Seminar - Automatentheorie

Äquivalenztest für eindeutige NEAs

Jens Helge Reelfs (Matr.-Nr 282953)

17. Januar 2012

1 Einleitung

Das Inklusionsproblem sowie das Äquivalenzproblem von regulären Sprachen, regulären Grammatiken und endlichlichen Automaten ([5, Theorem 2.47]) sind fundamentale Fragen in der Informatik. Diese Probleme haben sich als PSPACE vollständig erwiesen. Das heißt, dass sie nach heutigen Kenntnissen algorithmisch nicht praktisch entscheidbar sind.
Es gibt viele Ansätze, die durch verschiedene Voraussetzungen an der Struktur der Sprache, Grammatik oder Auomaten versuchen diese Probleme effizient zu lösen, doch umfassen diese immer Familien von echten regulären Teilmengen.

Die Hauptquelle dieser Seminararbeit [4] betrachtet die strukturelle Voraussetzung der *Eindeutigkeit* (die Eindeutigkeit beschreibt die Anzahl verschiedener Läufe von einem Automaten auf akzeptierten Wörtern bzw. Anzahl der Ableitungen einer Grammatik zu einem Wort). Diese Restriktion erscheint sehr natürlich, da durch die Semantik hinter Sprachen und Grammatiken diese häufig eindeutig sind. So sind bspw. alle $LR(0)$ Grammatiken, wie sie im Compilerbau vorkommen, immer eindeutig [3, 10.6 *LR(0)* Grammars].

Es wird gezeigt, dass das Inklusions- und Äquivalenzproblem eindeutiger Automaten effizient entscheidbar ist. Diese Ergebnisse werden dann auf Uneindeutigkeit eines fest beschränkten Grades erweitert. Die Resultate beziehen sich sowohl auf reguläre Sprachen und Grammatiken wie auch Automaten. Da der Umfang dieser Arbeit beschränkt ist, werden hier die Beweisführungen zu Automaten im Detail aufgegriffen. Diese lassen sich entsprechend auf reguläre Sprachen und Grammatiken übertragen.

2 Definitionen und Notation

Definition 1. Ein nichtdeterministischer endlicher Automat (ϵNEA) wird beschrieben durch ein 5-Tupel $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ mit

- einer nichtleeren endlichen Menge von Zuständen Q,
- einer nichtleeren endliche Menge von Eingabebuchstaben (Alphabet) Σ,
- einem Startzustand $q_0 \in Q$,
- der Menge der akzeptierenden Zustände $F \subseteq Q$ und
- der Transitionsfunktion $\Delta : Q \times (\Sigma \cup \{\epsilon\}) \to 2^Q$.

Besitzt $\mathcal{A}$ mindestens eine ϵ-Transition, d.h. $\exists\ q \in Q : \Delta(q, \epsilon) \neq \emptyset$, so handelt es sich um einen ϵ-NEA. Andernfalls sprechen wir von einem NEA **ohne** ϵ-Transitionen.

Definition 2. Ein Lauf von einem NEA $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ auf dem Wort $w = a_0 \cdots a_{n-1} \in \Sigma^*$ ist eine Sequenz

$$\pi = (p_0, a_0, p_1, \cdots, a_{n-1}, p_n)\,, \text{ wobei}$$

- $p_i \in Q$ für alle $0 \leq i \leq n$,
- $a_i \in \Sigma \cup \{\epsilon\}$ für alle $0 \leq i \leq n-1$,
- $p_{i+1} \in \Delta(p_i, a_i)$ für alle $0 \leq i \leq n-1$.

Gilt $p_0 = q_0$ und $p_n \in F$, so ist π ein akzeptierender Lauf von $\mathcal{A}$ auf w. Die Länge von π ist $|\pi| = n$.

D.h. insbesondere gilt für einen NEA ohne ϵ-Transitionen $|\pi| = |w|$.

Auf einem Wort können mehrere verschiedene Läufe von $\mathcal{A}$ existieren. Die Menge aller Läufe von $\mathcal{A}$ von einem Zustand $q \in Q$ aus bezeichnen wir als

$$\text{Runs}_{\mathcal{A}}(q) = \bigcup_{w \in \Sigma^*} \{\pi \mid \pi \text{ ist Lauf von } \mathcal{A} \text{ auf } w \text{ mit } \text{Start}(\pi) = q\}\,.$$

Damit kann die Funktion Δ in natürlicher Art auf $\Delta : Q \times \Sigma^* \to 2^Q$ erweitert werden durch

$$\Delta(q, w) = \bigcup_{\pi \in \text{Runs}_{\mathcal{A}}(q)} \{\text{Dest}(\pi) \mid \text{Word}(\pi) = w\}\,.$$

Schränken wir die Menge auf diejenigen Läufe ein, die in einen akzeptierenden Zustand führen, so bezeichnen wir diese Menge mit

$$\text{AccRuns}_{\mathcal{A}}(q) = \{\pi \in \text{Runs}_{\mathcal{A}}(q) \mid \text{Dest}(\pi) \in F\}\,.$$

Wir nennen weiterhin einen Zustand $q \in Q$ *produktiv*, wenn von ihm aus ein akzeptierender Pfad existiert, d.h. mit $n \in \mathbb{N}$ gilt: $\exists\, \pi \in \text{AccRuns}_{\mathcal{A}}(q)$.
Wir nenen einen Zustand $q \in Q$ *erreichbar*, wenn ein Pfad vom Startzustand aus zu diesem existiert, d.h. für ein $n \in \mathbb{N}$ existiert ein Lauf $\pi = (q_0, \cdots, q_n) \in \text{Runs}_{\mathcal{A}}(q_0)$ mit einem $q_i = q$ mit $0 \leq i \leq n$.
Ist ein Zustand $q \in Q$ sowohl erreichbar als auch produktiv, so nennen wir ihn *lebendig*, d.h. es existiert ein akzeptierender Pfad vom Startzustand aus, in dem q vorkommt.

Definition 3. Sei $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ ein NEA. Die von $\mathcal{A}$ akzeptierte Sprache ist $L(\mathcal{A}) = \{w \in \Sigma^* \mid \Delta(q_0, w) \cap F \neq \emptyset\}$.
Ein Wort w wird von einem NEA $\mathcal{A}$ genau dann akzeptiert, wenn $w \in L(\mathcal{A})$.

Definition 4. Un-/Eindeutigkeit. Sei $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ ein NEA. Der Grad der Uneindeutigkeit eines Eingabeworts $w \in \Sigma^*$ auf $\mathcal{A}$ (kurz: $\text{da}_{\mathcal{A}}(w)$) ist die Anzahl aller akzeptierende Pfade von $\mathcal{A}$ auf w.

$$\text{da}_{\mathcal{A}}(w) = |\{\pi \in \text{AccRuns}_{\mathcal{A}}(q_0) \mid w = \text{Word}(\pi)\}|$$

Dies kann auf den Grad der Uneindeutigkeit von $\mathcal{A}$ (kurz: $\text{da}(\mathcal{A})$) verallgemeinert werden. Dies ist die kleinste obere Schranke $k \in \mathbb{N}$ der Anzahl verschiedener akzeptierender Läufe über alle Wörter der Sprache $L(\mathcal{A})$.

$$\text{da}(\mathcal{A}) = \sup\left(\bigcup_{w \in \Sigma^*} \text{da}_{\mathcal{A}}(w)\right)$$

Existiert keine natürliche obere Schranke, so ist der Grad der Uneindeitigkeit von $\mathcal{A}$ unendlich, $\text{da}(\mathcal{A}) = \infty$. Gilt $\text{da}(\mathcal{A}) \leq 1$, so bezeichnen wir $\mathcal{A}$ als *eindeutig*.

Theorem 2.1. *Wenn eine Sprache L von einem NEA $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ mit $\mathrm{da}(\mathcal{A}) = k \in \{0, 1, \cdots, \infty\}$ akzeptiert wird, so existiert auch ein NEA $\mathcal{A}'$ ohne ϵ-Transitionen mit $w \in L(\mathcal{A}) \Leftrightarrow w \in L(\mathcal{A}')$ und $\mathrm{da}(\mathcal{A}') \leq \mathrm{da}(\mathcal{A})$. Zu $\mathcal{A}$ kann deterministisch in polynomieller Zeit ein solcher NEA $\mathcal{A}'$ konstruiert werden [3, Satz 2.2].*

Beweis. Sei $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ ein NEA. Wir konstruktieren einen zu $\mathcal{A}$ äquivalenten NEA ohne ϵ-Transitionen $\mathcal{A}' = (Q, \Sigma, \Delta', q_0, F')$. Die Zustandsmenge, das Eingabealphabet sowie der Startzustand sind gleich denen von $\mathcal{A}$.

Alle Zustände, die von einem Zustand q aus durch ϵ-Transitionen erreicht werden können, bezeichnen wir als ϵ-Hülle von $P \subseteq Q$:

$$\mathrm{Closure}_\epsilon(P) = \bigcup_{q \in P} \Delta(q, \epsilon)$$

Diese kann beispielsweise über Breitensuche über einen gerichteten Graphen, der nur die ϵ-Transitionen enthält, deterministisch in polynomieller Zeit bestimmt werden. Damit konstruieren wir eine neue Transitionsfunktion Δ' ohne ϵ-Transitionen allgemein mit $a \in \Sigma$ wie folgt:

$$\Delta'(q, a) = \mathrm{Closure}_\epsilon(P), \text{ wobei } P = \{p \mid \exists\, r \in \mathrm{Closure}_\epsilon(q) : p \in \Delta(r, a)\}.$$

Damit beinhaltet $\Delta(q, a)$ alle Zustände, die von q aus durch lesen des Symbols $a \in \Sigma$ sowie ϵ-Transitionen erreicht werden können, d.h.

$$\Delta'(q, a) = \bigcup_{\pi \in \mathrm{Runs}_\mathcal{A}(q)} \{\mathrm{Dest}(\pi) \mid a = \mathrm{Word}(\pi)\}.$$

Die neue Menge der akzeptierenden Zustände F' beinhaltet alle Zustände aus F. Gilt für den Startzustand, dass in seiner ϵ-Hülle ein Endzustand enthalten ist, so wird auch der Startzustand dieser Menge hingefügt, d.h.

$$F' = \begin{cases} F \cup \{q_0\} & \exists\, q \in \mathrm{Closure}_\epsilon(q_0) : q \in F, \\ F & \text{sonst.} \end{cases}$$

Es bleibt zu zeigen, dass die Konstruktion deterministisch in polynomieller Zeit erfolgen kann. Die Zustandsmenge, der Startzustand sowie das Eingabealphabet werden direkt übernommen. Die Berechnung der ϵ-Hülle kann wie angemerkt effizient durch bspw. Breitensuche erfolgen; Die Anzahl der Transitionen ist generell durch $\mathcal{O}(|Q|^2 \cdot |\Sigma|)$ beschränkt. Für die Menge der Endzustände gilt immer $F \subseteq Q$. Somit ist die Laufzeit generell polynomiell abhängig von $|Q|$ und $|\Sigma|$.

Die Korrektheit der Konstruktion kann per Induktion über die Wortstruktur gezeigt werden, siehe [3, Theorem 2.2]).

Der Grad der Uneindeutigkeit erhöht sich hierbei nicht, da durch die Konstruktion die resultierenden Pfade höchstens kürzer werden. Sei dazu mit $\pi_\epsilon = (q_0, \cdots, q_1, b, q_2, \epsilon, q_3, \cdots, q_F)$, wobei $q_1, q_2, q_3 \in Q,\ a, b, c \in \Sigma,\ q_F \in F$. Nach Eliminierung der ϵ-Transitionen gibts es zu jedem Pfad des Urpsrungsautomaten keine neuen Pfade, sondern es wird jeweils ein Teil eines Pfades durch einen kürzeren Infix ersetzt: $\pi = (q_0, \cdots, q_1, b, q_3, \cdots, q_F)$. Dies lässt sich per Induktion über die Länge der ϵ-Pfade zeigen. In manchen Fällen kommt es vor, dass mehrere ϵ-Pfade in nur einer neuen Transition resultieren, damit wird die Anzahl der Pfade des neuen Automaten auf einem Wort also kleiner, somit nimmt der Grad der Uneindeutigkeit also höchstens ab.

□

Lemma 2.2. *Es existiert ein deterministischer Algorithmus, der zu zwei eindeutigen NEAs $\mathcal{A}_1$ und $\mathcal{A}_2$ ohne ϵ-Transitionen mit n_1 bzw. n_2 Zuständen in polynomieller Zeit einen eindeutigen NEA $\mathcal{A}$, ohne ϵ-Transitionen mit $n_1 \cdot n_2$ Zuständen berechnet, so dass*

$$L(\mathcal{A}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2).$$

Den neuen Automaten bezeichnen wir als Produktautomat *von $\mathcal{A}_1$ und $\mathcal{A}_2$, welchen wir auch mit $\mathcal{A}_1 \cap \mathcal{A}_2$ abkürzen.*

Beweis. Seien $\mathcal{A}_1 = (Q_1, \Sigma_1, \Delta_1, q_{0_1}, F_1)$ und $\mathcal{A}_2 = (Q_2, \Sigma_2, \Delta_2, q_{0_2}, F_2)$ eindeutige NEAs ohne ϵ-Transitionen. Wir konstruieren einen neuen NEA $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ mit:

- $\Sigma = \Sigma_1 \cup \Sigma_2$,
- $Q = Q_1 \times Q_2$,
- $\Delta((t_1, t_2), a) = (\Delta_1(t_1, a), \Delta_2(t_2, a))$ für alle $a \in \Sigma_1 \cap \Sigma_2$,
- $q_0 = (q_{0_1}, q_{0_2})$ und
- $F = F_1 \times F_2$.

Diese Konstruktion simuliert die parallele Ausführung von $\mathcal{A}_1$ und $\mathcal{A}_2$. Der neue Automat $\mathcal{A}$ akzeptiert genau die Eingabewörter, die sowohl von $\mathcal{A}_1$ als auch von $\mathcal{A}_2$ akzeptiert werden. Da beide Ursprungsautomaten eindeutig sind, existiert für jedes Wort ihrer Sprache $L(\mathcal{A}_1)$ bzw. $L(\mathcal{A}_2)$ genau ein Lauf. Somit existiert auch für jedes akzeptierte Wort von $L(\mathcal{A})$ nur genau ein Lauf. Daraus folgt, dass auch $\mathcal{A}$ eindeutig ist und $L(\mathcal{A}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$. □

Lemma 2.3. *Es existiert ein deterministischer Algorithmus, der zu $k \in \mathbb{N}$ eindeutigen NEAs $\mathcal{A}_1, \cdots, \mathcal{A}_k$ ohne ϵ-Transitionen mit $n_1, \cdots, n_k$ Zuständen in polynomieller Zeit einen eindeutigen NEA $\mathcal{A}^k$, ohne ϵ-Transitionen mit $\prod_{i=1}^{k} n_i$ Zuständen berechnet, so dass*

$$L(\mathcal{A}) = \bigcap_{i=1}^{k} L(\mathcal{A}_i).$$

Den neuen Automaten bezeichnen wir als Produktautomat von $\mathcal{A}_1, \cdots, \mathcal{A}_k$, welchen wir auch mit $\bigcap_{i=1}^{k} \mathcal{A}_i$ abkürzen.

Beweis. Dies ist eine direkte Erweiterung von Lemma 2.2. Die Beweisführung erfolgt analog. □

Lemma 2.4. *([6, Theorem 2]) Sei $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ ein NEA. Dann gilt:* $\mathrm{da}(\mathcal{A}) = \infty$ *genau dann, wenn zwei Zustände $p \neq q \in Q$, ein Endzustand $q_F \in F$ sowie Wörter $u, v, w \in \Sigma^*$ existieren, so dass Transiotionen $(q_0, u, p), (p, v, p), (p, v, q), (q, v, q), (q, x, q_F) \in \Delta$ existieren (siehe Abbildung 1).*

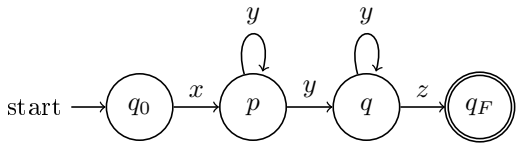

Abbildung 1: Mit $x, y, z \in \Sigma^*$ ist dies das Muster der Bedingung für einen unendlichen Grad der Uneindeutigkeit.

Beweis. Siehe [6, Theorem 2]. □

Theorem 2.5. *Sei $\mathcal{A}$ ein NEA. Es ist deterministisch in polynomieller Zeit entscheidbar, ob* $\mathrm{da}(\mathcal{A}) = \infty$ *gilt.*

Beweis. Der Beweis erfolgt nach [1, Lemma 2].
Nach Theorem 2.1 können wir deterministisch in polynomieller Zeit einen äquivalenten NEA $\mathcal{A}' = (Q, \Sigma, \Delta, q_o, F)$ ohne ϵ-Transitionen konstruieren, d.h. $\mathrm{da}(\mathcal{A}) = \infty \Leftrightarrow \mathrm{da}(\mathcal{A}') = \infty$. Seien Weiterhin alle Zustände lebendig[1].

Wir Zeigen: Für $\mathcal{A}'$ gilt $\mathrm{da}(\mathcal{A}') = \infty$ genau dann, wenn in seinem dreifachen Produktautomaten $\mathcal{A}' \cap \mathcal{A}' \cap \mathcal{A}'$ ein Pfad von einem Zustand $(p, p, q) \in Q^3$ nach $(p, q, q) \in Q^3$ existiert, wobei $p \neq q$ (dies korrespondiert direkt zu dem Muster aus Lemma 2.4).

Gelte $\mathrm{da}(\mathcal{A}') = \infty$. Dann existiert nach Lemma 2.4 ein Wort $v \in \Sigma^*$ mit drei Pfaden $\pi_1 = (p, v, p)$, $\pi_2 = (p, v, q)$ sowie $\pi_3 = (q, v, q)$ auf $\mathcal{A}'$. Da für alle drei Pfade $\mathrm{Word}(\pi_1) = \mathrm{Word}(\pi_2) = \mathrm{Word}(\pi_3)$ gilt, existiert auch in dem dreifachen Produktautomaten von $\mathcal{A}'$ nach Lemma 2.3 ein Pfad π mit $\mathrm{Word}(\pi) = v$ von $(\mathrm{Start}(\pi_1), \mathrm{Start}(\pi_2), \mathrm{Start}(\pi_2)) = (p, p, q)$ nach $(\mathrm{Dest}(\pi_1), \mathrm{Dest}(\pi_2), \mathrm{Dest}(\pi_2)) = (p, q, q)$.

Wenn nun ein Pfad π von (p, p, q) nach (p, q, q) mit $\mathrm{Word}(\pi) = v$ auf $\mathcal{A}' \cap \mathcal{A}' \cap \mathcal{A}'$ existiert, müssen in $\mathcal{A}'$ auch drei Pfade $\pi_i,\ i \in \{1, 2, 3\}$ existieren für die $v = \mathrm{Word}(\pi_i)$ gilt, wobei diese die Form $(p, v, p), (p, v, q)$ und (q, v, q) haben, was dem Muster entspricht.
Es existieren somit Wörter $x \in L(\mathcal{A}')$, die die Form $x = uv^i w$ mit $i \in \mathbb{N}$ sowie $u, w \in \Sigma^*$ haben. Per Induktion über i lässt sich leicht zeigen, dass für diese Wörter $\mathrm{da}_{\mathcal{A}'}(x) \geq i$ gilt. Aus $\lim_{i\to\infty} \mathrm{da}_{\mathcal{A}'}(x) = \infty$ folgt $\mathrm{da}(\mathcal{A}) = \infty$. □

Definition 5. Sei $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ ein NEA. Dann bezeichnet $\mathrm{D}_{\mathcal{A}}(q, k)$ mit $q \in Q$ und $k \in \mathbb{N}$ die Anzahl der möglichen akzeptierenden Läufe der Länge k von Zustand q in einen Endzustand:

$$\mathrm{D}_{\mathcal{A}}(q, k) = |\{\pi \in \mathrm{AccRuns}_{\mathcal{A}}(q) \mid |\pi| = k\}|.$$

$\text{ACC-SEQ}_{\mathcal{A}}(k)$ mit $k \in \mathbb{N}$ bezeichnet die Anzahl der akzeptierenden Läufe der Länge k vom Startzustand q_0 in einen Endzustand:

$$\text{ACC-SEQ}_{\mathcal{A}}(k) = \mathrm{D}_{\mathcal{A}}(q_0, k).$$

3 Differenzengleichungen

Die Beweise zur effizienten Entscheidbarkeit des Inklusions- und Äquivalenzproblems eindeutiger Automaten und jener mit festem Grad der Uneindeitigkeit basieren auf Differenzengleiuchungen. Hier werden nun die die elementaren Lemmata über diese vorgestellt.

Definition 6. Sei $A : \mathbb{N} \to \mathbb{R}$. A erfüllt eine homogene lineare Differenzengleichung (oder auch Rekursionsgleichung) mit konstanten Koeffizienten vom Grad $n \in \mathbb{N}$ genau dann, wenn Konstanten $c_i \in \mathbb{R}$ für $1 \leq i \leq n$ existieren mit $c_n \neq 0$ so dass für alle $k \geq 0$ gilt:

$$\sum_{i=0}^{n} c_i \cdot A(k + i) = 0 \tag{1}$$

Im Folgenden wird *homogene lineare Differenzengleichung mit konstanten Koeffizienten* mit *Differenzengleichung* abgekürzt.

Lemma 3.1. *Sei S eine nichtleere endliche Menge. Habe weiterhin für alle $s \in S$, $A_s : \mathbb{N} \to \mathbb{R}$ folgende Eigenschaft. Es existieren $c^1_{s,t} \in \mathbb{R}$ so dass für alle $k \geq 0$:*

$$A_s(k + 1) = \sum_{t\in S} c^1_{s,t} \cdot A_t(k)$$

[1] Es kann für jeden Zustand bspw. via Breitensuche effizient entschieden werden, ob er erreichbar sowie produktiv ist. Somit kann ein neuer Automat konstruiert werden, der diesselbe Sprache erkennt, jedoch nur noch lebendige Zustände enthält.

Dann erfüllt die Funktion A_s eine Differenzengleichung des Grades $\leq |S|$ für alle $s \in S$.

Beweis. Sei $s \in S$. Zuerst werden $|S|$ verschiedene Gleichungen für $1 \leq j \leq |S|$ sowie für alle $k \geq 0$ in folgender Form hergeleitet:

$$A_s(k+j) = \sum_{t \in S} c_{s,t}^j \cdot A_t(k) \tag{2}$$

Die erste Gleichung ergibt sich direkt aus den Voraussetzungen von Lemma 3.1. Alle weiteren werden induktiv bestimmt. Dazu wird k duch $k+1$ ersetzt:

$$A_s(k+1+j) = \sum_{t \in S} c_{s,t}^j \cdot A_t(k+1)$$

und durch Einsetzen von Gleichung 1 sowie Ausmultiplizieren erhalten wir for alle $k \geq 0$ die Gleichungen

$$\begin{aligned} A_s(k+1+j) &= \sum_{t \in S} c_{s,t}^j \cdot \left[\sum_{t \in S} c_{s,t}^1 \cdot A_t(k) \right] \\ &= \sum_{t \in S} c_{s,t}^{j+1} \cdot A_t(k) \end{aligned}$$

Diese $|S|$ Gleichungen beschreiben nun also $A_s(k+1), \cdots, A_s(k+|S|)$ einer feste Folge $A_s(k),\ s \in S$. Da wir aus diesen Gleichungen eine Differenzengleichung für A_s bestimmen wollen, müssen nun also alle anderen Abhängigkeiten $A_t(k),\ t \in S - \{s\}$ eleminiert werden. Dazu betrachten wir die Gleichungen 2 als lineares Gleichungssystem.
Es bleibt zu zeigen, dass sich aus dem linearen Gleichungssystem immer die Variablen $A_t(k),\ t \in S - \{s\}$ eliminieren lassen, d.h. es den Rang $|S|$ aufweist bzw. $|S|$ Gleichungen linear unabhängig sind. Dies folgt unmittelbar aus den $|S|$ verschiedenen Gleichungen für $A_s(k+j)$, da jede von ihnen eben nur genau die Variable $A_s(k+i)$ für ein festes $1 \leq i \leq |S|$ enthält. Da wir also $|S|$ verschiedene linear unabhängige Gleichungen haben, lassen sich auch $|S|-1$ Variablen mit bekannten Methoden der linearen Algebra eliminieren, nämlich genau die $A_t(k) \in S - \{s\}$.

□

Lemma 3.2. *Seien $A, B : \mathbb{N} \to \mathbb{R}$, die Differenzengleichungen von Grad a bzw. b erfüllen. Dann erfüllt auch die Folge $D : \mathbb{N} \to \mathbb{R}$, $D(k) = A(k) - B(k)$ für alle $k \in \mathbb{N}$ eine Differenzengleichung vom Grad $\leq a+b$.*
D.h. insbesondere wenn für $0 \leq k \leq a+b-1, A(k) = B(k)$ gilt, dann gilt $A(k) = B(k)\ \forall k \in \mathbb{N}$.

Beweis. Nach Konstruktion gilt für alle $0 \leq j \leq a+b,\ k \geq 0$

$$D(k+j) = A(k+j) - B(k+j)$$

Die Funktionen $A(k+j)$ und $B(k+j)$ können durch Linearkombinationen von $A(k+i_1),\ 0 \leq i_1 \leq a-1$ und $B(k+i_2),\ 0 \leq i_2 \leq b-1$ ersetzt werden. Daraus ergibt sich für alle $k \geq 0$ sowie $0 \leq j \leq a+b$ folgende Gleichung:

$$D(k+j) = \sum_{i_1=0}^{a-1} a_{i_1}^j \cdot A(k+i_1) + \sum_{i_2=0}^{b-1} b_{i_2}^j \cdot B(k+i_2)$$

Auch hier kann durch Eliminierung wie bereits im Beweis zu Lemma 3.1 gezeigt, eine Differenzengleichung konstruiert werden. Dazu nutzen wir induktiv bestimmte verschiedene Gleichungen für $D(k+i)$ mit $0 \leq i \leq a+b+1$. Die resultierende Differenzengleichung hat somit den Grad $a+b$

Ferner folgt aus dem Fall $A(k) = B(k)$ für alle $0 \leq k \leq a+b-1$, dass für diese k auch $D(k) = 0$ gelten muss. Da $D(k)$ eine Differenzengleichung des Grades kleiner oder gleich $a+b$ erfüllt, sind alle höheren Funktionswerte gleich null[2]. Somit gilt für alle $k > a+b$, dass auch $A(k) - B(k) = 0$ erfüllt wird. □

4 Inklusionsproblem und Äquivalenz eindeutiger nichtdeterministischer Automaten

Lemma 4.1. *Sei $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ ein NEA ohne ϵ-Transitionen. Mit $q \in Q$ und $k \in \mathbb{N}$ gilt:*

$$\begin{aligned} \mathrm{D}_{\mathcal{A}}(q,0) &= \begin{cases} 1 & q \in F, \\ 0 & \textit{sonst.} \end{cases} \\ \mathrm{D}_{\mathcal{A}}(q,k+1) &= \sum_{p \in Q} a_{q,p} \cdot \mathrm{D}_{\mathcal{A}}(p,k) \textit{ mit } a_{q,p} = |\{a \in \Sigma \mid p \in \Delta(q,a)\}| \end{aligned}$$

Beweis. Die Behauptung lässt sich mit Definition 2 leicht per Induktion über k beweisen, da $\mathcal{A}$ nach Voraussetzung keine ϵ-Transitionen besitzt und somit $k = |\pi|$ gilt. □

Lemma 4.2. *Sei $\mathcal{A}$ ein NEA ohne ϵ-Transitionen mit n Zuständen. Dann erfüllt die Folge* ACC-SEQ$_{\mathcal{A}}$ *eine Differenzengleichung des Grades $\leq n$.*

Beweis. Mit $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$, $|Q| = n$ sei $A_t(k) = \mathrm{D}_{\mathcal{A}}(t,k)$ für alle $t \in Q$. Nach Definition gilt $A_{q_0}(k) = \text{ACC-SEQ}_{\mathcal{A}}(k)$. Diese Gleichungen sind genau der Form von Lemma 4.1 und nach Lemma 3.1 erfüllt A_{q_0} somit eine Differenzengleichung vom Grad $\leq n$. □

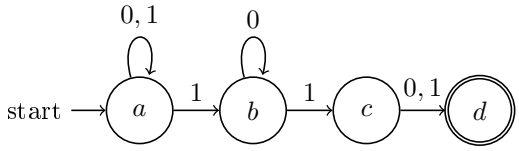

Abbildung 2: Dies ist der Automat, für den beispielhaft die Differenzengleichung berechnet wird.

Beispiel 1. Sei $\mathcal{A}$ ein NEA mit dem Alphabet $\Sigma = \{0,1\}$ der Automat aus Abbildung 2. Um nun eine Differenzengleichung nach Lemma 4.2 zu erhalten, werden folgende Schritte durchgeführt:

1. Aufstellen der Gleichungen $\mathrm{D}_{\mathcal{A}}(t, k+1)$, $t \in Q$:

$$\begin{aligned} \mathrm{D}_{\mathcal{A}}(a,k+1) &= 2 \cdot \mathrm{D}_{\mathcal{A}}(a,k) + \mathrm{D}_{\mathcal{A}}(b,k) && (3) \\ \mathrm{D}_{\mathcal{A}}(b,k+1) &= \mathrm{D}_{\mathcal{A}}(b,k) + \mathrm{D}_{\mathcal{A}}(c,k) && (4) \\ \mathrm{D}_{\mathcal{A}}(c,k+1) &= 2 \cdot \mathrm{D}_{\mathcal{A}}(d,k) && (5) \\ \mathrm{D}_{\mathcal{A}}(d,k+1) &= 0 && (6) \end{aligned}$$

[2] Für eine Differenzengleichung des Grades $n \in \mathbb{N}$ ergibt sich der Wert $A(k)$ mit $k \in \mathbb{N}$ aus einer Linearkombinationen von Funktionswerten $A(k-1), A(k-2), \cdots, A(k-n)$. Wenn nun für die ersten $k - n \leq m \leq k$ Funktionswerte $A(m) = 0$ gilt, so wissen wir auch, dass auch für alle weiteren $m > k$ Werte $A(m) = 0$ gelten muss.

2. Iteratives Aufstellen der Gleichungen $\mathrm{D}_{\mathcal{A}}(a, k+j),\ 1 \leq j \leq |Q| = 4$:

$$\begin{aligned}
\mathrm{D}_{\mathcal{A}}(a,k+1) &= 2 \cdot \mathrm{D}_{\mathcal{A}}(a,k) + \mathrm{D}_{\mathcal{A}}(b,k) && (7)\\
\mathrm{D}_{\mathcal{A}}(a,k+2) &= 2 \cdot \underbrace{\mathrm{D}_{\mathcal{A}}(a,k+1)}_{(3)} + \underbrace{\mathrm{D}_{\mathcal{A}}(b,k+1)}_{(4)} \\
&= 4 \cdot \mathrm{D}_{\mathcal{A}}(a,k) + 3\,\mathrm{D}_{\mathcal{A}}(b,k) + \mathrm{D}_{\mathcal{A}}(c,k) && (8)\\
\mathrm{D}_{\mathcal{A}}(a,k+3) &= 4 \cdot \underbrace{\mathrm{D}_{\mathcal{A}}(a,k+1)}_{(3)} + 3\underbrace{\mathrm{D}_{\mathcal{A}}(b,k+1)}_{(4)} + \underbrace{\mathrm{D}_{\mathcal{A}}(c,k+1)}_{(5)} \\
&= 8 \cdot \mathrm{D}_{\mathcal{A}}(a,k) + 7 \cdot \mathrm{D}_{\mathcal{A}}(b,k) + 3 \cdot \mathrm{D}_{\mathcal{A}}(c,k) + 2 \cdot \mathrm{D}_{\mathcal{A}}(d,k) && (9)\\
\mathrm{D}_{\mathcal{A}}(a,k+4) &= 8 \cdot \underbrace{\mathrm{D}_{\mathcal{A}}(a,k+1)}_{(3)} + 7 \cdot \underbrace{\mathrm{D}_{\mathcal{A}}(b,k+1)}_{(4)} + 3 \cdot \underbrace{\mathrm{D}_{\mathcal{A}}(c,k+1)}_{(5)} + 2 \cdot \underbrace{\mathrm{D}_{\mathcal{A}}(d,k+1)}_{(6)} \\
&= 16 \cdot \mathrm{D}_{\mathcal{A}}(a,k) + 15 \cdot \mathrm{D}_{\mathcal{A}}(b,k) + 7 \cdot \mathrm{D}_{\mathcal{A}}(c,k) + 6 \cdot \mathrm{D}_{\mathcal{A}}(d,k) && (10)
\end{aligned}$$

3. Eliminieren aller Variablen $\mathrm{D}_{\mathcal{A}}(x,k),\ x \in \{b,c,d\}$:
   Zuerst eliminieren wir $\mathrm{D}_{\mathcal{A}}(d,k)$, indem wir $3\times(9)$ von (10) subtrahieren. Dies ergibt die neue Gleichung:

$$-3 \cdot \mathrm{D}_{\mathcal{A}}(a,k+3) + \mathrm{D}_{\mathcal{A}}(a,k+4) = -8 \cdot \mathrm{D}_{\mathcal{A}}(a,k) - 6 \cdot \mathrm{D}_{\mathcal{A}}(b,k) - 2 \cdot \mathrm{D}_{\mathcal{A}}(c,k) \qquad (11)$$

   In diesem Fall genügt es $2\times(8)$ zu (11) hinzu zu addieren, um die ungewünschten Variablen zu eliminieren. Da $\text{ACC-SEQ}_{\mathcal{A}}(k) = \mathrm{D}_{\mathcal{A}}(a,k)$ für alle $k \geq 0$, lautet die Differenzengleichung für $\text{ACC-SEQ}_{\mathcal{A}}(k)$:

$$2 \cdot \text{ACC-SEQ}_{\mathcal{A}}(k+2) - 3 \cdot \text{ACC-SEQ}_{\mathcal{A}}(k+3) + \text{ACC-SEQ}_{\mathcal{A}}(k+4) = 0 \qquad (12)$$

Lemma 4.3. *Sei $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ ein NEA ohne ϵ-Transitionen. Dann existiert ein Algorithmus mit der Eingabe von $\mathcal{A}$ und dem String 1^n mit $n \in \mathbb{N}$, der die ersten n Werte von $\text{ACC-SEQ}_{\mathcal{A}}$ in polynomieller Zeit berechnet.*

Beweis. Zur Berechnung der $\text{ACC-SEQ}_{\mathcal{A}}$ Werte nutzen wir Algorithmus 1. Der Algorithmus berechnet bottom-up $\mathrm{D}_{\mathcal{A}}(q,i)$ für alle $q \in Q$ und $0 \leq i \leq n$. Jede Iteration ergibt somit jeweils die $\mathrm{D}_{\mathcal{A}}(q,i+1)$ Werte aus den vorherigen Werten $\mathrm{D}_{\mathcal{A}}(p,i)$ für alle $q,p \in Q$ wie bereits in Lemma 4.1 beschrieben.

Eingabe: Eindeutiger NEA $\mathcal{A} = (Q, \Sigma, \Delta, q_0, F)$ und Länge des Laufes $n \in \mathbb{N}$ unär kodiert: 1^n
Ausgabe: Anzahl $k \in \mathbb{N}$ der akzeptierenden Läufe der Länge n.

1. Berechne $\mathrm{D}_{\mathcal{A}}(q,0)$ für alle $q \in Q$.
2. Setze $i := 1$
3. Gilt $i > n$, beende Berechnung mit Ausgabe $k := \mathrm{D}_{\mathcal{A}}(q_0, n)$
4. Berechne für alle $q \in Q$:

$$\mathrm{D}_{\mathcal{A}}(q,i) = \sum_{(q,a,q') \in \Delta} \mathrm{D}_{\mathcal{A}}(q', i-1)$$

5. Setze $i = i + 1$ und gehe zu Schritt 3

Algorithmus 1: Berechnung von $\text{ACC-SEQ}_{\mathcal{A}}(n)$ zu einem NEA $\mathcal{A}$.

Es bleibt zu zeigen, dass diese Berechnung in polynomieller Zeit möglich ist. Die initialize Berechnung liegt in $\mathcal{O}(|Q|)$. Die Schleife von Schritt 3 bis 5 wird genau n mal ausgeführt. Alle Werte in Schritt 1 für $\mathrm{D}_{\mathcal{A}}(p,0),\ p \in Q$ sind ≤ 1. Weiterhin lässt sich per Induktion über $k \in \mathbb{N}$ zeigen, dass alle weiteren Werte in Schritt 4 für $\mathrm{D}_{\mathcal{A}}(p,k),\ p \in Q$ durch $\mathcal{O}(|\Delta|^k)$ beschränkt sind. Weiterhin sind diese Werte positive ganze Zahlen, deren binäre Darstellung also maximal $\log_2(|\Delta|^n) = n \cdot \log_2(|\Delta|)$ lang ist. Daraus ergibt sich, dass diese Werte stets kleiner als die Eingabe, nämlich $\mathcal{A}$ und 1^n sind.

Somit ergibt sich eine Laufzeit in $\mathcal{O}(n^2 \cdot |Q| \cdot |\Delta| \cdot \log_2(|\Delta|))$ für den gesamten Algorithmus, da die äußere Schleife genau n mal ausgeführt wird, die Berechnung der $\mathrm{D}_{\mathcal{A}}(q,i)$ Werte pro Iteration für jeden Zustand, also $|Q|$ mal ausgeführt wird, die Summe maximal $|\Delta|$ Summanden umfasst, also max. $|\Delta|$ mal addiert wird, und die Addition $n \cdot \log_2(|\Delta|)$ Rechenschritte benötigt [2, 34.1 Codierungen]. □

Definition 7. Das Inklusionsproblem für eine Klasse K von regulären Ausdrücken, regulären Grammatiken oder NEAs ist das Problem zu entscheiden, ob mit gegebenem $\mathcal{A}$ und $\mathcal{B} \in K$ Folgendes gilt:

$$L(\mathcal{A}) \subseteq L(\mathcal{B}).$$

Lemma 4.4. *Es existiert ein deterministischer Algorithmus, der in polynomieller Zeit mit der Eingabe zweier eindeutiger NEAs $\mathcal{A}_1$ und $\mathcal{A}_2$ mit $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$ entscheidet, ob dies eine echte Teilmenge ist.*

Beweis. Nach Theorem 2.1 können zu $\mathcal{A}_1$ und $\mathcal{A}_2$ äquivalente NEAs $\mathcal{A}_1'$ und $\mathcal{A}_2'$ mit n_1 bzw. n_2 vielen Zuständen konstruiert werden, die keine ϵ-Transitionen haben und ebenfalls n_1 bzw. n_2 Zustände haben und eindeutig sind.
Sei nach Voraussetzung $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$. Es handelt es sich genau dann um eine echte Teilmenge, wenn $\text{ACC-SEQ}_{\mathcal{A}_1'}(k) \neq \text{ACC-SEQ}_{\mathcal{A}_2'}(k)$ für ein $k \in \mathbb{N}$. Nach Lemma 4.1 erfüllen beide Folgen eine Differenzengleichung des Grades $\leq n_1$ bzw. $\leq n_2$. Weiter folgt nach Lemma 3.2, dass $\text{ACC-SEQ}_{\mathcal{A}_1'}(n) = \text{ACC-SEQ}_{\mathcal{A}_2'}(n)$ mit $n \in \mathbb{N}$ genau dann, wenn $\text{ACC-SEQ}_{\mathcal{A}_1'}(k) = \text{ACC-SEQ}_{\mathcal{A}_2'}(k)$ für alle $0 \leq k \leq n_1 + n_2 - 1$ gilt.
Dies kann nach Lemma 4.3 deterministisch in polynomiell begrenzter Zeit geprüft werden, indem für beide Automaten die ersten $0 \leq k \leq n_1 + n_2 - 1$ von $\text{ACC-SEQ}_{\mathcal{A}_i}(k),\ i \in \{1,2\}$ mit Algorithmus 1 berechnen. Die Eingabe ist dabei jeweils der entsprechende Automat sowie die zu vergleichende Länge k, welche abhängig von den Automaten durch die Anzahl ihrer Zustände beschränkt ist. □

Lemma 4.5. *Seien M_1, M_2 endliche Mengen. $M_1 \subseteq M_2$ gilt genau dann, wenn $\neg M_1 \cap M_2 \subsetneq M_1$ gilt.*

Beweis.

$$\begin{aligned}
& & & M_1 \subseteq M_2 \\
\Leftrightarrow & & \forall x: \quad & x \in M_1 \rightarrow x \in M_2 \\
\Leftrightarrow & & \forall x: \quad & x \notin M_1 \vee x \in M_2 \\
\Leftrightarrow & \neg\neg & \forall x: \quad & x \notin M_1 \vee x \in M_2 \\
\Leftrightarrow & \neg & \exists x: \quad & x \in M_1 \wedge x \notin M_2 \\
\Leftrightarrow & \neg & \exists x: \quad & x \notin M_1 \cap M_2 \wedge x \in M_1 \\
\Leftrightarrow & & & \neg(M_1 \cap M_2 \subsetneq M_1)
\end{aligned}$$

□

Theorem 4.6. *Das Inklusionsproblem ist für zwei eindeutige NEAs $\mathcal{A}_1$ und $\mathcal{A}_2$ mit oder ohne ϵ-Transitionen deterministisch in polynomieller Zeit entscheidbar.*

Beweis. Sind $\mathcal{A}_1$ und $\mathcal{A}_2$ eindeutige ϵNEAs, so lassen sich nach Lemma 2.1 äquivalente NEAs $\mathcal{A}'_i$ ohne ϵ-Transitionen konstruieren. Durch Lemma 2.2 lässt sich ein NEA $\mathcal{B}$ konstrieren mit $L(\mathcal{B}) = L(\mathcal{A}'_1) \cap L(\mathcal{A}'_2)$. Weiterhin lässt sich mit Lemma 4.4 entscheiden, ob $L(\mathcal{B}) \subsetneq L(\mathcal{A}'_1)$ gilt. Dies ist möglich, da nach Konstruktion immer $L(\mathcal{B}) \subseteq L(\mathcal{A}'_1)$ gilt. Nach Lemma 4.5 lässt sich also für $\mathcal{A}_1$ und $\mathcal{A}_2$ entscheiden, ob $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$ gilt. Sowohl die Konstruktion von $\mathcal{A}'_i$, des Produktautomaten als auch die Überprüfung auf die Teilmenge können deterministisch in polynomieller Zeit vorgenommen werden. Also lässt sich das Inklusionsproblem effizient entscheiden. □

Theorem 4.7. *Es existiert ein Algorithmus, der deterministisch in polynomieller Zeit mit der Eingabe zweier eindeutiger NEAs $\mathcal{A}_1$ und $\mathcal{A}_2$ mit n_1 bzw. n_2 vielen Zuständen, wobei $L(\mathcal{A}_1) \subsetneq L(\mathcal{A}_2)$ gilt, ein Ausgabewort $w \in L(\mathcal{A}_2) - L(\mathcal{A}_1)$ von minimaler Länge $|w| \leq n_1 + n_2$ berechnet.*

Beweis. Nach Lemma 2.1 nehmen wir an, dass die eindeutigen NEAs $\mathcal{A}_1 = (Q_1, \Sigma, \Delta_1, q_{0_1}, F_1)$ und $\mathcal{A}_2 = (Q_2, \Sigma, \Delta_2, q_{0_2}, F_2)$ mit $n_1 = |Q_1|$ und $n_2 = |Q_2|$ keine ϵ-Transitionen haben. Ferner gilt, dass $L(\mathcal{A}_1) \subsetneq L(\mathcal{A}_2)$ genau dann erfüllt ist, wenn ein Wort $w \in \Sigma^*$ existiert, so dass $w \in L(\mathcal{A}_2) - L(\mathcal{A}_1)$ wobei nach dem Beweis von Lemma 4.4 $j = |w| < n_1 + n_2$ gilt. Weiter folgt, dass dann auch gilt $\text{ACC-SEQ}_{\mathcal{A}_1}(j) \neq \text{ACC-SEQ}_{\mathcal{A}_2}(j)$.

Algorithmus 2 berechnet mit der Eingabe $\mathcal{A}_1$ und $\mathcal{A}_2$ ein Wort w minimaler Länge, so dass $|w| < n_1 + n_2$ und $w \in L(\mathcal{A}_2) - L(\mathcal{A}_1)$.

Dieser kann deterministisch in polynomieller Laufzeit durchgeführt werden:

- Nach Lemma 4.3 und Satz 4.6 kann $j_0 < n_1 + n_2$ von $\mathcal{A}_1$ und $\mathcal{A}_2$ deterministisch in polynomieller Zeit bestimmt werden.
- Die Schritte 4 bis 9 werden genau j_0 mal ausgeführt.
- Für alle $1 \leq i, j \leq i_0$, $p_1 \in Q_1$ und $p_2 \in Q_2$ können $\text{D}_{\mathcal{A}_1}(p_1, i)$ sowie $\text{D}_{\mathcal{A}_2}(p_2, j)$ ebenfalls deterministisch in polynomieller Zeit bestimmt werden, dies wurde bereits im Beweis zu Lemma 4.3 gezeigt.

Korrektheit, Beweis durch Vollständige Induktion über die Anzahl der Iterationen j.
Hypthese: Für $0 \leq j \leq i_0$ gilt nach j Iterationen der Schritte 4 bis 9 des Algorithmus, dass $|x| = j$ und es existiert ein Wort y mit $|y| = i_0 - j$, so dass gilt: $xy \in L(\mathcal{A}_2) - L(\mathcal{A}_1)$.
Induktionsanfang: Sei $j = 0$.
In Schritt 2 wird x mit dem leeren Wort initialisiert. Somit gilt $|x| = j = 0$. Ferner gilt mit Voraussetzungen und nach Definition 5 von ACC-SEQ, dass ein Wort y der Länge $i_0 - j = i_0$ existiert mit $xy = y \in L(\mathcal{A}_2) - L(\mathcal{A}_1)$.
Induktionsschritt: Gelte $0 < j < j_0$, somit auch $|x| = j$, da pro Iteration in Schritt 7 das Wort w um einen Buchstaben verlängert wird. Nun werden in Schritt 5 die neuen Erreichbarkeitsmengen für beide Automaten für jeden Buchstaben ihres gemeinsamen Alphabets berechnet. Ebenfalls werden dann in Schritt 6 die Differenzen der Anzahl akzeptierender Läufe mit der Länge $j_0 - |x| - 1 = j_0 - j - 1$ der beiden Automaten berechnet. In Schritt 7 wird nun der kleinste Index i mit einer Differenz der akzeptierenden Läufe ungleich null ausgewählt. Der korrespondierende Buchstabe ist a_i. Das Wort x wird nun um diesen Buchstaben a_i verlängert und die Erreichbarkeitsmenge entsprechend aktualisiert. Nun gilt $|xa| = j + 1$. Nach Definition 5 von D gilt nun, dass ein Wort y existiert, welches von einem der Zustände der aktuellen Erreichbarkeitsmengen aus die Länge $j_0 - (j+1)$ hat, für das gilt, dass die Differenz der akzeptierten Läufe auf beiden Automaten ungleich ist und somit $xay \in L(\mathcal{A}_2) - L(\mathcal{A}_1)$. □

5 Erweiterung auf Automaten festen Grades der Uneindeutigkeit

Die Prinzipien zur effizienten Entscheidbarkeit des Inklusionsproblems für eindeutige NEAs funktionieren durch geschickte Konstruktion auch für uneindeutige NEAs mit festem Grad der Unein-

Eingabe: Eindeutige NEAs $\mathcal{A}_1 = (Q_1, \Sigma, \Delta_1, q_{0_1}, F_1)$ und $\mathcal{A}_2 = (Q_2, \Sigma, \Delta_2, q_{0_2}, F_2)$ mit $\mathcal{A}_1 \subsetneq \mathcal{A}_2$
Ausgabe: Ein Wort minimaler Länge mit $w \in L(\mathcal{A}_2) - L(\mathcal{A}_1)$

1. Berechne die minimale Länge eines Laufes, so dass die Anzahl der akzeptierten Wörter verschieden ist:
   $j_0 \coloneqq \min\{j \mid \text{ACC-SEQ}_{\mathcal{A}_1}(j) \neq \text{ACC-SEQ}_{\mathcal{A}_2}(j)\}$
2. Initialisiere das Ausgabewort mit dem leeren Wort:
   $x \coloneqq \epsilon$
3. Initialisiere Erreichbarkeitsmengen für beide Automaten mit ihrem Startzustand:
   $A_x \coloneqq \{q_{0_1}\}$
   $B_x \coloneqq \{q_{0_2}\}$
4. Prüfe, ob die bisherige Wortlänge von x der minimalen Länge entspricht, bei der die Anzahl der akzeptierten Läufe der Automaten verschieden ist; Ist dies der Fall, stoppe die Berechnung und gebe das Wort x aus:
   if $|x| = j_0$ *then* halte mit der Ausgabe x
5. Berechne separat die Erreichbarkeitsmengen beider Automaten für jeden Buchstaben des gemeinsamen Alphabets Σ:
   Für $1 \leq i \leq |\Sigma|$:
   $A_{xa_i} \coloneqq \{p_1' \in Q_1 \mid \exists\ p_1 \in Q_1 \text{ mit } p_1' \in \Delta_1(p_1, a_i)\}$
   $B_{xa_i} \coloneqq \{p_2' \in Q_2 \mid \exists\ p_2 \in Q_2 \text{ mit } p_2' \in \Delta_2(p_2, a_i)\}$
6. Prüfe für jede dieser Erreichbarkeitsmengen, ob die Anzahl der akzeptierten Wörter von jedem Zustand dieser Mengen die Anzahl der akzeptierten Wörter mit der bis j_0 verbleibenden Buchstaben unterschiedlich ist:
   Für $1 \leq i \leq |\Sigma|$:
   $$d_i = \sum_{p_2' \in B_{xa_i}} \mathrm{D}_{\mathcal{A}_2}(p_2', j_0 - |x| - 1) - \sum_{p_1' \in A_{xa_i}} \mathrm{D}_{\mathcal{A}_1}(p_1', j_0 - |x| - 1)$$
7. Wähle den kleinsten Index für den dies der Fall ist und erweitere das Ausgabewort mit dem korrespondierenden Buchstaben.
   $i_0 \coloneqq \min\{i \mid d_i \neq 0\}$
   $x = xa_{i_0}$
8. Aktualisiere die Erreichbarkeitsmengen:
   $A_x = A_{xa_{i_0}}$
   $B_x = B_{xa_{i_0}}$
9. Gehe zu Schritt 4.

Algorithmus 2: Berechnung eines kürzesten Wortes $w \in L(\mathcal{A}_2) - L(\mathcal{A}_1)$ berechnet zu zwei eindeutigen NEAs $\mathcal{A}_1$ und $\mathcal{A}_2$ mit $\mathcal{A}_1 \subsetneq \mathcal{A}_2$

deutigkeit, $\mathrm{da}(\mathcal{A}) \leq k \in \mathbb{N}$.
Die Idee ist die Konstruktion von k Automaten $\mathcal{A}_i, 1 \leq i \leq k$, die jeweils einen festen Grad i der Uneindeutigkeit simulieren und nur dann akzeptieren, wenn $\mathcal{A}$ zu einem Wort mindestens i verschiedene akzeptierende Läufe besitzt. Zu diesen Automaten können die ACC-SEQ$_{\mathcal{A}_i}(n)$ Werte mit Algorithmus 1 berechnet werden. Diese beschreiben kombinatorisch die Anzahl der akzeptierten Wörter der Länge n mit höchstens i verschiedenen Läufen von $\mathcal{A}$.
Die Anzahl aller akzeptierten Wörter des Ursprungsautomaten einer bestimmten Wortlänge kann dann als Linearkombination aus diesen k verschiedenen ACC-SEQ$_{\mathcal{A}_i}(n)$ darstellt werden. Es lässt sich ähnlich wie in Lemma 4.2 zeigen, dass diese Gleichung eine Differenzengleichung erfüllt.

Das eigentliche Kriterium zur Entscheidung des Inklusionsproblems stützt sich ebenfalls auf die Existenz dieser Differenzengleichung. Zur algorithmischen Prüfung wird auch hier die Mengentheoretische Beziehung von Lemma 4.5, d.h. ein Produktautomat genutzt. Im Vergleich zum Fall der eindeutigen NEAs muss hier jedoch weitergehend die Existenz der Differenzengleichung ausgenutzt werden, um die Koeffizienten der Linearkombination zu bestimmem, also ein lineares Gleichungssystem gelöst werden. Damit kann schließlich aus den berechneten ACC-SEQ$_{\mathcal{A}_i}(n)$ Werten die eigentliche Anzahl der von $\mathcal{A}$ akzeptierten Wörter bestimmt werden.

Theorem 5.1. *Sei $k \geq 1$. Das Inklusionsproblem ist für zwei NEAs $\mathcal{A}_1$ und $\mathcal{A}_2$ mit beschränktem Grad der Uneindeutigkeit* $\mathrm{da}(\mathcal{A}_1) \leq k$ *und* $\mathrm{da}(\mathcal{A}_2) \leq k$ *deterministisch in polynomieller Zeit entscheidbar.*

Beweis. Die Beweisführung ist nahezu identisch zu Theorem 4.6. Für Details siehe [4, Korollar 5.9]. □

Lemma 5.2. *Für einen NEA $\mathcal{A}$ ist es determinitisch in polynomieller Zeit entscheidbar, ob er einen Grad der Uneindeutigkeit von* $1 \leq \mathrm{da}(\mathcal{A}) \leq k$ *aufweist.*

Beweis. Für Details siehe [4, Theorem 5.10]. □

6 Zusammenfassung

Diese Seminararbeit umfasst (nach einer kurzen Einleitung) im 2. Kapitel einführende Notationen, Definitionen und allgemeine Lemmata sowie Sätze zu NEAs. Danach werden Differenzengleichungen in Kapitel 3 vorgestellt und einige Eigenschaften dieser gezeigt. Darauf aufbauend wird in Kapitel 4 gezeigt wie zu eindeutigen NEAs Differenzengleichungen aufgestellt werden können und wie man diese nutzt, um deterministisch in polynomiell beschränkter Zeit das Inklusions- sowie Äquivalenzproblem zu entscheiden. Ferner wird ausgeführt wie man zu zwei eindeutigen NEAs ein Wort minimaler Länge berechnet, welches in der Sprache des einen, nicht aber in der des anderen Automaten ist. In Kapitel 5 wird die Idee aufgegriffen wie man die vorgestellten Methoden nutzen kann, um das Inklusionsproblem auch für NEAs mit festem Grad der Uneindeutigkeit deterministisch in polynomieller Zeit zu entscheiden.

In der Hauptquelle [4] wird angegeben, dass die Ergebnisse über eindeutige bzw. fest beschränkt uneindeutige Automaten auch für eindeutige reguläre Grammatiken und eindeutige reguläre Ausdrücke gelten. Sie führen weiter aus, dass diese Ergebnisse stark an der Grenze der effizienten Entscheidbarkeit liegen. Es wird gezeigt, dass alles was über einen festen Grad der Uneindeutigkeit hinaus geht CoNP-schwer ist [4, Theorem 6.5].

In vielen praktischen Fällen bspw. aus dem Compilerbau weisen die Sprachen und Grammatiken bzw. resultierenden Automaten jedoch einen festen Grad der Uneindeutigkeit auf und das Inklusionsproblem und somit auch die Äquivalenz lässt sich algorithmisch effizient entscheiden.
Dennoch gibt es Sprachen, die aufgrund ihrer Natur, *inhärent uneindeutig* sind, ein Beispiel dazu ist die kontextfreie Sprache $L = \{a^n b^n c^m d^m \mid m, n \geq 1\} \cup \{a^n b^m c^n d^m \mid n, m \geq 1\}$, zu der keine eindeutige kontextfreie Grammatik existiert [3, Theorem 4.7].

Die vorgestellten Methoden basieren auf der Eigenschaft des Grades der Uneindeutigkeit. Ob dieser Grad beschränkt ist, lässt sich nach Satz 2.5 für Automaten leicht zeigen. Diese Eigenschaft ist aber bspw. für beliebige kontextfreie Sprachen unentscheidbar [3, Theorem 8.9].

Literatur

[1] Allauzen, C., Mohri, M., and Rastogi, A. General algorithms for testing the ambiguity of finite automata. In *Developments in Language Theory*, M. Ito and M. Toyama, Eds., vol. 5257 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2008, pp. 108–120.

[2] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Algorithmen - Eine Einführung*. Oldenbourg Wissensch.Vlg, 2007.

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

[4] Stearns, R. E., and Hunt, H. B. I. On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. *SIAM J. Comput. 14*, 3 (1985), 598–611.

[5] Thomas, W. Applied automata theory, November 2005. Course Notes RWTH Aachen.

[6] Weber, A., and Seidl, H. On the degree of ambiguity of finite automata. *Theor. Comput. Sci. 88*, 2 (1991), 325–349.