2\mathcal{A}}) \wedge C_{q,l}(g) \wedge C_{q\,g(F(l))^{\mathcal{A}},l}(g)$

Lemma 13. $C_{q,e}(g) \rightarrow q\,g(F(e))^{\mathcal{A}} = q\,g(e)^{\mathcal{A}}$

Beweis. Durch Induktion über den Aufbau von e.

Wenn $e = 1$ oder $e \in X$, dann $e = F(e)$, also $q\, g(F(e))^{\mathcal{A}} = q\, g(e)^{\mathcal{A}}$.

Wenn $e = lr$, dann folgt aus $C_{q,e}(g)$ nach Induktionsannahme $q\, g(F(l))^{\mathcal{A}} = q\, g(l)^{\mathcal{A}}$ und $q\, g(F(l))^{\mathcal{A}}\, g(F(r))^{\mathcal{A}} = q\, g(F(l))^{\mathcal{A}}\, g(r)^{\mathcal{A}}$. Also $q\, g(F(e))^{\mathcal{A}} = q\, g(F(l))^{\mathcal{A}}\, g(F(r))^{\mathcal{A}} = q\, g(F(l))^{\mathcal{A}}\, g(r)^{\mathcal{A}} = q\, g(l)^{\mathcal{A}}\, g(r)^{\mathcal{A}} = q\, g(e)^{\mathcal{A}}$

Wenn $e = l^\omega$, dann gilt $q\, g(F(l))^{\mathcal{A}} = q\, g(F(l))^{2\mathcal{A}}$ und mit $q\, g(F(l))^{\mathcal{A}} = q\, g(l)^{\mathcal{A}}$ und $q\, g(F(l))^{\mathcal{A}}\, g(F(l))^{\mathcal{A}} = q\, g(F(l))^{\mathcal{A}}\, g(l)^{\mathcal{A}}$ aus Induktionsannahme gilt $q\, g(F(l))^{\mathcal{A}} = q\, g(l)^{\mathcal{A}} = q\, g(l)^{2\mathcal{A}} = \cdots = q\, g(l)^{\omega\mathcal{A}} = q\, g(l^\omega)^{\mathcal{A}}$. □

Wir können nun eine in NL entscheidbare Charakterisierung geben:

Lemma 14. *Für alle l, r, die die Bedingung aus Theorem 8 erfüllen, sind die folgenden Bedingungen äquivalent*

$$L(\mathcal{A}) \notin [\![l = r]\!] \tag{4.3}$$

$$\exists\, g \in X \to M(\mathcal{A}), p, s \in \Sigma^\star : q_0\, p^{\mathcal{A}}\, g(l)\, s^{\mathcal{A}} \in F \not\leftrightarrow q_0\, p^{\mathcal{A}}\, g(r)\, s^{\mathcal{A}} \in F \tag{4.4}$$

$$\begin{aligned}\exists\, \hat{g} \in X \to M(\mathcal{A}), p, s \in \Sigma^\star : & C_{q_0\, p^{\mathcal{A}}, l}(\hat{g}) \wedge C_{q_0\, p^{\mathcal{A}}, r}(\hat{g}) \\ & \wedge\, q_0\, p^{\mathcal{A}}\, \hat{g}(F(l))\, s^{\mathcal{A}} \in F \not\leftrightarrow q_0\, p^{\mathcal{A}}\, \hat{g}(F(r))\, s^{\mathcal{A}} \in F\end{aligned} \tag{4.5}$$

Beweis. (4.3)↔(4.4) ist Lemma 2.

(4.4)→(4.5) Zu jedem g gibt es jenes $\hat{g}$, dessen Existenz in Theorem 8 vorausgesetzt wird. In diesem gilt, wie bereits festgestellt, $\hat{g}(F(e)) = \hat{g}(F(ee))$ für alle e^ω in l und r und, da $C_{q_0\, p^{\mathcal{A}}, l}(\hat{g})$ und $C_{q_0\, p^{\mathcal{A}}, l}(\hat{g})$ nur aus Gleichungen der Form $q\, g(F(e))^{\mathcal{A}} = q\, g(F(ee))^{\mathcal{A}}$ bestehen, gelten diese auch. Außerdem gilt nach Lemma 11: $\hat{g}(F(l)) = g(l)$ und $g(r) = \hat{g}(F(r))$ und damit folgt (4.5) aus (4.4).

(4.4)←(4.5) $\hat{g}$ hat bereits die gewünschte Eigenschaft, da aus Lemma 13 folgt, dass $q_0\, p^{\mathcal{A}}\, \hat{g}(F(l)) = q_0\, p^{\mathcal{A}}\, \hat{g}(l)$ und $q_0\, p^{\mathcal{A}}\, \hat{g}(F(r)) = q_0\, p^{\mathcal{A}}\, \hat{g}(r)$. □

Aus Bedingung 4.5 lässt sich ein NL-Algorithmus ableiten: Man sieht, dass diese Bedingung nur Berechnungen von $q_0\, p^{\mathcal{A}}\, g(F(t))$ enthält. Jedes $F(t)$ entspricht einem Omega-Term $x_1 x_2 \cdots x_n$, $x_i \in X$. Bedingung 4.5 als Ganzes entspricht also einem Verbotsmuster, und kann folgendermaßen geprüft werden: Beim Berechnen von $q_0\, p^{\mathcal{A}}\, g(x_1 x_2 \cdots x_n)$ werden die Elemente des Transitionsmonoiden nur auf eine konstante Anzahl an Zuständen, nämlich $q_1 = q_0\, p^{\mathcal{A}}, q_2 = q_0\, p^{\mathcal{A}}\, g(x_1), q_3 = q_0\, p^{\mathcal{A}}\, g(x_1)\, g(x_2), \ldots$, angewendet. Der Algorithmus kann nun als erstes diese Zustände raten, und rät dann, anstatt der Werte $g'(x)^{\mathcal{A}}$ des Transitionsmonoids, direkt alle benötigten $q_i\, g'(x_i)^{\mathcal{A}}$. Da die Menge der benötigten Zustände konstant ist, und jeder Zustand in logarithmischem Platz gespeichert werden kann, ist dies in NL möglich.

Lemma 15. *Das Entscheidungsproblem, ob ein Automat eine Sprache der Klasse* piecewise testable *erkennt, ist NL-vollständig*

Beweis. Die Klasse der *piecewise testable* Sprachen kann als $[\![(ab)^\omega = (ab)^\omega a, (ab)^\omega = b(ab)^\omega]\!]$ charakterisiert werden. Beide Gleichungen erfüllen die Bedingung aus Theorem 8: Für $(ab)^\omega = (ab)^\omega a$ wähle $\hat{g}(a) = g(a)$ und $\hat{g}(b) = g(b)g(ab)^{\omega-1}$, für $(ab)^\omega = b(ab)^\omega$ wähle $\hat{g}(a) = g(ab)^{\omega-1}g(a)$ und $\hat{g}(b) = g(b)$. Damit existiert nach Theorem 8 ein NL-Algorithmus und das Problem ist, da es nach Theorem 6 auch NL-schwer ist, NL-vollständig. □

Lemma 16. *Das Entscheidungsproblem, ob ein Automat eine Sprache der Klasse* dot-depth one *erkennt, ist NL-vollständig*

Beweis. In [Pin] werden die Sprachen der Klasse *dot-depth one* als

$$[\![(e^\omega a f^\omega b)^\omega e^\omega (x f^\omega y e^\omega)^\omega (e^\omega a f^\omega b)^\omega e^\omega a f^\omega y e^\omega (x f^\omega y e^\omega)^\omega]\!]$$

charakterisiert. Diese Gleichung erfüllt die Bedingung aus Theorem 8: Wähle $\hat{g}(e) = g(e)^\omega$, $\hat{g}(f) = g(f)^\omega$, $\hat{g}(a) = g(a)$, $\hat{g}(b) = g(b)g(e^\omega a f^\omega b)^{\omega-1}$, $\hat{g}(x) = g(x f^\omega y e^\omega)^{\omega-1}g(x)$ und $\hat{g}(y) = g(y)$. Daraus folgt mit Theorem 8 und 6 die NL-Vollständigkeit. □

Entsprechend lässt sich die NL-Vollständigkeit von $\mathcal{DA} = [\![(xy)^\omega x(xy)^\omega = (xy)^\omega]\!]$ oder Gruppen $[\![x^\omega = 1]\!]$ zeigen.

5 Der PSPACE-Schwere-Beweis für sternfreie Sprachen

In diesem Kapitel untersuchen wir den PSPACE-Schwere-Beweis für sternfreie Sprachen aus [CH91].

Theorem 17. *(Cho und Huynh, 1991) Für jedes Alphabet Σ, das mindestens zwei Zeichen enthält, gilt: Das Entscheidungsproblem „gegeben ein DFA $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$: Gehört die von $\mathcal{A}$ akzeptierte Sprache $L(\mathcal{A})$ zu den sternfreien Sprachen" ist PSPACE-schwer bezüglich log-space-Reduktion.*

Die sternfreien Sprachen sind der Abschluss der endlichen Sprachen unter Produkt und booleschen Mengenoperationen. Der Unterschied zu den regulären Sprachen ist also das Fehlen des Abschluss unter Stern. Schützenberger gibt als Charakterisierung über das syntaktisches Monoid die Gleichung $[\![x^\omega = x^\omega x]\!]$.

Der Beweis von Cho und Huynh baut auf Kozens Beweis [Koz77] für die PSPACE-Schwere des Schnittproblems für DFAs auf („gegeben eine Menge $\mathbf{A}$ von DFAs: Ist die Sprache $\bigcap_{\mathcal{A}\in\mathbf{A}} L(\mathcal{A})$ der Worte, die von allen Automaten $\mathcal{A} \in \mathbf{A}$ akzeptiert werden, leer). Es ist nicht direkt möglich das Schnittproblem auf sternfreie Sprachen zu reduzieren, da wir sicherstellen müssen, dass alle Automaten in $\mathbf{A}$ selbst sternfrei sind. Wir wandeln daher Kozens Reduktion so ab, dass nur Automaten erzeugt werden, die Sprachen der Klasse *locally testable* erkennen. Damit zeigen wir, dass das Schnittproblem bereits für Automaten der Klasse *locally testable* – und aller Oberklassen – PSPACE-vollständig ist, und können, a *locally testable* eine echte Unterklasse der sternfreien Sprachen ist [BS71], die PSPACE-Vollständigkeit der Erkennung sternfreier Sprachen durch Reduktion des Schnittproblems sternfreier Sprachen beweisen.

Eine Sprache L über Σ ist *strictly k-testable*, wenn alle Worte, die länger sind als k Zeichen sind, durch drei Mengen $P, F, S \in \Sigma^k$ der zulässigen Präfixe, Suffixe und echten Faktoren definiert werden. Es gilt dann also $L = (P\Sigma^\star \cap \Sigma^\star S \cap \overline{\Sigma\Sigma^\star(\Sigma^k \setminus F)\Sigma\Sigma^\star}) \cup E$, wobei E die endliche Sprache der Worte bereitstellt, die kürzer als k Zeichen sind [Zal72]. Eine Sprache ist *strictly locally testable*, wenn sie *strictly k-testable* ist für irgendein k. *locally testable* ist der Abschluss von *strictly locally testable* unter booleschen Mengenoperationen. Die Zugehörigkeit einer Sprache zu *strictly locally testable* und damit zu *locally testable*, lässt sich also durch Angabe der drei Mengen P, F, S zeigen.

Lemma 18. *Für endliche Automaten, die Sprachen der Klasse* locally testable *akzeptieren, ist das Schnittproblem PSPACE-vollständig.*

Sei M eine beliebige polynomiell platzbeschränkte deterministische Turingmaschine und p ein Polynom, so dass M auf einer Eingabe der Länge n höchstens $p(n)$ Felder des Bandes verwendet. Sei

Q die Menge der Zustände von M, $q_0 \in Q$ der Startzustand, Γ ihr Bandalphabet, $\bar{b}$ das Blank und $\delta \in Q \times \Gamma \cup \{\bar{b}\} \to Q \times \Gamma \times \{L, 0, R\}$ ihre Überführungsfunktion. Wir dürfen voraussetzen, dass M eine einseitige Turingmaschine ist und nach einer akzeptierenden Rechnung auf dem linken Ende des leeren Bands im Zustand q_{acc} steht. Wir nehmen an dass Q, Γ und $\{\#\}$ paarweise disjunkt sind und verwenden das Alphabet $\Sigma = Q \cup \Gamma \cup \{\bar{b}, \#\}$. Sei $x \in \Gamma^\star$ eine beliebige Eingabe für M.

Eine *Konfiguration* (*instantaneous description*) codiert den aktuellen Zustand von M, den Inhalt des Bandes und die Position des Schreib-Lese-Kopfs als Wort aus $\Gamma^\star Q \Gamma^\star$. Der Schreib-Lese-Kopf steht dabei auf dem ersten Zeichen rechts des Zustands. Da M mit $p(n)$ platzbeschränkt ist, kann sich der Schreib-Lese-Kopf während einer Rechnung von M nicht mehr als $p(n)$ Felder von der Startposition fortbewegen und wir können alle Konfigurationen, die während der Rechnung von M auf Eingabe x vorkommen, als Worte aus $(\Gamma \cup Q \cup \{\bar{b}\})^{p(|x|)+1}$ darstellen, indem kürzere Konfigurationen durch Blanks aufgefüllt werden. Wir definieren $\Delta = \Gamma \cup Q \cup \{\bar{b}\}$, da diese Menge im Folgenden häufig verwendet wird. Eine Berechnung C ist eine Folge von durch $\#$ getrennten Konfigurationen $\#ID_0\#ID_1\# \cdots \#ID_{k-1}\#$, so dass M nach ihrer Überführungsfunktion aus der Konfiguration ID_i in ID_{i+1} übergeht. C ist eine akzeptierende Berechnung auf Eingabe x, wenn $ID_0 = q_0 x \bar{b}^{p(|x|)-|x|}$ die Startkonfiguration und die letzte Konfiguration ID_{k-1} eine akzeptierende Konfiguration kodiert.

Wir geben nun nach Kozen die Reduktion eines Wortes x auf eine Menge von Automaten, so dass der Schnitt dieser Automaten genau dann nicht leer ist, wenn M das Wort x akzeptiert. M und p sind hier beliebig aber fest.

Der Automat $\mathcal{A}_{\text{ends}}$ akzeptiert alle Berechnungen in denen die erste ID der Starkonfiguration und die letzte ID der Endkonfiguration entspricht, d. h.

$$L(\mathcal{A}_{\text{ends}}) = \#q_0 x \bar{b}^{p(|x|)-|x|}\#\{\Delta \cup \#\}^\star \#q_{\text{acc}}\bar{b}^p(|x|)\#$$

Diese Sprache ist *strictly* $p(|x|) + 3$*-testable* mit

$$\begin{aligned} P &= \{\#q_0 x \bar{b}^{p(|x|)-|x|}\#\} \\ S &= \{\#q_{\text{acc}}\bar{b}^p(|x|)\#\} \\ F &= \Sigma^{p(|x|)+3} \end{aligned}$$

Die Konstruktion von $\mathcal{A}_{\text{ends}}$ gegeben x bereitet keine Schwierigkeiten.

Die Automaten $\mathcal{A}_i$, $1 \leq i \leq p(|x|)$ prüfen, dass das i-te Zeichen jeder ID mit der Überführungsfunktion von M aus der vorhergehenden ID folgt. Das heißt, für jeden Faktor $abc\Delta^{p(|x|)-i}\#\Delta^{i-1}b'$ mit $a, b, c, b' \in \Delta$ muss b' das Zeichen sein, das nach der Überführungsfunktion von M aus a, b, c folgt. Kozen fordert hier Rechnungen gerader Länge und gibt zwei Automaten $\mathcal{A}_i^{\text{even}}$ und $\mathcal{A}_i^{\text{odd}}$, die dies jeweils für alle Paare $\#ID_{2j}\#ID_{2j+1}$ und $\#ID_{2j+1}\#ID_{2j+2}$ prüfen. Diese Automaten sind jedoch nicht sternfrei: Bei einer ID w mit $a, b, c \in \Gamma$ ist $b' = b$ (Bandfelder, an denen der Schreib-Lese-Kopf nicht steht, dürfen sich nicht ändern), und der Zyklus $(w\#w\#)$ zeigt, dass $L(\mathcal{A}_i^{\text{even}})$ nicht sternfrei ist.

Wir behaupten, dass eine Logspace-Reduktion auch einen Automaten $\mathcal{A}_i$ erzeugen kann, der die Sprache $(L(\mathcal{A}_i^{\text{even}}) \cap \#\Delta^{p(|x|)+1}L(\mathcal{A}_i^{\text{even}}))\Delta^{p(|x|)+1}\# \cup (\#\Delta^{p(|x|)+1}L(\mathcal{A}_i^{\text{even}}) \cap L(\mathcal{A}_i^{\text{even}})\Delta^{p(|x|)+1}\#)$ akzeptiert, und damit alle Übergänge $ID_j\#ID_{j+1}$ in Rechnungen beliebiger Länge prüft. Dass dies möglich ist, sieht man daran, dass die Struktur des Automaten nur von M abhängt und in der

Überführungsfunktion der reduzierenden Turingmaschine gespeichert wird. Die Eingabe x bestimmt lediglich die Länge der Ketten $q\,\Delta^{p(|x|)-i}\#\Delta^{i-2^{\mathcal{A}}}\,q'$ von Zuständen, die den Automat an die gleiche Position in der nächsten Konfiguration überführen.

Dass $L(\mathcal{A}_i)$ *strictly* $p(n)+4$*-testable* ist, zeigen die Test-Sets

$$\begin{aligned}
P = \quad & \#\Delta^{p(|x|)+1}\#\Delta \\
S = \quad & \Delta\#\Delta^{p(|x|)+1}\# \\
F = \quad & \bigcup_{a,b,c,b' \text{ nach Überführungsfunktion}} abc\Delta^{p(|x|)-i}\#\Delta^{i-1}b' \\
& \cup \bigcup_{j\neq i} \Delta^{p(|x|)-i+3}\#\Delta^{i} \\
& \cup P \cup S
\end{aligned}$$

die sicherstellen, dass das i-te Zeichen jeder Konfiguration aus der Vorgängerkonfiguration folgt, und dass jedes $p(|x|)+2$-te Zeichen das Trennzeichen $\#$ ist.

Nun gilt wie in [Koz77]: $w \in \bigcup_{0\leq i\leq p(|x|)+1} L(\mathcal{A}_i)$ g. d. w. M die Eingabe x mit Rechnung w akzeptiert. Da alle $\mathcal{A}_i$ *strictly* $p(|x|)+4$*-testable*, und damit auch *locally testable*, sind, ist das Schnittproblem für *locally testable* PSPACE-Schwer. Es ist als Sonderfall des allgemeinen Schnittproblems auch in PSPACE entscheidbar und somit folgt Lemma 18.

Wir reduzieren nun das Schnittproblem für sternfreie Sprachen auf die Erkennung sternfreier Sprachen

Sei $\mathbf{A}$ eine Instanz des Schnittproblems für sternfreie Sprachen, d. h. $\mathbf{A} = \mathcal{A}'_1, \mathcal{A}'_2, \ldots, \mathcal{A}'_n$ ist eine Menge von Automaten mit Alphabet Σ', die jeweils eine sternfreie Spache akzeptieren. Wir konstruieren nun einen Automaten $\mathcal{A}$ der genau dann eine sternfreie Sprache akzeptiert, wenn $\bigcup \mathcal{A}'_i = \emptyset$.

Wir können annehmen, dass die Anzahl $n = |\mathbf{A}|$ der Automaten eine Primzahl ist. Anderenfalls können solange Kopien der Automaten aus $\mathcal{A}_i$ hinzugefügt werden bis die Anzahl prim ist. Da es nach Betrands Postulat eine Primzahl $n \leq p < 2n$ gibt, sind alle Zahlen, die beim Suchen von p und beim Suchen von Teilern der Primzahlkandidaten verwendet werden, in $\lceil \log 2n \rceil$ Bit speicherbar.

Der gesuchte Automat $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$ verbindet nun alle $\mathcal{A}'_i = (Q_i, \Sigma', q_{0,i}, F_i, \delta_i)$ des Schnittproblems zu einem Zyklus. Wir nehmen ohne Beschränkung der Allgemeinheit an, dass alle Q_i paarweise disjunkt sind, verwenden ein Trennzeichen $\# \notin \Sigma'$ und definieren:

- Die Menge der Zustände $Q = \{\bot\} \cup \bigcup Q_i$ sind die Zustände aller Automaten $\mathcal{A}'_i$ und ein toter Zustand.
- $\Sigma = \Sigma' \cup \#$
- $q_0 = q_{0,1}$
- $F = \{q_0\}$
- Die Überführungsfunktion δ enthält die Überführungsfunktionen aller Teilautomaten und verbindet ihre Endzustände mit dem Startzustand des jeweils nächsten Automaten:

- $\delta(q, a) = \delta_i(q, a)$ für alle $q \in Q_i$, $1 \leq i \leq n$ und $a \in \Sigma'$.
- $\delta(q, \#) = q_{0,i+1}$ für alle $q \in F_i$, $1 \leq i < n$.
- $\delta(q, \#) = q_{0,1}$ für alle $q \in F_n$.
- $\delta(q, a) = \perp$ sonst.

Lemma 19. *Der Automat $\mathcal{A}$ akzeptiert genau dann eine sternfreie Sprachen, wenn der Schnitt der Automaten $\mathcal{A}'_i$ leer ist.*

Mit Lemma 2 erhalten wir aus $[\![x^\omega = x^\omega x]\!]$ folgende Charakterisierung der sternfreien Sprachen (wir schreiben, da die Variablenmenge X jetzt fest ist, x statt $g(x)$):

$$\exists\, x, p, s \in \Sigma^\star : q_0\, px^{\omega\mathcal{A}}\, s^{\mathcal{A}} \in F \not\leftrightarrow q_0\, px^\omega x^{\mathcal{A}}\, s^{\mathcal{A}} \in F \tag{5.1}$$

Beweis. ($\rightarrow$) Wenn der Schnitt nicht leer ist, gibt es ein Wort w, das von allen Automaten $\mathcal{A}'_i$ akzeptiert wird. $\mathcal{A}$ ist so konstruiert, dass $q_{0,i}\, w\#^{\mathcal{A}} = q_{0,i+1}$ für $1 \leq i < n$ und $q_{0,n}\, w\#^{\mathcal{A}} = q_{0,1}$, und somit auch $q_{0,1} = q_{0,1}\, (w\#)^{n\mathcal{A}}\, q_{0,1} = q_{0,1}\, (w\#)^{2n\mathcal{A}}\, q_{0,1} = q_{0,1}\, (w\#)^{\omega\mathcal{A}}\, q_{0,1}$. Wähle nun $p = \epsilon$, $s = \epsilon$, $x = w\#$: Dann ist $q_{0,1}\, px^\omega s^{\mathcal{A}} = q_{0,1}\, (w\#)^{\omega\mathcal{A}} = q_{0,1} \in F$ und $q_{0,1}\, px^\omega xs^{\mathcal{A}} = q_{0,1}\, (w\#)^\omega w\#^{\mathcal{A}} = q_{0,1}\, w\#^{\mathcal{A}} = q_{0,2} \notin F$, und damit nach Gleichung 5.1 $L(\mathcal{A}) \notin [\![x^\omega = x^\omega x]\!]\, (\Sigma^\star)$.

($\leftarrow$) Wenn $L(\mathcal{A}) \notin [\![x^\omega = x^\omega x]\!]\, (\Sigma)$, dann gibt es x, p, s, die die Gleichung 5.1 erfüllen.

Lemma 20. *Beim Lesen von $px^\omega x^l$ für beliebige l wird nie der tote Zustand erreicht.*

Beweis. Sonst wäre, da dieser nicht mehr verlassen werden kann, $q_{0,1}\, px^\omega x^{l\mathcal{A}} = q_{0,1}\, px^\omega x^l x^{\omega-l\mathcal{A}} = q_{0,1}\, px^{\omega\mathcal{A}} = q_{0,1}\, px^\omega x^{\mathcal{A}} = \perp$ und es gäbe kein s, so dass $q_{0,1}\, px^\omega s^{\mathcal{A}} \in F \not\leftrightarrow q_{0,1}\, px^\omega xs^{\mathcal{A}} \in F$. □

Lemma 21. *In dem Wort x ist das Zeichen $\#$ enthalten.*

Beweis. Anderenfalls würde sich der komplette Pfad von $q_{0,1}\, p^{\mathcal{A}}$ bis $q_{0,1}\, px^\omega x^{\mathcal{A}}$ innerhalb des gleichen Teilautomaten $\mathcal{A}'_i$ befinden, womit gezeigt wäre, dass dieser keine sternfreie Sprache akzeptiert, was im Wiederspruch zur Annahme steht. □

Das Wort x enthält also $1 \leq k$ mal das Trennzeichen $\#$ und hat die Form $u\#v_1\# \cdots \#v_{k-1}\#w$, $u, v_i, w \in \Sigma'^\star$.

Lemma 22. *Die Anzahl k der Trennzeichen $\#$ in x ist kein Vielfaches von n.*

Beweis. Wir zeigen: Wenn k ein Vielfaches von n ist, dann ist für alle Zustände q: $q\, xx^{\mathcal{A}} = q\, xxx^{\mathcal{A}} = q\, x^{\omega\mathcal{A}}$ im Wiederspruch zur Annahme, dass $q_{0,1}\, px^{\omega\mathcal{A}} \neq q_{0,1}\, px^\omega x^{\mathcal{A}}$.

Wenn $q\, xx^{\mathcal{A}} = \perp$, dann ist auch $q\, xxx^{\mathcal{A}} = \ldots = q\, x^{\omega\mathcal{A}} = \perp$ und wir sind fertig. Anderenfalls gilt: Der Zustand q gehört zu den Zuständen Q_i einer der Teilautomaten der $\mathcal{A}'_i$. Da das Trennzeichen $\#$ nur beim Übergang in den Startzustand des nächsten Teilautomat vorkommt, muss gelten: $q\, u\#^{\mathcal{A}} = q_{0,i+1 \bmod n}$, $q\, u\#v_1\#^{\mathcal{A}} = q_{0,i+2 \bmod n}$, ..., $q\, u\#v_1\# \cdots \#v_{n-1}\#^{\mathcal{A}} = q_{0,i+n \bmod n} = q_{0,i}$, und damit $q\, x^{\mathcal{A}} = q_{0,i}\, w^{\mathcal{A}} \in Q_i$. Damit gilt gleichermaßen $q\, xu\#^{\mathcal{A}} = q_{0,i+1 \bmod n}$, ...,

$q\,xu\#v_1\#\cdots\#v_{n-1}\#^{\mathcal{A}} = q_{0,i+n \bmod n} = q_{0,i}$, und somit $q\,xx^{\mathcal{A}} = q_{0,i}\,w^{\mathcal{A}} = q\,x^{\mathcal{A}}$. Für den Fall $q\,xx^{\mathcal{A}} \neq \perp$ gilt also ebenfalls $q\,x^{\mathcal{A}} = q\,xx^{\mathcal{A}} = \cdots = q\,x^{\omega\mathcal{A}}$. □

Lemma 23. *Wenn $x^e = x^\omega$, dann ist der Exponent e ein Vielfaches von n.*

Beweis. Da $q_0\,px^{\omega\mathcal{A}} = q_0\,px^\omega x^{\omega\mathcal{A}}$ trivialerweise zu dem gleichen Teilautomat $\mathcal{A}'_i$ gehören und x das Trennzeichen $\#$ enthält, muss aufgrund des Aufbaus von $\mathcal{A}$ die Anzahl der Vorkommen von $\#$ in $x^\omega = x^e$ ein Vielfaches von n sein. Da die Anzahl der Vorkommen von $\#$ in x nach Lemma 22 kein Vielfaches von n ist, muss der Exponent e eines sein. □

Lemma 24. *Für $x = u\#v_1\#\cdots\#v_{k-1}\#w$, $u, v_i, w \in \Sigma'^\star$ gilt: Die Worte $wu, v_1, \ldots, v_{k-1}$ werden von allen Teilautomaten $\mathcal{A}'_j$, $1 \leq j \leq n$ akzeptiert.*

Beweis. Wir zeigen dies für das Wort wu. Sei wieder $q = q_0\,px^{\omega\mathcal{A}} \in Q_i$. Durch Zählen der Trennzeichen folgt $q_l = q_0\,px^\omega x^{l\mathcal{A}} \in Q_{i+lk \bmod n}$. Damit gilt $q_l\,u\#v_1\#\cdots\#v_{k-1}\#^{\mathcal{A}} = q_{0,i+lk+k \bmod n}$ und $q_l\,u\#v_1\#\cdots\#v_{k-1}\#wu\#^{\mathcal{A}} = q_{0,i+lk+k+1 \bmod n}$, also ist $q_l\,u\#v_1\#\cdots\#v_{k-1}\#wu^{\mathcal{A}} \in F_{i+lk+k \bmod n}$ und es gilt für alle l: $\mathcal{A}'_{i+(l+1)k \bmod n}$ akzeptiert wu.

Da n prim ist und k nach Lemma 22 kein Vielfaches von n sein kann, also n und k teilerfremd sind, gibt es für jedes $1 \leq j \leq n$ ein l, so dass $j = i + (l+1)k \bmod n$, und somit akzeptieren alle Automaten $\mathcal{A}'_j$ das Wort wu. □

Damit ist Lemma 19 bewiesen. □

Wir haben hiermit also eine Reduktion von PSPACE über Schnitt sternfreier Sprachen auf die Erkennung sternfreier Sprachen mit variablem Alphabet erreicht. Für die Reduktion auf ein beliebiges festes Alphabet mit mindestens zwei Zeichen müssen die Zeichen des variablen Alphabets geeignet codiert werden.

Wie in [CH91] dargelegt wurde, kann eine normale Binärcodierung dazu führen, dass aus einer sternfreien Sprache eine nicht sternfreie wird. So ist zum Beispiel die Sprache $(ab)^\star$ sternfrei, wenn aber a mit 101 und b mit 010 codiert wird, enthält die resultierende Sprache $(101010)^\star$ den nichttrivialen Zyklus $(10)^3$ und ist somit nicht mehr sternfrei. Dieses Problem lässt sich lößen, indem eine Codierung verwendet wird, die die ursprünglichen Zeichen eindeutig voneinander abgrenzt, zum Beispiel durch ein Präfix 0011 gefolgt von der Binärcodierung des Zeichens mit den Ziffern 01 und 00. Dass eine derartige Codierung die gewünschte Reduktion ermöglicht, ist für den Fall der sternfreien Sprachen leicht zu zeigen. Wir wissen aber andererseits, dass es für die *piecewise testable* Sprachen keine geeignete Codierung gibt, da jede Codierng eine Schleife in einen Zyklus verwandelt. Ob diese Reduktion bei anderen Omega-Gleichungen, die einen PSPACE-Schwere-Beweis für variables Alphabet zulassen, funktionieren würde, ist offen.

6 Zusammenfassung

Der PSPACE-Algorithmus für das Erkennen sternfreier Sprachen aus [Ste85b] und der NL-Schwere-Beweis für *piecewise testable* aus [CH91] konnten durch naheliegende Modifikationen auf beliebige Omega-Gleichungen generalisiert werden.

Wir konnten in Kapitel 4 eine Bedingung angeben unter der systematisch ein Entscheidungsverfahren in NL konstruiert werden kann. Damit lässt sich die Komplexität mancher Varietäten, wie z. B. $\mathbf{DA}$ und $\mathbf{G}$, als NL-vollständig bestimmen. Da es nicht gelungen ist eine notwendige Bedingung für die Entscheidbarkeit einer Gleichung in NL zu geben, gewinnen wir in Fällen, in denen die Bedingung nicht zutrifft, leider keine Erkenntnisse über die Komplexität der Klasse.

Indem wir zeigen, dass das Schnittproblem für DFAs auch dann PSPACE-schwer ist, wenn die Automaten Sprachen aus der Klasse *locally testable* akzeptieren, zeigen wir die PSPACE-Schwere eines Problems, das sich gut auf DFA Aperiodizität reduzieren lässt, und daher möglicherweise auch gut dazu geeignet ist, die PSPACE-Schwere anderer Sprachklassen zu zeigen.

Die Fragen, ob es Mengen von Gleichung gibt, die weder NL- noch PSPACE-vollständig sind ist noch offen.

Literaturverzeichnis

[BS71] J. A. Brzozowski, I. Simon. Characterizations of Locally Testable Events. In *SWAT (FOCS)*, S. 166–176. IEEE Computer Society, 1971.

[CB71] R. S. Cohen, J. A. Brzozowski. Dot-Depth of Star-Free Events. *J. Comput. Syst. Sci.*, 5(1):1–16, 1971. URL http://dblp.uni-trier.de/db/journals/jcss/jcss5.html#CohenB71.

[CH91] S. Cho, D. T. Huynh. Finite-automaton aperiodicity is PSPACE-complete. *Theoretical Computer Science*, 88(1):99 – 116, 1991. doi:http://dx.doi.org/10.1016/0304-3975(91)90075-D. URL http://www.sciencedirect.com/science/article/pii/030439759190075D.

[HU79] J. E. Hopcroft, J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Publishing Company, 1979.

[Koz77] D. Kozen. Lower bounds for natural proof systems. In *Proc. 18th Symp. Found. Comput. Sci.*, S. 254–266. IEEE, 1977.

[Pin] J.-É. Pin. *Mathematical Foundation of Automata Theory.* URL http://www.liafa.jussieu.fr/~jep/PDF/MPRI/MPRI.pdf. Version of August 23, 2013.

[Sch01] U. Schöning. *Theoretische Informatik - kurzgefaßt (4. Aufl.).* Hochschultaschenbuch. Spektrum Akademischer Verlag, 2001.

[Sim75] I. Simon. Piecewise testable events. In H. Barkhage, Herausgeber, *Automata Theory and Formal Languages*, Band 33 von *Lecture Notes in Computer Science*, S. 214–222. Springer, 1975. URL http://dblp.uni-trier.de/db/conf/automata/automata1975.html#Simon75.

[Ste85a] J. Stern. Characterizations of some classes of regular events. *Theoretical Computer Science*, 35(0):17 – 42, 1985. doi:http://dx.doi.org/10.1016/0304-3975(85)90003-9. URL http://www.sciencedirect.com/science/article/pii/0304397585900039.

[Ste85b] J. Stern. Complexity of some problems from the theory of automata. *Information and Control*, 66(3):163 – 176, 1985. doi:http://dx.doi.org/10.1016/S0019-9958(85)80058-9. URL http://www.sciencedirect.com/science/article/pii/S0019995885800589.

[Zal72] Y. Zalcstein. Locally Testable Languages. *J. Comput. Syst. Sci.*, 6(2):151–167, 1972.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift