Generalisierte Synchronisation und Schnittleerheit endlicher Automaten[1]

Petra Wolf[2]

Abstract: Endliche Automaten begegnen uns überall im Alltag, von der Steuerung der Kaffeemaschine über den Aufzug bis hin zu autonomen Systemen. Synchronisierende Wörter agieren hierbei als ein Software-Reset, eine Sequenz von Befehlen, die den Automaten von jedem Zustand aus in den gleichen Zustand überführt. Hierbei ist es wünschenswert eine gewisse Kontrolle über die Aktionen eines synchronisierenden Wortes zu behalten. Wir untersuchen daher die Komplexität der Frage, ob ein endlicher Automat ein synchronisierendes Wort besitzt, unter verschiedenen Einschränkungen, wie beispielsweise, dass das Wort aus einer bestimmten Sprache stammen muss, oder, dass die Sequenz der durchlaufenen Zustände eingeschränkt wird. Anschließend generalisieren wir das Konzept synchronisierender Automaten auf das mächtigere Berechnungsmodell des Kellerautomatens. Im letzten Teil der Arbeit untersuchen wir, inwiefern sich ein endlicher Automat vereinfachen lässt, indem wir die akzeptierte Sprache eines Automatens in den Schnitt simplerer Sprachen zerlegen.

1 Einleitung

Endliche Automaten bilden eines der fundamentalsten und simpelsten Berechnungsmodelle der theoretischen Informatik. Im Gegensatz zu Turing-Maschinen sind für endliche Automaten viele praktisch relevante Probleme entscheidbar und meist sogar effizient lösbar. Endliche Automaten kommen in der Steuerung vieler alltäglicher Maschinen vor und sind von großer Relevanz bei der Verifikation von Programmen, dem sogenannten Model-Checking. Die hier vorgestellte Dissertation befasst sich mit Generalisierungen des Synchronisationsproblems endlicher Automaten sowie ihrem Schnittleerheitsproblem.

Synchronisation ist ein wichtiges Konzept für viele Anwendungsbereiche: parallele und verteilte Programmierung, System- und Protokolltests, Informationscodierung, Robotik, und so weiter. Einige Aspekte der Synchronisation werden durch den Begriff des synchronisierenden Automaten erfasst; beispielsweise modellieren synchronisierende Automaten adäquat Situationen, in denen man ein bestimmtes System in einen bestimmten Zustand bringen muss, ohne a priori dessen aktuellen Zustand zu kennen.

Ein endlicher Automat wird synchronisierend genannt, wenn es ein Wort gibt, das ihn unabhängig vom Ausgangszustand in denselben Zustand überführt. Dieses Konzept wurde in den letzten sechs Jahrzehnten intensiv untersucht. Es ist verwandt mit der wohl berühmtesten offenen kombinatorischen Frage der Automatentheorie, die von Černý und

[1] Englischer Titel der Dissertation [Wo22]: „Generalized Synchronization and Intersection Non-Emptiness of Finite-State Automata“

[2] Universitetet i Bergen, HIB - Thormøhlens gate 55, 5006 Bergen, Norwegen, mail@wolfp.net.
Promotion an der Universität Trier, Fachbereich 4 - Informatikwissenschaften, Deutschland.

Starke in [Če64, St66] formuliert wurde. Die Černý-Vermutung besagt, dass jeder synchronisierende Automat mit n Zuständen durch ein Wort der Länge höchstens $(n-1)^2$ synchronisiert werden kann. Obwohl diese Schranke für mehrere Klassen von endlichen Automaten bewiesen wurde, ist der allgemeine Fall noch weitgehend offen. Die derzeit beste obere Schranke für die Länge eines synchronisierenden Wortes ist kubisch, und bis jetzt wurden nur geringe Fortschritte beim Reduzieren dieser Schranke gemacht, im Wesentlichen durch Verbesserung des multiplikativen Faktors vor dem kubischen Term.

Aufgrund der Bedeutung des Begriffs der synchronisierenden Wörter wurde in der Literatur eine große Anzahl von Verallgemeinerungen und Modifikationen betrachtet. Anstatt die gesamte Menge der Zustände zu synchronisieren, kann es von Interesse sein, nur eine Teilmenge der Zustände zu synchronisieren. Diese und verwandte Fragen wurden erstmals von Rystsov in [Ry83] betrachtet. Anstatt deterministische endliche Automaten zu betrachten, kann man alternativ den Begriff der Synchronisierbarkeit für nichtdeterministische endliche Automaten untersuchen. Der Begriff der Synchronisierbarkeit lässt sich auf natürliche Weise auf partiell definierte Übergangsfunktionen übertragen, wobei ein synchronisierender Automat, der undefinierte Übergänge vermeidet, *vorsichtig synchronisierend* genannt wird. Während für deterministische endliche Automaten die Frage der Synchronisierbarkeit in Polynomialzeit lösbar ist, ist sie in allen genannten Verallgemeinerungen PSPACE-vollständig. Diese Tendenz ist auch bei den Generalisierungen zu beobachten, die in dieser Arbeit eingeführt werden.

Das klassische *Synchronisationsproblem* fragt für einen gegebenen deterministischen endlichen Automaten, ob es für diesen Automaten ein *synchronisierendes Wort* gibt, d.h. ein Eingabewort, das alle Zustände des Automaten in einen einzigen Zustand überführt. Die Idee, einen Automaten durch das Lesen eines Wortes, ausgehend von einem beliebigen Zustand, in einen wohldefinierten Zustand zu bringen, kann als Implementierung eines Software-Resets angesehen werden. Aus diesem Grund wird ein synchronisierendes Wort manchmal auch als *Reset-Wort* bezeichnet. Im Folgenden werden wir einige Verallgemeinerungen des Synchronisationsproblems vorstellen. Unser Hauptziel ist dabei die Untersuchung der Komplexität der vorgestellten Probleme. Dabei erhalten wir nicht nur klassische Komplexitätsergebnisse, sondern betrachten auch parametrisierte Versionen des Problems.

2 Synchronisation unter Beschränkungen

Bei der Betrachtung synchronisierender Automaten lässt man normalerweise zu, dass ein synchronisierendes Wort ein beliebiges Wort über dem Alphabet des entsprechenden Automaten ist. In der Realität können die verfügbaren Befehle jedoch bestimmten Einschränkungen unterliegen; so ist es zum Beispiel plausibel anzunehmen, dass eine Reset-Sequenz immer mit einem bestimmten Befehl beginnt und endet, der den Automaten zunächst in einen “Reset”-Modus schaltet und ihn dann in seinen üblichen Modus zurückversetzt. In seiner einfachsten Form kann das Umschalten zwischen “normalem” Modus und “Reset-” (Synchronisations-)Modus als ab^*a modelliert werden. Diese Beschränkungen werden durch einen (festen) endlichen Automaten (den sogenannten be-

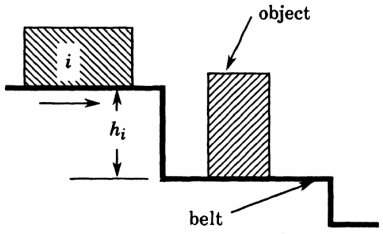

Abb. 1: Veranschaulichung von Modifikatoren als Stufen auf einem Fließband in der Einführungsarbeit von Natarajan [Na86]. Hier entspricht ein Zustand einer möglichen Orientierung eines Teils und ein Übergang eines Buchstabens a vom Zustand q entspricht der Anwendung des Modifikators, der a entspricht, auf ein Teil in der Orientierung q.

schränkenden Automaten) definiert, der eine reguläre Sprache R beschreibt, und die Frage ist, ob ein gegebener endlicher Automat A irgendein synchronisierendes Wort aus R hat.

Einschränkungen dieser Art, die durch eine feste reguläre Sprache modelliert werden können, die die Menge der potenziellen synchronisierenden Wörter für einen Eingabeautomaten einschränkt, werden *Regular Constraints* genannt und sind das Thema der ersten Forschungsarbeit in dieser Dissertation. Hier wird eine vollständige Analyse der Komplexität des Synchronisationsproblems unter regulären Constraints für beschränkende Automaten mit zwei Zuständen und höchstens drei Buchstaben durchgeführt.

Abgesehen von Software-Reset-Wörtern, ist eine der ältesten Anwendungen des intensiv untersuchten Begriffs der synchronisierenden Automaten das Problem des Entwurfs von Fließbändern, die ein Objekt in einer (mangels teurer Sensoren) unbekannten Orientierung in eine definierte Orientierung überführen. In seiner Pionierarbeit modellierte Natarajan [Na86] solche Fließbänder als deterministische vollständige Automaten (siehe Abbildung 1), bei denen ein Zustand einer möglichen Orientierung eines Teils entspricht und ein Übergang eines Buchstabens a vom Zustand q der Anwendung des a entsprechenden Modifikators auf ein Teil in Orientierung q entspricht. Viele verschiedene Klassen von Automaten sind seitdem hinsichtlich ihres Synchronisationsverhaltens untersucht worden. Die ursprüngliche Motivation, ein Fließband zur Orientierung von Teilen zu entwerfen, wurde in [TY15] wieder aufgegriffen, wo Türker und Yenigün den Entwurf einer Montagelinie modellierten, die wiederum ein Teil von einer unbekannten Ausrichtung in eine bekannte Ausrichtung überführt, wobei verschiedene Modifikatoren unterschiedliche Kosten verursachen.

Was bisher nicht berücksichtigt wurde, ist, dass verschiedene Modifikatoren unterschiedliche Auswirkungen auf die Teile haben können, und da wir die aktuelle Ausrichtung nicht kennen, möchten wir vielleicht die Chronologie der angewandten Modifikatoren einschränken. Wenn es sich bei dem Teil beispielsweise um eine Box mit aufklappbarem Laschen handelt, so bewirkt das Drehen der Box ein Öffnen der Laschen. Um die Laschen wieder zu schließen, könnte ein weiterer Modifikator erforderlich sein, z. B. ein niedriger Balken, der die Laschen streift und sie wieder schließt, wie in Abbildung 2 dargestellt.

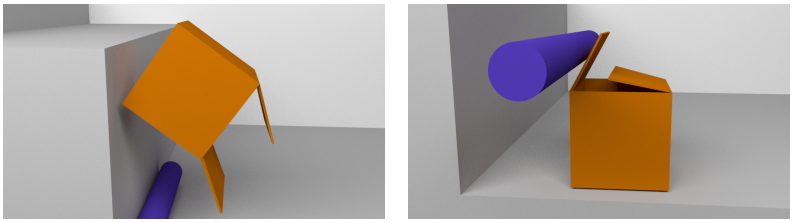

Abb. 2: Illustration einer Box mit Laschen, die sich öffnen, wenn die Box gedreht wird, zusammen mit einem Modifikator, der die Laschen schließt.

Die Spezifikation, dass die Box mit geschlossenen Laschen nach oben ausgeliefert werden soll, kann kodiert werden als: “Wenn sich die Box im Zustand *auf dem Kopf* befindet, muss sie sich später im Zustand *Laschen geschlossen*” befinden. Aber das hindert uns nicht daran, die Laschen wieder zu öffnen, also müssen wir präziser sein und kodieren: “Nachdem die Box zum letzten Mal im Zustand *auf dem Kopf* war, muss sie mindestens einmal den Zustand *Laschen geschlossen* einnehmen”.

Bedingungen dieser Art beschreiben *positive* Beschränkungen für die Zukunft, da sie den zukünftigen Besuch eines bestimmten Zustands verlangen. Aber auch die umgekehrte Situation des Verbots des zukünftigen Besuchs eines Zustands ist vorstellbar, wie unser nächstes Beispiel zeigt. Stellen wir uns wieder die Box mit Laschen vor, aber diesmal enthält die Box zunächst etwas Wasser. Wir möchten die Box in einer bestimmten Ausrichtung mit geöffneten Laschen haben, aber das Wasser soll während der Ausrichtung nicht auslaufen. Wir haben einen Modifikator, der die Laschen öffnet, und einen Modifikator, der die Box dreht. Wir wollen natürlich nicht, dass die Box nach dem Öffnen der Laschen nach unten zeigt. Also kodieren wir: “Sobald der Zustand *Laschen offen* erreicht ist, sollte der Zustand *nach unten gerichtet* nie wieder betreten werden”.

Bedingungen der oben diskutierten Formen, die das dynamische Verhalten eines synchronisierenden Wortes einschränken, stehen im Mittelpunkt der zweiten Forschungsarbeit, die in der Dissertation vorgestellt wird. Dort werden die Einschränkungen an den Entwurf eines Fließbands dadurch modelliert, dass wir einen gegebenen endlichen Automaten mit einer Relation R erweitern und dann unterschiedliche Möglichkeiten betrachten, wie ein synchronisierendes Wort eine Ordnung auf den Zuständen des Automaten induzieren kann. Anschließend betrachten wir das Problem ein synchronisierendes Wort zu finden, dessen induzierte Zustandsordnung mit der gegebenen Relation R verträglich ist.

Diese Forschungsarbeit wurde 2021 mit dem Publikationspreis des Graduiertenzentrums der Universität Trier, Fachbereich IV, ausgezeichnet. Ein Video der Präsentation der Arbeit im Rahmen der Preisverleihung ist unter https://youtu.be/kmDPUmUt-R4 zu finden.

3 Synchronisation von Kellerautomaten

Die Überführung eines Automaten von einem beliebigen in einen wohldefinierten Zustand ist nicht nur für endliche Automaten relevant. Im zweiten Teil der Arbeit bewegen wir

uns weg von deterministischen endlichen Automaten hin zu allgemeineren deterministischen Kellerautomaten. Ein Kellerautomat besteht aus einem endlichen Automaten, der zusätzlich mit einem Keller ausgestattet ist. Auf dem Keller können beliebig viele Zeichen gespeichert werden, jedoch kann immer nur das oberste Zeichen des Kellers gelesen werden. Neue Zeichen werden hierbei von oben auf den Keller gelegt und beim Lesen wird das oberste Zeichen vom Keller entfernt.

Was soll ein synchronisierendes Wort im Zusammenhang mit Kellerautomaten bedeuten? Mikami und Yamakami führten in [MY20] drei verschiedene Modelle für das Synchronisieren eines Kellerautomaten ein. Wie bei endlichen Automaten muss ein synchronisierendes Wort w die Zustände des Kellerautomaten synchronisieren, und zusätzlich müssen verschiedene Anforderungen an den Kellerinhalt eines jeden durch w induzierten Laufs erfüllt werden. So fordern wir beispielsweise, dass nach dem Lesen eines synchronisierenden Wortes: (a) die Keller aller Läufe leer sind; oder (b) dass der Kellerinhalt aller Läufe identisch (aber nicht unbedingt leer) ist; oder (c) dass der Kellerinhalt beim Eintritt in den synchronisierenden Zustand völlig irrelevant ist. Für diese drei Modelle von synchronisierenden Kellerautomaten wurden in [MY20] einige obere und untere Schranken für die maximale Länge des kürzesten synchronisierenden Wortes in Abhängigkeit von der Kellerhöhe gezeigt. Hier untersuchen wir diese drei Modelle aus einer komplexitätstheoretischen Perspektive. Wir zeigen, dass die Frage der Synchronisierbarkeit eines Kellerautomaten in allen drei Kellermodellen unentscheidbar ist.

Neben allgemeinen deterministischen Kellerautomaten untersuchen wir das Synchronisationsproblem weiterer Unterklassen von deterministischen Kellerautomaten, wie deterministische Zählautomaten, deterministische (partiell) blinde Zählautomaten, und Finite-Turn-Varianten hiervon. Wir zeigen, dass das Synchronisationsproblem für deterministische Zählerautomaten ebenfalls unentscheidbar ist. Das Problem wird für deterministische, partiell blinde Zählautomaten entscheidbar und ist PSPACE-vollständig für einige Arten von Finite-Turn-Varianten von Kellerautomaten.

4 Synchronisation sichtbarer Kellerautomaten

Ein weiteres Automatenmodell, bei dem die Zustandsmenge um eine möglicherweise unendliche Speicherstruktur erweitert wird, ist die Klasse der *Nested Word Automata* (NWA), bei der ein Eingabewort um eine passende Relation erweitert wird, die bestimmt, an welchem Paar von Positionen in einem Wort ein Symbol auf den Keller gelegt und vom Keller gelesen wird. Die Klasse der Sprachen, die von NWAs akzeptiert werden, ist identisch mit der Klasse der *Visibly Push-Down Sprachen* (VPL), die von *sichtbaren Kellerautomaten* (im Englischen Visibly Push-Down Automata, kurz VPDA) akzeptiert werden, und bildet eine eigene Unterklasse der deterministischen kontextfreien Sprachen. VPDAs wurden zuerst von Mehlhorn unter dem Namen *Input-Driven Push-Down Automata* untersucht und wurden in jüngerer Zeit durch die Arbeit von Alur und Madhusudan populär, die zeigten, dass VPLs mehrere wünschenswerte Eigenschaften mit regulären Sprachen teilen. In [CMS19] wurde das Synchronisationsproblem für NWAs untersucht. Dort wurde das Konzept der Synchronisation dahingehend verallgemeinert, dass alle Zustände in einen

einzigen Zustand gebracht werden, sodass bei allen Durchläufen der Keller nach dem Lesen des synchronisierenden Wortes leer ist (bzw. in seiner Startkonfiguration ist). Es wurde gezeigt, dass für NWAs das Synchronisationsproblem in Polynomialzeit lösbar ist.

Im nächsten Abschnitt der Dissertation untersuchen wir das Synchronisationsproblem für deterministische sichtbare Kellerautomaten und mehrere Unterklassen hiervon, wie deterministische sehr sichtbare Kellerautomaten (Very Visibly Push-Down Automata), deterministische sichtbare Zählerautomaten und Finite-Turn-Varianten davon. Wir möchten darauf hinweisen, dass sich trotz der Äquivalenz der jeweiligen Sprachklassen die Automatenmodelle von NWAs und VPDA unterscheiden und die Ergebnisse von [CMS19] nicht unmittelbar auf VPDAs übertragbar sind, da für NWAs ein Eingabewort mit einer Matching-Relation ausgestattet ist, die bei VPDAs fehlt. Im Allgemeinen kann die Komplexität des Synchronisationsproblems für verschiedene Automatenmodelle, die dieselbe Sprachklasse akzeptieren, unterschiedlich sein. Im Gegensatz zu dem in Polynomialzeit lösbaren Synchronisationsproblem für deterministische endliche Automaten ist beispielsweise das verallgemeinerte Synchronisationsproblem für endliche Automaten mit einem nicht-deterministischen Übergang PSPACE-vollständig, ebenso wie das Problem der vorsichtigen Synchronisation eines Automaten mit einem undefinierten Übergang.

Im Gegensatz zur Unentscheidbarkeit für allgemeine Kellerautomaten ist für sichtbare Kellerautomaten und die hier betrachteten Unterklassen das Synchronisationsproblem für alle drei Kellermodelle in EXPTIME und damit entscheidbar. Für sehr sichtbare Kellerautomaten und sichtbare Zählerautomaten sind die Synchronisationsprobleme für alle drei Kellermodelle sogar in Polynomialzeit lösbar. Wie das Synchronisationsproblem für NWAs im leeren Kellermodell, das in [CMS19] betrachtet wurde, stellen wir fest, dass das Synchronisationsproblem für VPDAs im leeren Kellermodell in Polynomialzeit lösbar ist, während die Synchronisation von VPDAs im gleichen und beliebigen Kellermodell mindestens PSPACE-schwer ist. Wenn die Anzahl der Turns im Höhenprofil des Kellers, die durch ein synchronisierendes Wort verursacht werden, eingeschränkt wird, wird das Synchronisationsproblem für alle betrachteten Automatenmodelle für $n > 0$ PSPACE-schwer und ist nur in P für $n = 0$ im leeren Kellermodell. Des Weiteren führen wir Varianten von Synchronisationsproblemen ein, bei denen sich die Komplexität des gleichen und beliebigen Kellermodells unterscheidet. Für die zuvor betrachteten Synchronisationsprobleme hatten diese beiden Kellermodelle immer den gleichen Komplexitätsstatus.

5 Zusammenhang zwischen Synchronisation und Schnittleerheit

Das Schnittleerheitsproblem endlicher Automaten ist eines der grundlegendsten und am besten untersuchten Probleme im Zusammenspiel zwischen Algorithmen, Komplexitätstheorie und Automatentheorie. Dabei gilt es für eine gegebene endliche Menge von endlichen Automaten $\{A_1, A_2, \ldots, A_n\}$ über einem gemeinsamen Alphabet Σ, zu bestimmen, ob es ein Wort $w \in \Sigma^*$ gibt, das von jedem der Automaten in der Menge akzeptiert wird. Oder anders gesagt, die Frage ist, ob $\mathscr{L}(A_1) \cap \mathscr{L}(A_2) \cap \cdots \cap \mathscr{L}(A_n) \neq \emptyset$.

Das Schnittleerheitsproblem ist PSPACE-vollständig und eng mit synchronisierenden Automaten verwandt. Einerseits kann die Frage, ob ein Automat A mit n Zuständen in einen

Zustand q synchronisiert werden kann, als Schnittleerheitsproblem von n Automaten modelliert werden, wobei jeder Automat eine Kopie von A mit q als einzigem Endzustand und jeweils einem anderen Startzustand ist. Die Schnittmenge der von den n Automaten akzeptierten Sprachen enthält dann genau die Wörter, die A in die Zustandsmenge $\{q\}$ synchronisieren. Andererseits kann das Schnittleerheitsproblem für endliche Automaten auf das vorsichtige Synchronisierungsproblem reduziert werden, das fragt, ob es für einem *partiellen* endlichen Automaten A, ein Wort w gibt, das A synchronisiert, ohne einen undefinierten Übergang beim Lesen von w auf irgendeinem Pfad zu nehmen. Martyugin hat in [Ma14] gezeigt, dass das vorsichtige Synchronisierungsproblem PSPACE-vollständig ist, selbst wenn der Eingabeautomat nur einen undefinierten Übergang hat, und zwar durch eine Reduktion vom Schnittleerheitsproblem. Aufgrund dieser natürlichen Beziehung befassen sich die beiden verbleibenden Forschungsarbeiten im dritten Teil der Dissertation mit dem Schnittleerheitsproblem.

6 Schnittleerheit sternfreier Sprachen

In der fünften Forschungsarbeit wird die Komplexität des Schnittleerheitsproblems untersucht, unter der Annahme, dass die von den Eingabeautomaten akzeptierten Sprachen zu einer bestimmten Ebene der Straubing-Thérien-Hierarchie oder der Cohen-Brzozowski-Dot-Depth-Hierarchie gehören. Die betrachteten Sprachen sind stark eingeschränkt, in dem Sinne, dass beide unendlichen Hierarchien vollständig in der Klasse der sternfreien Sprachen enthalten sind, einer Klasse von Sprachen, die durch Ausdrücke dargestellt werden können, die Vereinigung, Konkatenation und Komplementierung verwenden, aber *keine* Kleene-Sternoperation erlauben. Trotz dieser Einschränkung ist es schwierig die Sprachen, die zu einzelnen Ebenen einer der beiden Hierarchien gehören, zu charakterisieren; so ist die Frage, ob die von einem gegebenen Automaten akzeptierte Sprache zu einem festen Level einer der beiden Hierarchien gehört, bis auf wenige untere Stufen ein ungelöstes Problem. Automaten, die Sprachen auf niedrigeren Ebenen dieser Hierarchien akzeptieren, treten in einer Vielzahl von Anwendungen auf, wie beispielsweise Model-Checking, wo das Schnittleerheitsproblem von grundlegender Bedeutung ist.

Wir betrachten hier die Frage, wie sich die Komplexität des Schnittleerheitsproblems ändert, wenn wir uns in den Ebenen der Straubing-Thérien- und der Dot-Depth-Hierarchie nach oben bewegen. Verändert sich die Komplexität dieses Problems nur allmählich, wenn wir die Komplexität der Eingabesprachen erhöhen? Tatsächlich zeigen wir, dass dies nicht der Fall ist, und dass die Komplexitätslandschaft für das Schnittleerheitsproblem bereits durch die allerersten Ebenen einer der beiden Hierarchien bestimmt wird. So zeigen wir, dass bereits für die erste Ebene der Dot-Depth-Hierarchie und die zweite Ebene der Straubing-Thérien-Hierarchie das Schnittleerheitsproblem PSPACE-vollständig ist. Weiter zeigen wir, dass für die Ebenen Null und ein Halb der Dot-Depth-Hierarchie sowie die Ebenen Eins und drei Halbe der Straubing-Thérien-Hierarchie das Schnittleerheitsproblem NP-hart ist. Zuletzt zeigen wir, dass das Schnittleerheitsproblem für nichtdeterministische und deterministische endliche Automaten, die Sprachen der Ebene ein Halb der Straubing-Thérien-Hierarchie akzeptieren, NLOG- bzw. LOG-vollständig unter AC^0-Reduktionen ist. Als ein Nebenprodukt zeigen wir unserer Kenntnis nach die erste expo-

nentielle Trennung zwischen der Zustandskomplexität von allgemeinen nichtdeterministischen Automaten und der von partiell geordneten nichtdeterministischen Automaten.

7 Zerlegungen endlicher Automaten

Wir können nicht nur das Synchronisationsproblem als ein Schnittleerheitsproblem von endlichen Automaten darstellen, sondern auch einige minimale endliche Automaten selbst als Schnittmenge von endlich vielen *kleineren* endlichen Automaten darstellen. In diesem Sinne *zerlegen* wir den größeren Automaten in eine Menge kleinerer Automaten, sodass deren Schnittmenge die gleiche Sprache akzeptiert wie der ursprüngliche Automat. Wenn wir nur *minimale* endliche Automaten betrachten, so ist es eine durch die akzeptierte Sprache bestimmte Eigenschaft, ob eine solche Zerlegung möglich ist oder nicht.

Kompositionalität ist ein grundlegender Begriff in zahlreichen Bereichen der Informatik. Dieses Prinzip lässt sich wie folgt zusammenfassen: Jedes System sollte durch das Zusammensetzen einfacher Teile so entworfen werden, dass die Bedeutung des Systems aus der Bedeutung seiner Teile und der Art und Weise, wie sie kombiniert werden, abgeleitet werden kann. Dies ist beispielsweise ein entscheidender Aspekt der modernen Softwareentwicklung: Ein in einfache Module aufgeteiltes Programm ist schneller zu kompilieren und leichter zu warten. Auch in der theoretischen Informatik ist die Kompositionalität von entscheidender Bedeutung: Sie wird eingesetzt, um das Problem der Zustandsexplosion zu vermeiden, das normalerweise bei der Kombination paralleler Prozesse auftritt, und um das Problem der Skalierbarkeit von Problemen mit hoher theoretischer Komplexität zu bewältigen. In der letzten vorgestellten Forschungsarbeit untersuchen wir Kompositionalität im Rahmen formaler Sprachen: Wir zeigen, wie man Sprachen vereinfachen kann, indem man sie in größere Sprachen zerlegt, die von kleineren Automaten akzeptiert werden. Dies ist durch *Model-Checking* Probleme motiviert. Das LTL-Model-Checking Problem fragt zum Beispiel bei einer linearen temporalen Logikformel φ und einem endlichen Automaten M, ob jede Ausführung von M φ erfüllt. Dieses Problem ist entscheidbar, hat aber eine hohe theoretische Komplexität (PSPACE) in Bezug auf die Größe von φ. Hier kommt die Kompositionalität ins Spiel: Wenn wir die Spezifikationssprache in eine Schnittmenge einfacher Sprachen zerlegen können, d. h. φ in eine Konjunktion $\varphi = \varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_k$ kleiner Spezifikationen zerlegen, genügt es zu prüfen, ob alle φ_i einzeln erfüllt sind.

Hier konzentrieren wir unsere Untersuchung auf endliche *Permutationsautomaten*, d.h. endliche Automaten, deren Übergangsmonoide Gruppen sind: Jeder Buchstabe induziert eine Eins-zu-Eins-Abbildung der Zustandsmenge auf sich selbst. Diese endlichen Automaten werden auch als *reversible* endliche Automaten bezeichnet. Reversibilität ist stärker als Determinismus: Diese mächtige Eigenschaft erlaubt es, deterministisch zwischen den Schritten einer Berechnung hin und her zu navigieren. Dies ist besonders wichtig für die Untersuchung der Physik von Berechnungen, da Irreversibilität zu Energieverlusten führt. Diese Fähigkeit führt jedoch bei endlichen Automaten zu einem Verlust an Ausdruckskraft: Im Gegensatz zu leistungsfähigeren Modellen (z. B. Turing-Maschinen) sind reversible endliche Automaten weniger ausdrucksstark als allgemeine endliche Automaten.

Das Problem, das für diese Arbeit von Interesse ist, ist das sogenannte Zerlegungsproblem, das für einen gegebenen endlichen Automaten A fragt, ob er zusammengesetzt ist, d.h., ob es eine endliche Menge von endlichen Automaten $\{A_1, A_2, \ldots, A_n\}$ gibt, sodass $\mathscr{L}(A) = \mathscr{L}(A_1) \cap \mathscr{L}(A_2) \cap \cdots \cap \mathscr{L}(A_n)$ und jeder der Automaten in der Schnittmenge echt weniger Zustände hat als A. Das Zerlegungsproblem für endliche Automaten wurde erstmals von Kupferman und Moscheiff [KM15] eingeführt. Sie bewiesen, dass es in EXPSPACE entscheidbar ist, ließen aber die genaue Komplexität offen: die beste bekannte untere Schranke ist Härte für NLOG. Sie gaben effizientere Algorithmen für eingeschränkte Automatenklassen an: einen PSPACE-Algorithmus für *Permutationsautomaten* und einen Polynomialzeit-Algorithmus für *normale* Permutationsautomaten, eine Klasse von endlichen Automaten, die alle *kommutativen* Permutationsautomaten enthält. Kürzlich wurde bewiesen, dass das Zerlegungsproblem für endliche Automaten mit einem unären Alphabet in logarithmischem Platz lösbar ist [JKM20].

In der hier vorgestellten Forschungsarbeit erweitern wir den Bereich der Instanzen, für die das Zerlegungsproblem praktisch lösbar ist. Wir konzentrieren uns auf Permutationsautomaten und schlagen neue Techniken vor, die die bekannten Komplexitäten verbessern. Wir geben einen NP-Algorithmus für Permutationsautomaten an und zeigen, dass die Komplexität direkt mit der Anzahl der nicht-akzeptierenden Zustände verknüpft ist. Genauer gesagt erhalten wir einen parametrisierten Algorithmus mit dem Parameter: Anzahl der nicht-akzeptierenden Zustände. Außerdem beweisen wir, dass Permutationsautomaten, die eine Primzahl von Zuständen haben, nicht zerlegt werden können. Des Weiteren betrachten wir *kommutative* Permutationsautomaten, bei denen das Zerlegungsproblem bereits als effizient lösbar bekannt war, und senken die Komplexität von P auf NLOG weiter ab, und sogar LOG, wenn die Größe des Alphabets konstant ist. Obwohl es einfach ist, zu entscheiden, ob ein kommutativer Permutationsautomat zusammengesetzt ist, zeigen wir, dass in dieser Automatenklasse immer noch reichhaltige und komplexe Verhaltensweisen auftreten: Es gibt Familien von zusammengesetzten kommutativen Permuationsautomaten, die polynomiell viele Faktoren für eine Zerlegung benötigen. Genauer gesagt konstruieren wir eine Familie $(A_n^m)_{m,n\in\mathbb{N}}$ von zusammengesetzten endlichen Automaten, sodass A_n^m ein endlicher Automat der Größe n^m ist, der in $(n-1)^{m-1}$ Faktoren zerlegt werden kann, nicht jedoch in $(n-1)^{m-1}-1$. Vor diesem Ergebnis waren nur Familien von zusammengesetzten endlichen Automaten bekannt, die sub-logarithmisch viele Faktoren benötigen.

Schließlich untersuchen wir ein beschränktes Zerlegungsproblem. Für praktische Zwecke ist es nicht wünschenswert, viele Faktoren in einer Zerlegung zu haben, da der Umgang mit einer großen Anzahl kleiner endlicher Automaten am Ende komplexer sein kann als der Umgang mit einem einzigen endlichen Automaten moderater Größe. Das beschränkte Zerlegungsproblem bewältigt diese Schwierigkeit, indem es die Anzahl der in den Zerlegungen zulässigen Faktoren begrenzt. Wir zeigen, dass diese Flexibilität ihren Preis hat: Überraschenderweise ist dieses Problem NP-vollständig für kommutative Permutationsautomaten, eine Automatenklasse, für die das Zerlegungsproblem einfach ist. Wir zeigen auch, dass das beschränkte Zerlegungsproblem für allgemeine endliche Automaten in PSPACE liegt und für unäre endliche Automaten in LOG liegt.

Literaturverzeichnis

[Če64] Černý, Ján: Poznámka k homogénnym experimentom s konečnými automatmi. Matematicko-fyzikálny časopis, 14(3):208–216, 1964.

[CMS19] Chistikov, Dmitry; Martyugin, Pavel; Shirmohammadi, Mahsa: Synchronizing Automata over Nested Words. Journal of Automata, Languages and Combinatorics, 24(2-4):219–251, 2019.

[JKM20] Jecker, Ismaël; Kupferman, Orna; Mazzocchi, Nicolas: Unary Prime Languages. In: MFCS. Jgg. 170 in LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, S. 51:1–51:12, 2020.

[KM15] Kupferman, Orna; Mosheiff, Jonathan: Prime languages. Information and Computation, 240:90–107, 2015.

[Ma14] Martyugin, Pavel V.: Computational Complexity of Certain Problems Related to Carefully Synchronizing Words for Partial Automata and Directing Words for Nondeterministic Automata. ACM Transactions on Computer Systems, 54(2):293–304, 2014.

[MY20] Mikami, Eitatsu; Yamakami, Tomoyuki: Synchronizing pushdown automata and reset words. IEICE Technical Report; IEICE Tech. Rep., 119(433):57–63, 2020.

[Na86] Natarajan, B. K.: An Algorithmic Approach to the Automated Design of Parts Orienters. In: FOCS. IEEE Computer Society, S. 132–142, 1986.

[Ry83] Rystsov, Igor K.: Polynomial Complete Problems in Automata Theory. Information Processing Letters, 16(3):147–151, 1983.

[St66] Starke, Peter H.: Eine Bemerkung über homogene Experimente. Elektronische Informationsverarbeitung und Kybernetik, 2(4):257–259, 1966.

[TY15] Türker, Uraz Cengiz; Yenigün, Hüsnü: Complexities of Some Problems Related to Synchronizing, Non-Synchronizing and Monotonic Automata. International Journal of Foundations of Computer Science, 26(1):99–122, 2015.

[Wo22] Wolf, Petra: Generalized Synchronization and Intersection Non-Emptiness of Finite-State Automata. Dissertation, University of Trier, Germany, 2022.

Petra Wolf wurde 1992 in Tübingen geboren, wo sie auch 2016 ihr Bachelorstudium in Informatik abschloss. Im Masterstudium legte sie den Schwerpunkt auf Theoretische Informatik und erlangte 2018 von der Eberhard Karls Universität Tübingen den Master of Science in Informatik mit Auszeichnung. Anschließend promovierte sie an der Universität Trier unter der Betreuung von Prof. Dr. Henning Fernau im DFG-geförderten Forschungsprojekt „Moderne Komplexitätsaspekte formaler Sprachen“. Während ihrer Promotionszeit erhielt sie 2021 den Publikationspreis des Graduiertenzentrum der Universität Trier im Fachbereich IV und, gemeinsam mit dem Team der Vortragsreihe #LecturesForFuture, den Lehrpreis der Universität Trier 2020/21 in der Kategorie „Grenzen überwunden“. Im Januar 2022 schloss sie ihre Promotion in Theoretischer Informatik mit summa cum laude ab. Seit August 2022 ist sie als Postdoktorandin an der Universität Bergen in Norwegen.