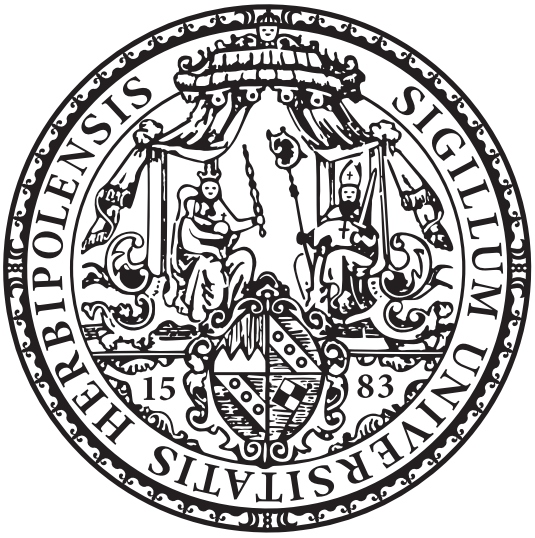
Bachelorarbeit

Existenz (I-)optimaler Beweissysteme für schwache Berechnungsmodelle

Linus Pleyer

Abgabedatum:	8. Februar 2024
Betreuer:	Prof. Dr. Christian Glaßer
	Fabian Egidy, M.Sc.



Julius-Maximilians-Universität Würzburg
Lehrstuhl für Informatik I
Algorithmen und Komplexität

Zusammenfassung

In der Beweiskomplexität ist bisher nicht geklärt, ob außerhalb von NP Sprachen mit (p-)optimalem Beweissystem existieren. Diese Arbeit soll dazu beitragen zu verstehen, wieso diese Existenzfragen schwierig sind. Wir erweitern dazu die übliche Definition von Beweissystemen von Cook und Reckhow [CR79], indem wir andere Berechnungskomplexitäten betrachten. Für diese modifizierte Definition kann dann das Verhalten der Fragen untersucht werden. Speziell für auf logarithmischen Raum beschränkte Beweissysteme untersuchen wir diese Existenzfragen anhand einer Reihe von Überlegungen, die ursprünglich bei Messner [Mes01] für die übliche Definition erschienen sind. Wir entwickeln außerdem eine Technik, mit der wir FP-Beweissysteme in FL-Beweissysteme übersetzen können. Das setzt die Existenzfragen für die beiden Definitionen in Verbindung und eröffnet neue Wege, die ursprünglichen offenen Fragen zu untersuchen.
Einen Überblick über die wichtigsten Resultate liefert die folgende Auflistung:

- Jede Sprache in L hat ein l-optimales FL-Beweissystem. Jede Sprache in NP hat ein optimales FL-Beweissystem (Satz 3.2).
- Sprachen mit (l-)optimalem FL-Beweissystem sind unter $\leq_m^{\log}, \cap$ und $\times$ abgeschlossen (Satz 3.4, Satz 3.6).
- Sprachen mit l-optimalem FL-Beweissystem haben einen raumoptimalen Akzeptor. Für Sprachen mit padding-Eigenschaft gilt auch die Rückrichtung (Satz 3.7).
- Für beliebig komplexe raumkonstruierbare $s : \mathbb{N} \to \mathbb{N}$ existieren Sprachen $\notin \mathrm{DSPACE}(\mathcal{O}(s))$ mit raumoptimalem deterministischen Akzeptor (Satz 3.22).
- Es existieren beliebig komplexe Sprachen ohne (l-)optimales FL-Beweissystem. Insbesondere haben $\leq_m^{\log}$-harte Mengen für $\mathrm{DSPACE}(s) \supsetneq \mathrm{L}$ kein l-optimales FL-Beweissystem (Satz 3.16).
- Das Standardbeweissystem für GAP ist genau dann l-optimal, wenn L = NL (Satz 4.2).
- $A \subseteq \Sigma^*$ hat ein optimales FL-Beweissystem $\iff$ A hat ein optimales FP-Beweissystem (Satz 4.4).
- $A \subseteq \Sigma^*$ hat ein l-optimales FL-Beweissystem $\implies$ A hat ein p-optimales FP-Beweissystem (Satz 4.8).

Inhaltsverzeichnis

1. Einführung

Das Beweisen hat in der Mathematik einen zentralen Stellenwert. Indem man Beweise liefert, will man sich von der Korrektheit von Behauptungen überzeugen und diese Korrektheit mit anderen kommunizieren. Dabei ist es nicht immer einfach, für eine wahre Aussage auch einen Beweis zu finden. Ein berühmtes Beispiel ist der Beweis zum großen Fermatschen Satz. Dieser blieb trotz des großen Interesses über Jahrhunderte unbewiesen und wurde erst 1994 von Andrew Wiles gefunden. Ein weiteres bemerkenswertes Beispiel ist der Beweis für LIN = coLIN. Lange Zeit war unklar, ob diese Beziehung gilt. Im Jahr 1987 wurde dieses Problem dann unabhängig von Robert Szelepcsényi und Neil Immerman gelöst. Dieser Beweis ist deshalb bemerkenswert, da er überraschend einfach ist. Obwohl die Antwort auf die Frage über Jahrzehnte unbekannt war, war ein knapper und einfacher Beweis möglich. Auch lange Zeit offene Probleme sind also nicht zwingend nur mit sehr komplizierten Beweisen lösbar.

Unabhängig davon, wie schwierig ein Beweis auffindbar war, sollte er von anderen leicht überprüft werden können. Mathematische Argumente sind dafür so aufgebaut, dass feste Regeln auf bereits verifizierte Wahrheiten angewandt werden, um neue Aussagen zu folgern. Dabei können die einzelnen Regelanwendungen sehr leicht nachvollzogen werden, da jede Regel nur auf einer konstanten Anzahl an vorhergehenden Wahrheiten aufbaut. Es ergibt sich eine Reihe an aufeinanderfolgenden Regelanwendungen, die separat voneinander sehr lokal und ohne Schwierigkeit geprüft werden können. Um dann einen gesamten Beweis zu verstehen und zu prüfen, kann ein Mathematiker den Referenzen auf vorhergehende Resultate folgen und auch deren Korrektheit leicht nachvollziehen. So ergibt sich insgesamt die Korrektheit des gesamten Beweises.

Dieses Bild ist die grundlegende Motivation der vorliegenden Arbeit. Der üblicherweise in der Beweiskomplexität verwendete Begriff von „leichter“ Überprüfbarkeit beschränkt nämlich nur die Zeit, die ein Prüfer zum verstehen braucht. Dabei kann aber auch zulässig sein, dass der Prüfer in dieser Zeit weitere Notizen machen muss, um Lücken in den Argumenten zu füllen. Diese Notizen könnten demnach auch die Länge des gegebenen Beweises übertreffen. Unsere Anschauung führt uns jedoch zu Beweisen, für deren Prüfung wir uns nur *Positionen* im Beweis merken müssen, um kurzzeitig zu Zwischenresultaten zurückzuspringen. Der Prüfer soll sich nicht signifikant mehr merken müssen. Das bedeutet, dass der Beweis bereits so ausführlich geschrieben sein muss, sodass ausführliche Notizen eines Prüfers nicht erforderlich sind.

Wir betrachten in dieser Arbeit nicht menschliche Prüfer, sondern wollen die Verifikation von Beweisen algorithmisch durchführen. Um den Wahrheitsbegriff für Algorithmen greifbar zu machen, stellen wir alle möglichen Behauptungen als Folge von Zeichen dar. Die Menge der Zeichenketten aller wahren Aussagen bilden dann eine Teilmenge dieser Zeichenfolgen, wir können die Menge der wahren Elemente daher als eine Sprache

auffassen. Unser Algorithmus muss dann nur Aussagen der Form „$x \in A$“ überprüfen, denn eine Behauptung ist genau dann wahr, wenn das entsprechende Wort in A enthalten ist. Wir brauchen außerdem ein Verständnis davon, was Effizienz von Algorithmen bedeutet. Dazu betrachten wir, wie in der Komplexitätstheorie üblich, den Bedarf an Speicher und Zeit abhängig von der Eingabelänge. Für ein System, dessen Beweise noch nicht ausreichend effizient geprüft werden können, könnten wir die Beweise also künstlich verlängern. Dadurch stünden dem Algorithmus mehr Ressourcen zur Verfügung, um auch die Beweissuche selbst durchzuführen. Wir haben aber bereits eine Trennung zwischen den Schwierigkeiten von Beweissuche und Verifikation beschrieben. Es könnte also sein, dass nach einem anderen Verfahren Beweise angegeben werden können, in dem diese künstliche Verlängerung nicht benötigt wird und das so deutlich kürzere Beweise zulässt. Im Längenunterschied der Beweise dieser Systeme können wir den eingangs beobachteten Unterschied zwischen der Suche nach einem Beweis und dem Prüfen von Beweisen wiedererkennen.

Möchte man Beweissysteme vergleichen, so ist es naheliegend die Beweislängen zu vergleichen. Wir interessieren uns in dieser Arbeit insbesondere für beste Beweissysteme, also Beweissysteme in denen alle wahren Aussagen kürzestmögliche Beweise besitzen. Wir nennen solche Beweissysteme *optimal*. Es ist unklar, für welche Sprachen solche Systeme existieren. Wir können nun die Frage stellen, ob beliebig komplexe Sprachen existieren, die ein solches optimales Beweissystem haben. Eine negative Antwort auf diese Frage würde bedeuten, dass nur gewisse „einfache“ Sprachen optimale Beweissysteme aufweisen können und wir für komplexere Sprachen nicht darauf hoffen können, dass ein einzelnes Beweissystem bereits die bestmöglichen Ergebnisse erzielt. Umgekehrt können wir auch fragen, ob es überhaupt beliebig komplexe Sprachen gibt, die *kein* optimales Beweissystem haben. Würde diese Frage negativ beantwortet, so hätten wir automatisch auch die erste Frage positiv beantwortet. Wir wissen also, dass mindestens eine der beiden Fragen eine positive Antwort haben muss.

Messner [Mes01] hat in seiner Doktorarbeit solche Fragen untersucht. Er findet beliebig komplexe Sprachen ohne optimale Beweissysteme, er kann jedoch nicht klären, ob außerhalb von NP Sprachen mit optimalem Beweissystem existieren. Er betrachtet außerdem eine weitere Technik, mit der Beweissysteme nach schärferen Kriterien verglichen werden können. Hier wird nicht nur die Beweislänge verglichen, sondern es wird sogar gefordert, dass der Beweis eines Systems effizient in den Beweis des anderen übersetzt werden kann. Ein Beweissystem, welches in jedes andere übersetzt werden kann, heißt auch *p-optimal*. Hier stellen sich genauso die Fragen nach Existenz von Sprachen außerhalb von NP mit p-optimalem Beweissystem. Auch hier kann Messner zeigen, dass beliebig komplexe Sprachen ohne p-optimales Beweissystem existieren, die zweite Frage bleibt jedoch erneut unklar. Messner kann diese Fragen nur unter Annahme von Kollapsvermutungen beantworten. Da er nicht zeigen kann, dass diese Annahmen auch notwendig sind, könnte es aber möglich sein, diese Bedingungen zu verbessern.

Wir wollen in dieser Arbeit einen Beitrag zur Untersuchung solcher Grenzen liefern. Wir ändern dafür die übliche Definition von Beweissystemen ab und schränken die Berechnungskomplexität von Beweissystemen stärker ein. Da p-Simulation in diesem Kontext keine konstruktive Übersetzung mehr liefert, die Übersetzung kann also nicht selbst

als Teil eines Beweissystems durchgeführt werden, führen wir hierfür den stärkeren Begriff der l-Simulation ein. Für die neuen scheinbar schwächeren Beweissysteme untersuchen wir dann, ob beliebig komplexe Sprachen mit optimalem bzw. l-optimalem FL-Beweissystem und analog beliebig komplexe Sprachen ohne optimales bzw. l-optimales FL-Beweissystem existieren. Wir interessieren uns dabei besonders dafür, ob diese Existenzfragen durch die stärkere Einschränkung der Beweissysteme leichter werden. Anhang A zeigt, dass ein solcher Effekt für eine ausreichend starke Einschränkungen tatsächlich auftritt. Wir werden jedoch auch zeigen, dass für weniger starke Einschränkungen die Fragen noch eng mit den Varianten bei Messner zusammenhängen und diese teilweise sogar implizieren.

Im Rest dieses Kapitels widmen wir uns einer allgemeinverständlichen Erklärung von Beweissystemen. Im Sinne der Lesbarkeit verzichten wir dabei auf technische Details und Definitionen. Kapitel 2 soll dann dazu dienen, diese Anschauungen mathematisch präzise zu machen. In Kapitel 3 sind Beweise zu finden, die bereits bei Messner [Mes01] geführt wurden und die durch kleine Modifikationen in den Kontext unserer schwächeren Beweissysteme gebracht werden können. Das ist nicht für alle Beweise möglich. In Kapitel 4 versuchen wir dann die Besonderheiten unserer Beweissysteme auszunutzen und folgern insbesondere einen engen Zusammenhang zwischen den Definitionen von Beweissystemen. Diese Arbeit soll einen Beitrag leisten, die Schwierigkeiten der gestellten Existenzfragen besser zu verstehen, indem das Verhalten dieser Fragen unter leichter Modifikation der zugrundeliegenden Definitionen untersucht wird.

1.1. Beweissysteme

Beweise in der Mathematischen Logik Das Gebiet der Mathematischen Logik hat den Begriff von Beweisen präzisiert. Üblicherweise leitet man hier eine Aussage nach fest definierten Regeln aus anderen Aussagen (Axiomen) her. Welche Regeln und Axiome genau verwendet werden, hängt vom verwendeten System ab. Ein Beispiel dafür ist das *Kalkül des natürlichen Schließens* von Gerhard Gentzen. Der folgende Beweisbaum[1] beweist einen Teil der De Morganschen Gesetze.

$$\dfrac{\dfrac{\dfrac{u_3 : \varphi \vee \psi \qquad \dfrac{\dfrac{u_1 : \varphi \qquad \dfrac{u_0 : \neg\varphi \wedge \neg\psi}{\neg\varphi}\,(\text{iv})}{\bot}\,(\text{iii})}{\bot} \qquad \dfrac{\dfrac{u_2 : \psi \qquad \dfrac{u_0 : \neg\varphi \wedge \neg\psi}{\neg\psi}\,(\text{iv})}{\bot}\,(\text{iii})}{\bot}}{\bot}\,(\text{v});\ u_1, u_2}{\neg\varphi \vee \psi}\,(\text{iii});\ u_3}{(\neg\varphi \wedge \neg\psi) \to \neg(\varphi \vee \psi)}\,(\text{ii});\ u_0$$

Hier steht die zu beweisende Behauptung in der Wurzel eines Baums. Die Wahrheit jedes inneren Knotens geht aus seinen Vorgängerknoten sowie einer festen Menge an Regeln

[1] Wir orientieren uns hier am Kalkül des natürlichen Schließens, wie es von Freund [Fre23] verwendet wird. Römische Ziffern geben an, welche Schlussregel angewendet wird. Mit u_i sind die verschiedenen Annahmen gekennzeichnet. Kann eine Annahme nach Schlussregel entfernt werden, wird das neben der Nummer der Schlussregel gekennzeichnet.

hervor. Im Kontext der Beweiskomplexität können wir zwei Beobachtungen machen. Zunächst ist es oft keine einfache Aufgabe, einen passenden Beweisbaum zu konstruieren. Es ist nicht eindeutig klar, welche Vorgänger man wählen muss, um in einem Blatt ein Axiom zu erhalten. Andererseits kann, gegeben einen Beweisbaum, die Korrektheit leicht nachvollzogen werden, wenn man die Regeln des natürlichen Schließens kennt. Es ist ausreichend, von den Blättern ausgehend immer die Korrektheit einer einzelnen Regelanwendung zu prüfen. Ein Verifier kann so die globale Entscheidung auf eine Reihe von lokalen zurückführen, die leicht zu treffen sind. Eine prüfende Maschine muss dafür nur den Beweis einlesen und seine Struktur erkennen können.

Dieser Effekt ist auch nicht auf das natürliche Schließen beschränkt. Vielmehr ist das eine Gemeinsamkeit vieler logischer Kalküle. Das zeigt, dass der bereits beobachtete Effekt nicht auf in der Praxis geführte Beweise beschränkt ist und motiviert daher die abstraktere Betrachtungsweise von Beweissystemen dieser Arbeit. Diese wollen wir nun im Folgenden genauer untersuchen.

Formale Beweissysteme Wir nähern uns dem Begriff von Beweissystemen zunächst über die beobachtete Verifizierbarkeit. Ein Beweissystem lässt sich dafür als Maschine auffassen, die als Eingabe zwei Wörter c und p (Behauptung und Beweis) aus einer formalen Sprache erhält. Falls p tatsächlich ein korrekter Beweis für c ist, so gibt die Maschine *wahr* aus, ansonsten *falsch*. Wir müssen natürlich fordern, dass sich das Beweissystem *korrekt* verhält, dass also ausschließlich für wahre Behauptungen c ein Beweis existiert, mit dem c akzeptiert wird. Für jede falsche Behauptung muss also das Beweissystem mit jedem Beweis die Falschheit erkennen. Das heißt nicht, dass die Maschine bereits für einen konkreten Beweis erkennt, dass die Behauptung allgemein falsch ist, sondern nur, dass der gegebene Beweis für die Behauptung nicht korrekt ist.

Außerdem soll ein Beweissystem *vollständig* sein, sodass auch für jede wahre Behauptung ein passender Beweis existiert, mit dem das Beweissystem die Wahrheit erkennen kann. Es ist zu beachten, dass eine wahre Behauptung nicht mit jedem Beweis akzeptiert werden muss, bereits die Existenz eines einzigen Beweises genügt uns.

Wir wollen schließlich noch fordern, dass die Maschine nur leichte Berechnungen durchführen kann. Dadurch ist sichergestellt, dass die Schwierigkeit der Verifikation unabhängig von der Suche nach dem Beweis selbst ist. Für diese Schranke kann man sich einen Computer vorstellen, der mit begrenzten Ressourcen die Funktion ausrechnen soll, die zwischen wahr und falsch unterscheidet. Um den Beweis zu prüfen kann er nur simple Schlüsse selbst ziehen, die wesentlichen Gedanken des Beweises müssen ihm explizit gegeben sein. Wir belassen es an dieser Stelle bei dem unpräzisen Begriff von „leichten“ Berechnungen, in Kapitel 2 definieren wir diesen Begriff aber exakt. Tatsächlich werden wir dort Beweissysteme so einführen, dass wir Beweissysteme auf verschiedene Berechnungskomplexitäten beschränken können.

Wir wollen diese Anschauung nun an einem Beispiel betrachten. Wir betrachten dafür Aussagen der Form „x ist das Produkt von genau zwei Primzahlen“ mit natürlichen Zahlen x. Mit Ω bezeichnen wir genau die Zahlen, für die diese Aussage wahr ist. Bekommen wir nun eine Zahl x gegeben, so fällt es uns vermutlich schwer herauszufinden, ob diese

in Ω enthalten ist. Tatsächlich basiert die Sicherheit einiger moderner kryptographischer Verfahren wie beispielsweise RSA darauf, dass die Primfaktoren einer Zahl nicht effizient berechenbar sind. Will uns jemand davon überzeugen, dass die Aussage für x wahr ist, so können wir zusätzlich zu x selbst noch die beiden Primfaktoren verlangen. Ist tatsächlich $x \in \Omega$, so existieren diese Primfaktoren. Wir müssen sie nur multiplizieren und erkennen so leicht, dass x tatsächlich das Produkt von zwei Primzahlen ist. Ist x dagegen kein Element von Ω, so kann es kein Primzahlpaar geben, mit dem wir getäuscht werden können. Das Multiplizieren von zwei Zahlen ist ausreichend einfach, sodass unser Beweissystem effizient von einer Maschine berechnet werden kann. Außerdem können Maschinen effizient prüfen, ob eine gegebene Zahl eine Primzahl ist.

Interessieren wir uns für die Länge der Eingabe, so ist es häufig sinnvoll, die zu beweisende Aussage vollständig vom Beweis zu trennen. Erhält die Maschine nur den Beweis als Eingabe, so entspricht die Beweislänge genau der Länge der Eingabe. Statt wahr und falsch soll die Maschine nun für ihre Eingaben Behauptungen ausgeben. Korrektheit stellen wir nun sicher, indem ausschließlich wahre Behauptungen ausgegeben werden, Vollständigkeit indem jede wahre Behauptung für mindestens eine Eingabe ausgegeben wird. Auch diese Maschine soll effizient rechnen. Es mag zunächst seltsam wirken, dass das Beweissystem nicht explizit erfährt, welche konkrete Behauptung bewiesen werden soll. Diese Information muss aus dem Beweis selbst extrahiert werden. Im eben genannten Beispiel würde das Beweissystem für Ω nun nur noch Zahlen erhalten. Falls genau zwei Primzahlen p und q eingegeben wurden, kann das Beweissystem ihr Produkt $p \cdot q$ ausgeben. Wir können (p, q) insofern als Beweis für $p \cdot q$ auffassen, da wir wissen, dass unser Beweissystem ausschließlich Werte aus Ω ausgibt.

1.2. Simulationen zwischen Beweissystemen

Um die Mächtigkeit von Beweissystemen miteinander vergleichen zu können, führen wir nun ein paar Begriffe ein. Wir vergleichen Beweissysteme zunächst rein auf Basis der Länge ihrer Beweise. Man spricht von *Simulation*, wenn Beweissystem A verglichen mit Beweissystem B nur unerheblich längere Beweise erfordert. Für jeden B-Beweis muss also ein A-Beweis für die identische Behauptung existieren, der nur unerheblich länger ist. Was genau mit „nur unerheblich länger“ gemeint ist, werden wir in Kapitel 2 mathematisch präzisieren. Wollen wir ein Beweissystem mit jedem anderen System der gleichen Sprache vergleichen, so können wir das über den Begriff von *optimalen* Beweissystemen tun. Wir nennen ein Beweissystem optimal, wenn es jedes Beweissystem für die gleiche Menge simuliert. In gewisser Hinsicht kann man optimale Beweissysteme also als bestmögliche Systeme verstehen, da kein anderes Beweissystem signifikant kürzere Beweise zulässt. Möchte man also zeigen, dass kein Beweissystem einer Sprache überall kurze Beweise haben kann, so kann man ein optimales Beweissystem betrachten. Man muss dann nur zeigen, dass dieses optimale Beweissystem nicht überall kurze Beweise hat. Dieses Vorgehen im Bezug auf die NP $\stackrel{?}{=}$ coNP Frage ist auch als Cooks Programm [Bus12] bzw. Cook-Reckhow Programm [Bey12] bekannt.

Es gibt noch einen weiteren Simulationsbegriff für Beweissysteme, den wir zur Abgrenzung zunächst wie in der Literatur üblich als *p-Simulation* bezeichnen werden (die schärfere l-Simulation werden wir erst später definieren). Für die p-Simulation ist es nicht ausreichend, nur die Länge von Beweisen zu betrachten. Hier wollen wir zusätzlich fordern, dass aus Beweisen des simulierten Systems sogar effizient entsprechende Beweise des simulierenden Systems berechnet werden können. Intuitiv kann man diese Art der Simulation daher so verstehen, dass das simulierende System bereits mit geringfügig umformulierten Beweisen des simulierten Systems arbeiten kann. Man muss also einen gegebenen Beweis höchstens leicht anpassen, damit er im simulierenden System verstanden wird, man muss aber keinen grundlegend neuen Beweis finden.

Wie auch bei normaler Simulation wollen wir Systeme, die jedes Beweissystem für die gleiche Sprache p-simulieren, als p-optimal bezeichnen. Solche Systeme kann man als universelle Beweissysteme auffassen. In gewisser Weise existiert kein besseres Kalkül, da das p-optimale System bereits jedes andere Beweissystem bzw. eine Umformulierung davon enthält. Besonders interessant wirkt diese Variante der Simulation im Kontext von konstruktiven und nicht-konstruktiven Beweisen. Ist nur nach der Existenz einer Lösung gefragt, so könnte es möglich sein, diese zu beweisen, ohne dabei eine konkrete Lösung anzugeben. Ist aber ein Beweissystem, in welchem die Lösung bereits Teil des Beweises ist, p-optimal, so können auch aus solchen nicht-konstruktiven Beweisen effizient konkrete Lösungen bestimmt werden.

1.3. Komplexität

Berechnungskomplexität Ein übliches Maß für die Schwierigkeit einer Berechnung ist die benötigte Laufzeit des Algorithmus. Diese wird abhängig von der Länge der Eingabe gemessen, für längere Eingaben bekommt ein Algorithmus also mehr Zeit für die Verarbeitung. Deterministische Algorithmen, deren Laufzeit höchstens polynomiell in der Länge der Eingabe wächst, erachtet man in der Regel als effizient. Es hat sich daher in der Komplexitätstheorie etabliert, Beweissysteme auf polynomielle Laufzeit einzuschränken. Dagegen wird Speicher als Ressource in diesem Kontext eher selten beachtet. Eine Ausnahme ist Bonacina [Bon18], der für konkrete Beweissysteme den Speicher untersucht. Sogar Bonacina macht die Effizienz von Beweissystemen aber nicht an einer Schranke für den Speicherbedarf fest. Dabei haben wir zu Beginn des Kapitels bereits erkennen können, dass es für Beweise im Sinne der mathematischen Praxis wünschenswert wäre, sich für das Prüfen nur Referenzen in den Beweis merken zu müssen. Für Algorithmen kann man diese stärkere Einschränkung dadurch ausdrücken, dass man sich auf logarithmischen Speicher beschränkt. Auch wenn man übliche Beweissysteme der mathematischen Logik wie das bereits genannte Kalkül des natürlichen Schließens oder auch das Sequenzenkalkül betrachtet, so scheint die Einschränkung auf logarithmischen Raum die natürlichere Wahl zu sein. Diese Kalküle lassen sich nämlich auch Überprüfungen ohne viel zusätzlichen Speicher zu, da die einzelnen angewandten Regeln sehr leicht prüfbar sind.

Wie auch Zeit, messen wir den Speicherbedarf abhängig von der Eingabelänge, wobei wir aber den Speicher für Ein- und Ausgabe nicht mitzählen. In dieser Arbeit verwenden wir vor allem eine Einschränkung der Speicherkomplexität unserer Beweissysteme, mit der wir einen schärferen Effizienzbegriff erreichen. Eine Einschränkung des Speichers bedeutet gleichzeitig auch eine beschränkte Laufzeit. Unsere Wahl von logarithmischem Raum beschränkt die Beweissysteme auch weiterhin auf polynomielle Laufzeit, lässt aber keinen polynomiellen Raum mehr zu. Somit ist die in dieser Arbeit untersuchte Menge an Beweissystemen eine Teilmenge der üblichen.

Ein zentrales Konzept für unseren Effizienzbegriff ist das *asymptotische Wachstum*. Ist ein Algorithmus für ein Problem gegeben, so ist es eine leichte Aufgabe für einzelne Probleminstanzen den Ressourcenverbrauch zu minimieren. Dafür kann man das Ergebnis des Algorithmus auf diesen Instanzen bereits vorher bestimmen und in einem modifizierten Algorithmus nun abfragen, ob einer der bekannten Fälle gefragt ist. Dieses Vorgehen funktioniert aber nur für endlich viele Modifikationen, da sonst allein die Liste der zu prüfenden Sonderfälle unendlich wäre. Wir betrachten daher vorwiegend den asymptotischen Verbrauch von Ressourcen für genügend große Eingaben.

Arbeitet ein Beweissystem mit begrenzten Ressourcen, so ist klar, dass für Aussagen, die nur mit hohem Ressourcenaufwand beweisbar sind, der Beweis nicht vom Prüfer selbst gefunden werden kann. Um die Maschine dennoch von der Korrektheit der Eingabe zu überzeugen wird es daher nötig, bereits beim Auffinden des Beweises die Arbeit, die der Prüfer nicht selbst ausführen kann, vorwegzunehmen und so zu dokumentieren, dass der Prüfer sie nachvollziehen kann. Daraus resultieren längere Beweise.

Beweiskomplexität Es ist naheliegend, auch die Länge von Beweisen als eine Art Ressource zu betrachten. Die Beweislänge wird dabei abhängig von der Länge des Bewiesenen betrachtet. Intuitiv darf also für kompliziertere Behauptungen auch der Beweis länger werden. Ist eine Menge gegeben, so kann man nun fragen, ob für beliebige Systeme eine untere Schranke angegeben werden kann. Wie auch für Zeit und Raum betrachten wir die Asymptotik der Beweislänge, da jedes Beweissystem um kurze Beweise für eine endliche Menge ergänzt werden kann.

Hat man eine feste Menge gegeben, so ist oft bereits eine untere Schranke für die Beweislänge interessant. Man untersucht also, ob jedes Beweissystem für diese Menge asymptotisch mindestens eine gewisse Beweislänge hat. Auf den ersten Blick scheint diese Fragestellung unabhängig von den Fragen der Komplexitätstheorie zu sein. Doch versteht man Beweissysteme als Verifier, so wird klar, dass es Zusammenhänge geben muss. Es wird beispielsweise der Zusammenhang mit nichtdeterministischen Berechnungen offenbar. Kennt man ein Beweissystem mit beschränkter Beweislänge, so kann man die bewiesene Menge entscheiden, indem man nichtdeterministisch den Beweis rät und schließlich prüft, ob ein korrekter Beweis für die Eingabe gefunden wurde. Da die Beweislänge beschränkt ist, kann man auf die Laufzeit dieses Algorithmus schließen. Dieser Zusammenhang ist der Kern des bereits genannten Cook-Reckhow Programms. Es ist bekannt, dass genau dann NP = coNP gilt, wenn ein Beweissystem für TAUT mit höchstens polynomiell langen Beweisen existiert [CR79]. Um dieses berühmte Problem

zu lösen, könnte man also ein Beweissystem für TAUT mit kurzen Beweisen suchen oder nachweisen, dass ein optimales Beweissystem für TAUT nicht überall kurze Beweise hat.

Bei Fenner et al. [FFNR03] sowie bei Pudlák [Pud17] zeigt sich außerdem, dass auch p-optimale Beweissysteme eine enge Beziehung mit Fragen aus der Komplexitätstheorie aufweisen. Es gelingt auch Messner [Mes01] eine Reihe von Kollapsvermutungen der Komplexitätstheorie mit Vermutungen der Beweiskomplexität in Verbindung zu setzen. Eine tiefere Diskussion der hier genannten Resultate würde den Rahmen sprengen, daher sei an dieser Stelle darauf verzichtet. Aus solchen Beziehungen können wir jedoch die Motivation ziehen, die Fragen dieser Arbeit zu untersuchen.

2. Vorbereitungen

Wir wollen nun zunächst Berechnungen formalisieren. In unseren Berechnungen erhalten wir immer eine Eingabe und wollen nach fest definierbaren Regeln eine Ausgabe tätigen. Ein- und Ausgabe müssen dargestellt werden. Dafür verwenden wir ein Alphabet Σ mit dessen Symbolen die Objekte codiert werden können. Als Standardalphabet werden wir in dieser Arbeit $\Sigma = \{0, 1\}$ verwenden. Sprachen über anderen Alphabeten lassen sich gegebenenfalls effizient in $\{0, 1\}$ umcodieren.

Wir führen nun zunächst Notationen für Worte und Sprachen ein. Ein Wort w über Σ ist eine Zeichenkette $w_1, \ldots, w_n$ mit $w_i \in \Sigma$. Die Länge von w ist die Länge der Zeichenkette und wird als $|w|$ geschrieben. Die Menge der Worte über Σ mit Länge n schreiben wir als Σ^n. Die Worte über Σ der Länge mindestens bzw. höchstens n bezeichnen wir entsprechend mit $\Sigma^{\geq n}$ bzw. $\Sigma^{\leq n}$. Die Menge aller Worte über Σ wird mit Σ^* bezeichnet. Für eine kompaktere Darstellung schreiben wir ein Wort auch als Konkatenation seiner Bestandteile. Das Wort der Sequenz $w_1, w_2, w_3, \ldots, w_n$ bezeichnen wir also auch mit $w_1 w_2 w_3 \ldots w_n$.

Wollen wir nun Objekte codieren, so können wir eine invertierbare Abbildung dieser Objekte in Σ^* verwenden. Besonders interessant ist das, wenn wir mehrere Objekte codieren wollen. Die resultierende Funktion nennen wir *Listencodierung*. Wir verwenden dafür in den folgenden Kapiteln eine Standardcodierung, in der man die Codierung der einzelnen Objekte verwendet, ihre Symbole jeweils dupliziert und durch ein alternierendes Paar Trennungen einfügt. Diese Codierung schreiben wir kurz durch $\langle a_1, \ldots, a_n \rangle$. Wir wählen diese Codierung, da die Codierung von Listen so ausreichend kurz ist und andererseits die Liste effizient decodiert werden kann.

2.1. Turingmaschinen

Um nun über die Komplexität von Rechnungen auf codierten Objekten argumentieren zu können, definieren wir ein Maschinenmodell, welches unseren Berechnungsbegriff darstellen soll. Wir orientieren uns dabei an Turingmaschinen (TM), wie sie von Homer und Selman [HS11] verwendet werden.

Intuitiv besteht eine solche Maschine aus vier Bestandteilen. Die Maschine hat eine Steuereinheit, die immer einen von endlich vielen Zuständen speichert. Weiterhin hat die Maschine eine endliche Anzahl von beidseitig unbegrenzten Bändern mit je einem Lese- und Schreibkopf. Diese Bänder sind in Zellen unterteilt, von denen jede genau ein Symbol aus Σ oder ein Leersymbol $\square \notin \Sigma$ enthält. Auf den *Arbeitsbändern* können sich die Köpfe in jedem Takt nach links oder rechts bewegen. Sie können außerdem das Symbol an ihrer aktuellen Position verändern und abhängig von den Regeln der Steuereinheit überschreiben. Den Inhalt eines Bands kann man durch das kürzeste $b \in \Sigma^* \cup \{\square\}$

darstellen, für das der Bandinhalt genau $\ldots \square\square b\square\square \ldots$ ist. Auf einem *Eingabeband* erhält die Maschine die Eingabe für die Rechnung. Dieses Band ist read-only. Außerdem hat die Maschine ein *Ausgabeband*, auf dem das Ergebnis der Rechnung ausgegeben wird. Hier kann die Maschine weder lesen noch explizit die Position verändern. Schreibt die Maschine ein Symbol auf das Ausgabeband, so verschiebt sich stattdessen der Kopf auf diesem Band automatisch um eine Position nach rechts. So ist sichergestellt, dass einmal ausgegebene Symbole nicht wieder überschrieben werden.

Formal können wir eine Turingmaschine mit k Arbeitsbändern wie folgt definieren.

Definition 2.1 (Turingmaschine)**.** *Eine* k-Band Turingmaschine *(k-TM) ist vollständig durch* $(\Sigma, Z, \delta, z_0, F)$ *beschrieben, wobei*

- Σ *ein Alphabet mit* $\square \notin \Sigma$*,*
- Z *eine endliche nichtleere Menge von Zuständen,*
- $\delta \subseteq (Z \times (\Sigma \cup \{\square\})^{k+1}) \times (Z \times (\Sigma \cup \{\square\})^{k+1} \times \{L, O, R\}^{k+1})$ *eine Überführungsrelation,*
- z_0 *der Startzustand und*
- $F \subseteq Z$ *die Menge an akzeptierenden Zuständen ist.*

Ist δ sogar eine partielle Funktion, so nennen wir die Maschine auch deterministisch*. Um zu betonen, dass kein Determinismus gefordert ist, können wir sonst auch von* nichtdeterministischen *Maschinen sprechen. Diese schließen die deterministischen mit ein.*

Definition 2.2 (Konfiguration einer TM)**.** *Die* Konfiguration *einer k-TM ist durch ihren Zustand sowie die vollständigen Inhalte ihrer lesbaren Bänder mit den jeweiligen Kopfpositionen gegeben. Das Ausgabeband ist kein Bestandteil der Konfiguration.*

Definition 2.3 (Rechenweg einer TM)**.** *Die Berechnung einer TM M auf einer Eingabe x bezeichnen wir mit $M(x)$. Ein Rechenweg einer solchen Berechnung ist eine Sequenz $K_1, K_2, \ldots$ von Konfigurationen, wobei sich M in K_1 genau im Startzustand befindet, x auf dem Eingabeband steht und alle anderen Bänder leer sind. Eine Konfiguration K_i wird dann in einem Takt in K_{i+1} überführt. Ist diese Sequenz endlich mit der letzten Konfiguration K_n, so ist entweder der Zustand in K_n akzeptierend (akzeptierender Rechenweg) oder es existiert in δ keine auf K_n anwendbare Regel (ablehnender Rechenweg).*

Wir können zwei Arten von Problemstellung unterscheiden. Haben wir eine Teilmenge $A \subseteq \Sigma^*$ gegeben und sollen für Worte $x \in \Sigma^*$ entscheiden, ob $x \in A$, so sprechen wir von einem *Entscheidungsproblem*. Hier interessieren wir uns nicht für die Ausgabe der Maschine, durch akzeptierende und ablehnende Rechenwege erhalten wir bereits die gewünschte Information. Eine Maschine für ein solches Problem heißt *Akzeptor*. Soll die Maschine eine Relation (im deterministischen Fall eine partielle Funktion) berechnen, so ist der Inhalt des Ausgabebands eine Ausgabe der Maschine, falls sie sich in einem akzeptierenden Zustand befindet. Eine solche Maschine nennen wir *Transduktor*.

In der Komplexitätstheorie interessieren wir uns für die Ressourcen, die ein Algorithmus benötigt. Üblicherweise betrachtet man dafür Laufzeit und Speicherbedarf. Diese Größen können wir nun für unser Berechnungsmodell definieren.

Definition 2.4 ([Pap94])**.** *Die Laufzeit von $M(x)$, geschrieben als $\text{time}_M(x)$, ist die Anzahl an Übergängen im längsten Rechenweg von $M(x)$. Der von einer Konfiguration verbrauchte Speicher ist das Maximum der Länge der Arbeitsbandsbandinhalte in der Konfiguration. Der verbrauchte Speicher von $M(x)$, geschrieben als $\text{space}_M(x)$ ist das Maximum des Speicherverbrauchs der Konfigurationen, die auf einem Rechenweg von $M(x)$ eingenommen werden.*

Definition 2.5. *Die Komplexität von Sprachen ergibt sich aus einer Schranke für den Ressourcenverbrauch auf allen Eingaben. Für eine Sprache $A \subseteq \Sigma^*$ und eine Schranke $f : \mathbb{N} \to \mathbb{N}$ gilt:*

- $A \in \text{NTIME}(f) \iff$ *es existiert eine TM M mit* $\text{time}_M(x) \leq f(|x|)$ *für alle $x \in \Sigma^*$, die genau A akzeptiert.*
- $A \in \text{DTIME}(f) \iff$ *es existiert eine deterministische TM M mit* $\text{time}_M(x) \leq f(|x|)$ *für alle $x \in \Sigma^*$, die genau A akzeptiert.*
- $A \in \text{NSPACE}(f) \iff$ *es existiert eine TM M mit* $\text{space}_M(x) \leq f(|x|)$ *für alle $x \in \Sigma^*$, die genau A akzeptiert.*
- $A \in \text{DSPACE}(f) \iff$ *es existiert eine deterministische TM M mit* $\text{space}_M(x) \leq f(|x|)$ *für alle $x \in \Sigma^*$, die genau A akzeptiert.*

Definition 2.6. *Mit* FP *bezeichnen wir die Funktionen, die von einer deterministischen TM in Polynomialzeit berechnet werden.*

Definition 2.7. *Mit* FL *bezeichnen wir die Funktionen, die von einer deterministischen TM in logarithmischem Raum berechnet werden.*

Eine übliche Technik um den Ressourcenverbrauch von Maschinen zu kontrollieren ist, explizit die gewünschten Schranken zu berechnen. Dazu ist es natürlich nötig, die Schranke bereits mit den begrenzten Ressourcen berechnen zu können. Solche Funktionen nennen wir zeit- bzw. raumkonstruierbar.

Definition 2.8. *Eine totale Funktion $f : \mathbb{N} \to \mathbb{N}$ heißt zeitkonstruierbar $\iff$ es existiert eine deterministische TM, die auf Eingaben 0^n in genau $f(n)$ Takten akzeptierend hält.*

Definition 2.9. *Eine totale Funktion $s : \mathbb{N} \to \mathbb{N}$ heißt raumkonstruierbar $\iff$ es existiert eine deterministische TM, die auf Eingaben 0^n genau $s(n)$ Speicher beschreibt und dann akzeptierend hält.*

Behauptung 2.10. *Jede zeitkonstruierbare Funktion ist raumkonstruierbar.*

Beweis. Ist f raumkonstruierbar, so betrachtet man M, die auf 0^n in $f(n)$ Takten akzeptiert. M kann höchstens $f(n)$ Speicher beschreiben. Gleichzeitig kann eine Maschine M' genauso arbeiten und auf einem zusätzlichen Arbeitsband in jedem Takt ein Symbol schreiben sowie eine Rechtsverschiebung ausführen. Somit schreibt M' auch mindestens $f(n)$ Symbole. □

Wir können in der Definition von Turingmaschinen sehen, dass eine TM durch endliche Objekte eindeutig definiert ist. Es ist daher möglich, die Maschine selbst als Wort in Σ^* zu codieren. Eine genaues Vorgehen liefern Homer und Selman [HS11]. Wählen wir als Alphabet $\{0,1\}$, so erhalten wir über die Binärdarstellung von Zahlen für jede TM eine natürliche Zahl, die wir als *Code* der Maschine bezeichnen. Eine TM mit Code i schreiben wir auch als M_i. Umgekehrt kann auch jede natürliche Zahl zu einer Maschine decodiert werden. Zahlen, die zunächst kein gültiger Code einer TM sind, decodieren wir dabei zu einer festen Maschine. Wir erhalten eine Aufzählung $M_1, M_2, \ldots$ aller Turingmaschinen. Eine solche Aufzählung können wir auch erhalten, wenn wir uns auf deterministische Maschinen beschränken, da man im Code einer Maschine bereits prüfen kann, ob diese deterministisch ist.

Es ist bekannt, dass *universelle Turingmaschinen* existieren. Eine solche Maschine erhält Eingaben der Form $\langle i, x\rangle$ und simuliert dann M_i auf Eingabe x. Wir verwenden in dieser Arbeit eine Turingmaschine, die diese Simulation auch effizient durchführen kann.

Definition 2.11. *U ist eine von Neary und Woods [NW12] garantierte universelle Turingmaschine, die auf Eingaben $\langle i, x\rangle$ die Rechnung $M_i(x)$ simuliert. Es gilt dabei $\text{space}_U(\langle i, x\rangle) \leq c_i \cdot \text{space}_{M_i}(x) + c_i$ für eine Konstante c_i, die nur von i abhängt.*

Reduktionen Um in der Komplexitätstheorie die Schwierigkeiten von Problemen vergleichen und die von Komplexitätsklassen charakterisieren zu können, verwendet man *Reduktionen*. Dafür sind Funktionen nötig, die Instanzen eines Problems in die eines anderen übersetzt. Interessant sind solche Reduktionen vor allem dann, wenn die Berechnungskomplexität der Reduktionsfunktion nicht ausreicht, um die eingegebene Instanz selbst zu entscheiden. Wir werden in Kapitel 3 zeigen, dass durch Reduktionen auch Beweissysteme übertragen werden können.

Definition 2.12. *Für $A, B \subseteq \Sigma^*$ ist $A \leq_m^p B \iff$ es existiert eine totale Funktion $f \in \text{FP}$, sodass für alle Worte $x \in \Sigma^*$ gilt $x \in A \Leftrightarrow f(x) \in B$.*

Definition 2.13. *Für $A, B \subseteq \Sigma^*$ ist $A \leq_m^{\log} B \iff$ es existiert eine totale Funktion $f \in \text{FL}$, sodass für alle Worte $x \in \Sigma^*$ gilt $x \in A \Leftrightarrow f(x) \in B$.*

Für manche Beweise benötigen wir eine zusätzliche Bedingung in der Reduktion. Hier fordern wir zusätzlich, dass leichte Teilmengen des Ursprungsproblems auch auf leichte Instanzen abgebildet werden.

Definition 2.14. *Für zwei Sprachen $A, B \subseteq \Sigma^*$, Komplexitätsklassen $\mathcal{C}$ und $\mathcal{D}$ sowie $\circ \in \{p, \log\}$ gilt $A \leq_{m,\mathcal{C}-\mathcal{D}}^{\circ} \iff A \leq_m^{\circ} B$ via einem f, sodass für jedes $S \subseteq A$ mit $S \in \mathcal{C}$ auch $f(S) \in \mathcal{D}$ gilt.*

2.2. Optimale Algorithmen

Definition 2.15 ([Mes01])**.** *Sei $\mathcal{M}$ eine Menge an Turingmaschinen und $S \subseteq \Sigma^*$. Eine Maschine $M \in \mathcal{M}$ ist* zeitoptimal *für $\mathcal{M}$ auf S $\iff$ für jede Maschine $M' \in \mathcal{M}$ existiert ein Polynom p mit $\text{time}_M(x) \leq p(\text{time}_{M'}(x) + |x|)$ für alle $x \in S$.*

Definition 2.16. *Sei $\mathcal{M}$ eine Menge an Turingmaschinen und $S \subseteq \Sigma^*$. Eine Maschine $M \in \mathcal{M}$ ist* raumoptimal *für $\mathcal{M}$ auf S $\iff$ für jede Maschine $M' \in \mathcal{M}$ existiert eine Konstante c mit $\text{space}_M(x) \leq c \cdot (\text{space}_{M'}(x) + |x|)$ für alle $x \in S$.*

FL-Funktionen sind raumoptimal invertierbar

Bekanntes Resultat 1 (Messner Korollar 2.18)**.** Für jede partielle Funktion $h \in$ FP existiert ein zeitoptimaler Transduktor I_h zur Berechnung des Inversen von h. Dabei hält I_h auf Eingaben außerhalb von W_h nicht, es gilt also $I_h(y) \neq \perp \iff y \in W_h$.

Der Beweis bei Messner basiert auf einem Algorithmus von Levin [Lev73], mit dem jedes NP-Problem bis auf polynomielle Unschärfe in optimaler Zeit gelöst werden kann. Die Idee dieses Algorithmus ist, nach und nach die Arbeit aller Turingmaschinen zu simulieren, wobei jede betrachtete Maschine zu gleichen Teilen Arbeitszeit erhält. Ab einem gewissen Punkt wird dann auch eine Maschine simuliert, die die korrekte Lösung liefert. Diese Technik lässt sich auch verwenden, um raumbeschränkte Probleme zu lösen.

Lemma 2.17. *Für jede partielle Funktion $h \in$ FL existiert ein raumoptimaler Transduktor I_h zur Berechnung des Inversen von h. Dabei hält I_h auf Eingaben außerhalb von W_h nicht, es gilt also $I_h(y) \neq \perp \iff y \in W_h$.*

Beweis. Sei $M_1, M_2, \ldots$ eine Aufzählung aller Turing Maschinen. Sei T ein Algorithmus, der auf Eingaben $\langle i, y, n \rangle$ die Ausgabe von $M_i(y)$ bestimmt und prüft, ob M_i ein Inverses zu y berechnet. Der Parameter n dient dabei als Raumschranke.

Algorithmus 1 : T

```
Eingabe : i, y
wenn M_i(y) eine Ausgabe tätigt dann
    x := M_i(y)
    wenn h(x) = y dann
        gib x aus
```

Behauptung 2.18. *Es gilt $\text{space}_U(\langle i, y \rangle) \in \mathcal{O}(\text{space}_{M_i}(y))$.*

Beweis. In Zeile 2 muss nicht zwingend die ganze Ausgabe gespeichert werden. Da der Speicher aus Zeile 1 wiederverwendet werden kann, um die Berechnung erneut durchzuführen, ist in Zeile 3 nur ein Zeiger auf das betrachtete Symbol von x nötig. Da x höchstens exponentiell im Speicherverbrauch von M_i auf y liegt, gilt $\log|x| \in \mathcal{O}(\text{space}_{M_i}(y))$. Da außerdem $h \in \text{FL}$ liegt, kann $h(x)$ im gleichen Raum ausgewertet werden. Der Vergleich mit y lässt sich analog durch Programmierung mit Zeigern lösen. Nach Neary und Woods [NW12] ist schließlich auch die Simulation von M_i in Raum $\mathcal{O}(\text{space}_{M_i})$ möglich, insgesamt gilt daher $\text{space}_U(\langle i, y\rangle) \in \mathcal{O}(\text{space}_{M_i}(y))$. ■

Einen raumoptimalen Transduktor erhält man nun, indem man die Arbeit aller Turingmaschinen mit einer inkrementell wachsenden Raumschranke simuliert. Damit für einen Wert der Raumschranke nur endlich viele Simulationen durchgeführt werden müssen, erweitert man außerdem die Menge der simulierten Maschinen erst nach und nach. So ist trotzdem sichergestellt, dass jede Maschine zumindest für eine ausreichend große Raumschranke simuliert werden kann.

Algorithmus 2 : *UE*

```
   Eingabe : y
1  für n = 0 bis ∞ tue
2      für i = 1 bis ∞ tue
3          wenn T(i, y) in Raum n eine Ausgabe tätigt dann
4              gib T(i, y) aus
```

Behauptung 2.19. *UE arbeitet korrekt.*

Beweis. Nach einem Resultat von Sipser [Sip80] können wir ohne Beschränkung der Allgemeinheit annehmen, dass T ohne in Endlosschleifen zu geraten in der Raumschranke n arbeitet. Das kann man in T durch einen Schrittzähler umsetzen. Daher ist die Bedingung in Zeile 3 immer prüfbar.

UE imitiert die Ausgaben von U. Dort können nur solche x ausgegeben werden, für die in Zeile 3 $h(x) = y$ gilt. Für Eingaben $y \notin W_h$ kann UE also nicht terminieren. Ansonsten existiert ein x mit $h(x) = y$ und somit auch eine Maschine M_j, die konstant x ausgibt. Für ein ausreichend großes n wird daher M_j simuliert, wonach der Algorithmus terminieren muss. ■

Behauptung 2.20. *UE ist ein raumoptimaler Algorithmus für das Invertieren von* FL*-Funktionen.*

Beweis. Sei M_j nun eine beliebige TM, die h invertiert. Sobald in UE $n \geq j$ gilt, wird die Arbeit von M_j simuliert. Die vollständige Simulation von M_j gelingt somit in Raum $\mathcal{O}(\max\{j,\ c_j \cdot \text{space}_{M_j}(y)\})$. Da j und c_j jeweils Konstanten für M_j sind, ist $\mathcal{O}(\text{space}_{M_j}(y))$ Raum ausreichend, damit UE die Ausgabe von M_j imitiert. Falls UE

bereits vor der vollständigen Simulation von M_j terminiert, so müssen die Ausgaben nicht übereinstimmen. Trotzdem sind beide korrekte Inverse. ■

Durch eine Turingmaschine, die UE berechnet, erhält man den Transduktor I_h. □

2.3. Beweissysteme

Ein Beweissystem für eine Menge A können wir als surjektive Funktion $f : \Sigma^* \to A$ auffassen. Für $f(x) = y$ heißt x dann *f-Beweis für y*. Da wir wissen, dass f nur Werte aus A ausgeben kann, folgt aus der Existenz eines f-Beweises für y bereits $y \in A$. Erhält ein Verifier einen passenden Beweis für y, so kann er $f(x)$ auswerten und sich so von $y \in A$ überzeugen. Umgekehrt wissen wir durch die Surjektivität von f, dass für jedes Element von A auch ein passender f-Beweis existiert.

Es mag ungewohnt erscheinen, nur Beweise für die Zugehörigkeit zu einer Menge zu betrachten. Allerdings kann man durch Mengen wie TAUT einsehen, dass sich dadurch auch allgemeinere Aussagen codieren lassen. TAUT entspricht nämlich genau den gültigen Sätzen der Aussagenlogik. Ein Beweissystem für TAUT kann also die Gültigkeit von solchen Sätzen beweisen.

Erweiterung von Cook und Reckhow Durch die Definition über surjektive Funktionen sind bereits Korrektheit und Vollständigkeit abgedeckt. Wir haben jedoch bereits beobachtet, dass Beweissysteme auch leicht prüfbar sein sollen. Mithilfe der formalen Definitionen zur Berechnung von Funktionen kann nun auch dieser Forderung Rechnung getragen werden.

Definition 2.21 (Cook und Reckhow [CR79])**.** *Für $A \subseteq \Sigma^*$ ist eine Funktion $f : \Sigma^* \to A$ genau dann ein* FP*-Beweissystem für A, wenn $f \in$ FP und f surjektiv ist.*

In der Literatur sind solche FP-Beweissysteme üblich, sodass dort nur von Beweissystemen gesprochen wird. In dieser Arbeit sollen Beweissysteme aber abhängig von ihrer Berechnungskomplexität untersucht werden. Dafür erweitern wir nun den Begriff von Beweissystemen. Die neue Definition enthält die gezeigte von Cook und Reckhow als Spezialfall.

Definition 2.22. *Für $A \subseteq \Sigma^*$ und eine Menge $\mathcal{F}$ an Funktionen ist $f : \Sigma^* \to A$ genau dann ein $\mathcal{F}$-Beweissystem für A, wenn $f \in \mathcal{F}$ und f surjektiv ist.*

Von besonderem Interesse wird in dieser Arbeit $\mathcal{F} =$ FL sein. Grundsätzlich ist dabei bereits unklar, ob FL = FP gilt. Gleichheit würde jedoch auch L = P implizieren, was üblichen Annahmen widerspricht. Gerade weil diese beiden Klassen aber schwer zu trennen sind, ist eine separate Betrachtung von Interesse, um eventuelle Unterschiede besser verstehen zu können.

Optimale Beweissysteme Um Beweissysteme miteinander vergleichen zu können, wollen wir nun die Simulationsbegriffe definieren. Diese hängen stark mit den bereits vorgestellten Reduktionen zusammen.

Definition 2.23. *Ein Beweissystem f* simuliert *ein Beweissystem g $\iff$ es existiert eine höchstens polynomiell verlängernde Funktion t mit $g = f \circ t$. Wir schreiben $g \leq_s f$.*

Diese Simulation stellt sicher, dass zu jedem g-Beweis ein höchstens polynomiell längerer f-Beweis gefunden werden kann. Üblicherweise werden FP-Beweissysteme betrachtet. Hier kann man die polynomielle Unschärfe der Beweislänge dadurch motivieren, dass die Laufzeit für das Verifizieren von Beweisen dadurch nur unerheblich beeinflusst wird (FP selbst erlaubt eine polynomielle Unschärfe). Aber auch für FL-Beweissysteme ist diese Definition passend, da hier eine polynomiell längere Eingabe nur einen konstanten Faktor im Speicherbedarf bedeutet.

Allerdings ist nicht garantiert, dass ein f-Beweis effizient aus einem g-Beweis berechnet werden kann. Da wir für diesen nur die Länge beschränkt haben, kann eine Maschine nur nichtdeterministisch raten, was die korrekte Übersetzung wäre. Soll die Übersetzung aber auch deterministisch möglich sein, so brauchen wir einen schärferen Simulationsbegriff. Tatsächlich definieren wir hierfür verschiedene Varianten, um den Unterschieden der betrachteten Berechnungskomplexitäten gerecht zu werden.

Definition 2.24. *Ein Beweissystem f* p-simuliert *ein Beweissystem g $\iff$ es existiert ein $t \in \mathrm{FP}$ mit $g = f \circ t$. Wir schreiben auch kurz $g \leq_s^p f$. Ist sogar $t \in \mathrm{FL}$, so* l-simuliert *f g. Wir schreiben dann auch $g \leq_s^{\log} f$.*

Bekanntes Resultat 2 (Messner, Kapitel 2.5)**.** Die Simulationshierarchie zwischen FP-Beweissystemen ist ein algebraischer Verband. Folglich sind $\leq_s$ und $\leq_s^p$ transitive Relation auf FP.

Satz 2.25. *Die Relationen $\leq_s, \leq_s^p$ und $\leq_s^{\log}$ sind transitiv.*

Beweis. Für $f = g \circ t_1$ und $g = h \circ t_2$ erhalten wir $f = g \circ t_1 = (h \circ t_2) \circ t_1 = h \circ (t_2 \circ t_1)$. Daher ist nur zu zeigen, dass die für die Simulationen verwendeten Funktionsklassen unter Verkettung abgeschlossen sind. Für FP und FL gilt das bereits nach Balcázar et al. [BDJ11].

Für $\leq_s$ seien p, q Polynome mit positiven Koeffizienten, sodass für beliebige Eingaben x gilt $|t_1(x)| \leq p(|x|)$ und $|t_2(x)| \leq q(|x|)$. Sei $x' \in \Sigma^{\leq p(|x|)}$ nun so gewählt, dass die Länge von $t_2(x')$ maximal wird. Es folgt $|(t_2 \circ t_1)(x)| \leq |t_2(x')| \leq q(|x'|)$. Da q monoton steigend ist, gilt wegen $|x'| \leq p(|x|)$ auch $q(|x'|) \leq q(p(|x|))$. Folglich ist $t_2 \circ t_1$ eine höchstens polynomiell verlängernde Funktion. Somit gilt $f \leq_s h$. □

Für eine Sprache $A \subseteq \Sigma^*$ und eine Menge von Funktionen können wir nun die Optimalitätsbegriffe definieren. Wir haben die Simulationsbegriffe unabhängig von $\mathcal{F}$ definiert, damit wir auch zu Funktionen mit höherer Komplexität eine Simulationsrelation zeigen können. Für die Optimalitätsbegriffe ist aber natürlich eine Einschränkung auf eine konkrete Funktionenklasse nötig.

Definition 2.26. *Ein $\mathcal{F}$-Beweissystem f ist* optimal $\iff$ *$g \leq_s f$ für alle $g \in \mathcal{F}$ mit $W_f = W_g$.*

Definition 2.27. *Ein $\mathcal{F}$-Beweissystem f ist* p-optimal $\iff$ *$g \leq_s^p f$ für alle $g \in \mathcal{F}$ mit $W_f = W_g$.*

Definition 2.28. *Ein $\mathcal{F}$-Beweissystem f ist* l-optimal $\iff$ *$g \leq_s^{\log} f$ für alle $g \in \mathcal{F}$ mit $W_f = W_g$.*

3. Übertragbare Beweistechniken

3.1. Existenz von Beweissystemen

Bekanntes Resultat 3 (Messner Lemma 2.1)**.** Eine Sprache $A \subseteq \Sigma^*$ hat genau dann ein FP-Beweissystem, wenn $A \in \mathrm{RE}$.

Satz 3.1. *Eine Sprache $A \subseteq \Sigma^*$ hat genau dann ein* FL*-Beweissystem, wenn $A \in \mathrm{RE}$.*

Beweis. Hat A ein FL-Beweissystem, so ist A der Wertebereich einer surjektiven berechenbaren Funktion $f : \Sigma^* \to A$. Durch Codierung von Worten über Σ erhält man eine entsprechende Funktion $f' : \mathbb{N} \to A$. Damit ist A rekursiv aufzählbar.

Ist umgekehrt $A \in \mathrm{RE}$, so ist A der Wertebereich einer surjektiven berechenbaren Funktion $f : \mathbb{N} \to A$. Es existiert also eine Turingmaschine M, die f berechnet. Man kann nun $f' \in \mathrm{FL}$ definieren, indem M für eine Raumschranke simuliert wird:

$$f'(\langle x, 0^n \rangle) = \begin{cases} y & \text{falls } M(x) = y \text{ in Raum } \leq \log n \\ \bot & \text{sonst} \end{cases}$$

Es gilt $W_{f'} \subseteq W_f = A$. Außerdem beobachtet man für $a \in A$, dass ein x mit $f(x) = a$ gilt. Dann gilt auch $M(x) = a$. Sei s der von der Rechnung $M(x)$ benötigte Speicher. Dann gilt $f'(\langle x, 0^{2^s} \rangle) = a$. Folglich ist $W_{f'} = A$ und f' ein FL-Beweissystem für A. □

3.2. Triviale Beweissysteme

Auch wenn die Suche nach (l)-optimalen Beweissystemen im Allgemeinen nicht leicht lösbar ist, können für manche Sprachen sehr simple Beweissysteme (l)-optimal sein. Das ist insbesondere der Fall, wenn das Entscheidungsproblem der bewiesenen Sprache sehr einfach ist. Hier muss der Beweis nur die Information enthalten, welches Element bewiesen werden soll. Von der Korrektheit kann sich ein Verifier dann im Grunde selbst überzeugen, indem er entscheidet, ob das eingegebene Element in der Sprache liegt.

Bekanntes Resultat 4 (Messner, Theorem 3.1)**.** Jedes $A \in \mathrm{NP}$ hat ein optimales FP-Beweissystem. Jedes $A \in \mathrm{P}$ hat sogar ein p-optimales FP-Beweissystem.

Satz 3.2. *Jedes $A \in \mathrm{NL}$ hat ein optimales* FL*-Beweissystem. Jedes $A \in \mathrm{L}$ hat sogar ein l-optimales* FL*-Beweissystem.*

Beweis.

Behauptung 3.3. *Jedes* FL*-Beweissystem mit polynomiell langen Beweisen ist optimal.*

Beweis. Sei $f \in \mathrm{FL}$ mit $|x| \leq p(|f(x)|)$ für ein monoton steigendes Polynom p. Für jede Funktion $g \in \mathrm{FL}$ gilt außerdem $|g(x)| \leq q(|x|)$ für ein Polynom q, da g nur in logarithmischem Raum arbeiten kann. Für ein festes $y \in W_f \cap W_g$ erhalten wir daher einen f-Beweis x_f mit $|x_f| \leq p(|f(x_f)|) \leq p(|y|)$. Für jeden g-Beweis x_g für y gilt außerdem $|y| = |g(x_g)| \leq q(|x_g|)$. Durch Kombinationen der beiden Ungleichungen erhalten wir wegen der Monotonie von p

$$|x_f| \leq p(|y|) \leq p(q(|x_g|))$$

und es folgt somit die Behauptung, da Polynome unter $\circ$ abgeschlossen sind. ■

Sei $A \in \mathrm{L}$. Dann ist

$$f(x) = \begin{cases} x & \text{falls } x \in A \\ \bot & \text{sonst} \end{cases}$$

in FL berechenbar. Für jedes FL-Beweissystem g für A gilt $g(x) = f(g(x))$ für beliebige $x \in A$, g übersetzt also selbst g-Beweise in f-Beweise. Folglich gilt $g \leq_s^{\log} f$ und f ist ein l-optimales FL-Beweissystem für A.

Sei nun $A \in \mathrm{NL}$ und M eine NLTM mit $L(M) = A$. Eine Rechnung $M(x)$ kann nicht direkt von einer deterministischen Maschine in L simuliert werden, da die Nachfolgekonfiguration nicht eindeutig festgelegt sein muss. Es kann dabei sein, dass nur eine davon zu einer akzeptierenden Konfiguration führt. Um die Rechnung selbst simulieren zu können, müsste ein deterministischer Verifier also wissen, welche Übergänge erfolgen. Mit M' sei die LTM bezeichnet, die für Eingaben $\langle x, w \rangle$ den Berechnungspfad von $M(x)$ simuliert, der durch die in w codierten Übergänge festgelegt wird. Eine solche Codierung erhält man beispielsweise, indem man die Übergänge nummeriert. Für jeden Übergang sind dann nur konstant viele Bits nötig. Nun lässt sich analog zu f ein Beweissystem h wie folgt definieren:

$$h(\langle x, w \rangle) = \begin{cases} x & \text{falls } M'(\langle x, w \rangle) \text{ akzeptiert} \\ \bot & \text{sonst} \end{cases}$$

Da nur die Arbeit von M' auf der Eingabe simuliert und ein Teil der Eingabe auf das Ausgabeband kopiert werden muss, ist $h \in \mathrm{FL}$. Da außerdem jedes $x \in A$ von mindestens einem Rechenweg von $M(x)$ akzeptiert werden muss, existiert für x auch ein h-Beweis. Umgekehrt erkennt man, dass ein h-Beweis für ein y bereits einen akzeptierenden Rechenweg von $M(y)$ definiert. Wegen $L(M) = A$ gilt folglich $y \in A$. Also ist h ein FL-Beweissystem für A.

Weiterhin sind h-Beweise höchstens polynomiell lang. Dafür genügt es, die Länge von w zu betrachten. Da zu jedem Zeitpunkt nur konstant viele Übergänge möglich sind, kann die Übergangssequenz mit höchstens linearem Overhead in der Länge des Rechenwegs codiert werden. Es ist also $|w| \leq p(|x|)$ für ein Polynom p.

□

Aus Satz 4.4 wird folgen, dass sogar jede Sprache in NP ein optimales FL-Beweissystem hat. Dafür ist allerdings noch eine zusätzliche Einsicht nötig, da das im Beweis genannte h die Rechenwege eines Akzeptors simuliert. Soll eine NLTM eine NPTM simulieren, so ist unklar, wie das in logarithmischem Raum geschehen kann, da die NPTM bis zu polynomiellem Raum verwenden kann.

3.3. Abgeschlossenheit

Wir werden im Folgenden Methoden erhalten, mit denen die Existenz bzw. die Nichtexistenz von FL-Beweissystemen für Sprachen auf andere übertragen werden können. Das kann als Abgeschlossenheit der Menge der Sprachen mit (l-)optimalem Beweissystemen unter den Operationen $\cap$, $\times$ und $\leq_m^l$ aufgefasst werden. Wir betrachten dazu zwei Mengen $A, B \subseteq \Sigma^*$.

Es seien $A, B \subseteq \Sigma^*$ zwei Sprachen.

Bekanntes Resultat 5 (Messner Theorem 3.2)**.** Hat A ein (p-)optimales Beweissystem und ist $B \leq_m^p A$, so hat auch B ein (p-)optimales Beweissystem.

Satz 3.4. *Hat A ein (l)-optimales* FL*-Beweissystem und ist $B \leq_m^{\log} A$, so hat auch B ein (l)-optimales* FL*-Beweissystem.*

Beweis. Sei h ein l-optimales FL-Beweissystem für A und $B \leq_m^{\log} A$ via f. Sei weiterhin

$$h'(\langle x, w\rangle) = \begin{cases} x & \text{falls } h(w) = f(x) \\ \bot & \text{sonst} \end{cases}$$

Sei nun g' ein FL-Beweissystem für B. Sei g ein FL-Beweissystem für A mit $g(1w) = f(g'(w))$ und $g(0w) = h(w)$. Wegen der Optimalität von h existiert ein $t \in \mathrm{FL}$, sodass $h(t(1w)) = g(1w) = f(g'(w))$. Nach Definition von h' gilt dann $h'(\langle g'(w), t(1w)\rangle) = g'(w)$. Da g', t sowie die Listencodierung zweier Werte in FL berechnet werden können, wird g' folglich von h' l-simuliert.

Falls h nur ein optimales FL-Beweissystem für A ist, so ist t nicht zwingend in FL, es muss aber dennoch für beliebige Eingaben $|t(x)| \leq p(|x|)$ für ein Polynom p gelten. Dann ist auch $\langle g'(w), t(1w)\rangle$ höchstens polynomiell länger als w. Folglich wird g' von h' simuliert. □

Über die Kontraposition dieser Aussage erhält man auch eine Methode, um Nichtexistenz von (l-)optimalen FL-Beweissystemen zu übertragen.

Korollar 3.5. *Ist $B \leq_m^{\log} A$ und hat B kein (l-)optimales* FL*-Beweissystem, so hat auch A kein (l-)optimales* FL*-Beweissystem.*

Bekanntes Resultat 6 (Messner Theorem 3.3)**.** Haben A und B (p-)optimale Beweissysteme, so auch $A \cap B$, $A \times B$.

Satz 3.6. *Haben A und B (l)-optimale Beweissysteme, so auch $A \cap B$, $A \times B$.*

Beweis. Wir betrachten zunächst $A \cap B$. Seien h_A und h_B l-optimale FL-Beweissysteme für A und B. Dann kann h definiert werden als

$$h(w) = \begin{cases} x & \text{falls } w = \langle u, v \rangle \text{ und } x = h_A(u) = h_B(v) \\ \bot & \text{sonst} \end{cases}$$

Sei nun f ein FL-Beweissystem für $A \cap B$. Durch Kombination erhält man für $i \in \{A, B\}$ je ein FL-Beweissystem für A und für B mit $f_i(1w) = f(w)$ und $f_i(0w) = h_i(w)$. f-Beweise sind dabei leicht in f_i-Beweise übertragbar. Nach Annahme existieren Funktionen $t_i \in \mathrm{FL}$ mit $f_i(x) = h_i(t_i(x))$. Sei nun t definiert als $t(w) = \langle t_A(1w), t_B(1w) \rangle$. Um t zu berechnen, müssen nur zwei FL-Funktionen ausgewertet und ihre Ausgaben in ein Tupel codiert werden. Also ist t selbst in FL. Wegen

$$f(w) = h_A(t_A(1w)) = h_B(t_B(1w))$$

gilt $f(w) = h(t(w))$. Somit ist h l-optimal. Falls die Beweissysteme für A und B nur optimal sind, erhält man durch die gleiche Konstruktion ein optimales Beweissystem für $A \cap B$, da $\langle t_A(1w), t_B(1w) \rangle$ dann auch nur polynomiell länger als w ist.

Weiterhin ist zu beobachten, dass $A \times \Sigma^* \leq_m^{\log} A$ und $\Sigma^* \times B \leq_m^{\log} B$. Wegen $A \times B = (A \times \Sigma^*) \cap (\Sigma^* \times B)$ hat dann mit den bereits gezeigten Abschlusseigenschaften auch $A \times B$ ein (l)-optimales FL-Beweissystem. □

3.4. Zusammenhang mit Akzeptoren

Wir haben bereits gesehen, dass man Beweissysteme auch als Verifier auffassen kann. Daher ist es naheliegend, für unsere Optimalitätsbegriffe einen Zusammenhang zu optimalen Akzeptoren zu suchen. Auf diese sind übliche Techniken der Komplexitätstheorie anwendbar.

Bekanntes Resultat 7 (Messner Theorem 3.4)**.** Falls $A \times \Sigma^* \leq_{m,\mathrm{P}-\mathrm{P}}^p A$, so sind folgende Aussagen äquivalent:

(i) Es existiert ein p-optimales FP-Beweissystem für A.

(ii) Es existiert ein FP-Beweissystem für A, welches auf allen P-Teilmengen von A in FP invertiert werden kann.

(iii) Es existiert ein zeitoptimaler deterministischer Akzeptor für A.

(iv) Es existiert ein Akzeptor für A, welcher polynomielle Laufzeit auf jeder P-Teilmenge hat.

Zusätzlich gelten die Implikationen $(i) \Longrightarrow (iii)$, $(iii) \Longrightarrow (iv)$ und $(ii) \Longleftrightarrow (iv)$ für beliebige Sprachen.

Satz 3.7. *Falls* $A \times \Sigma^* \leq^{\log}_{m,\mathrm{L}-\mathrm{L}} A$*, so sind folgende Aussagen äquivalent:*

(i) Es existiert ein l-optimales FL*-Beweissystem für A.*

(ii) Es existiert ein FL*-Beweissystem für A, welches auf allen* L*-Teilmengen von A in* FL *invertiert werden kann.*

(iii) Es existiert ein raumoptimaler deterministischer Akzeptor für A.

(iv) Es existiert ein Akzeptor für A, welcher auf jeder L*-Teilmenge eine logarithmische Raumbeschränkung hat.*

Zusätzlich gelten die Implikationen $(i) \Longrightarrow (iii)$, $(iii) \Longrightarrow (iv)$ *und* $(ii) \Longleftrightarrow (iv)$ *für beliebige Sprachen.*

Beweis. Zu einem gegebenen FL-Beweissystem h sei I_h der von Lemma 2.17 garantierte raumoptimale Transduktor zur Invertierung von h. Wegen $y \in W_h \iff I_h(y) \neq \perp$ kann daraus auch ein Akzeptor A_h für W_h abgeleitet werden, der I_h simuliert und akzeptiert, sobald I_h einen Wert ausgeben würde. Für einen Akzeptor M sei zudem ein Beweissystem h_M für $L(M)$ definiert als

$$h_M(\langle y, 0^s \rangle) = \begin{cases} y & \text{falls } y \in L(M) \text{ mit } \mathrm{space}_M(y) \leq \log s \\ \perp & \text{sonst} \end{cases}$$

Behauptung 3.8. $(i) \Longrightarrow (iii)$

Beweis. Angenommen h ist ein l-optimales FL-Beweissystem für A. Es kann nun gezeigt werden, dass A_h ein optimaler deterministischer Akzeptor von A ist. Sei dafür M ein beliebiger Akzeptor mit $L(M) = A$ und h_M das entsprechende Beweissystem. Da h l-optimal ist, existiert ein $g \in \mathrm{FL}$ mit $h_M(x) = h(g(x))$ für $x \in A$. Mit einer Funktion $f : y \to \langle y, 0^{2^{\mathrm{space}_M(y)}} \rangle$ gilt somit für beliebige $y \in A$:

$$y = h_M(f(y)) = h(g(f(y)))$$

Die Verkettung $g \circ f$ invertiert also h. Weiterhin ist f in $\mathcal{O}(\mathrm{space}_M(y))$ Raum berechenbar und $g \in \mathrm{FL}$. Also wird h von $g \circ f$ sogar in $\mathcal{O}(\mathrm{space}_M(y))$ Raum invertiert. Da I_h ein optimaler Algorithmus ist, um h zu invertieren, benötigt I_h also nur $\mathcal{O}(\log |y| + \mathrm{space}_M(y))$ Raum. Da A_h den gleichen Raumbedarf wie I_h hat, ist A_h ein optimaler deterministischer Akzeptor von A. ∎

Behauptung 3.9. $(iii) \Longrightarrow (iv)$

Beweis. Sei O ein raumoptimaler deterministischer Akzeptor für A und $S \in \mathrm{L}$ eine Teilmenge von A. Man kann eine L-Maschine O' für S mit O zu einer neuen Maschine O^* kombinieren, sodass beide Maschinen gleichzeitig simuliert werden. Dafür müssen nur die Kopfpositionen sowie die bisherigen Bandinhalte der beiden Maschinen gespeichert werden. Außerdem wird in einem Takt immer nur die Maschine simuliert, die dafür weniger Speicher benötigt. Dadurch gilt:

$$\mathrm{space}_{O^*}(x) = \min\{\mathrm{space}_O(x), \mathrm{space}_{O'}(x)\} + \log|x|$$

Auf S hat O^* folglich höchstens logarithmischen Raumbedarf. Damit O wie angenommen raumoptimal ist, muss $\mathrm{space}_O \in \mathcal{O}(\log|x| + \log|x|) = \mathcal{O}(\log|x|)$ gelten. ■

Behauptung 3.10. $(ii) \Longrightarrow (iv)$

Beweis. Sei h ein FL-Beweissystem, welches auf jeder L-Teilmenge FL-invertierbar ist, und $S \in \mathrm{L}$ eine beliebige Teilmenge von W_h. Es existiert ein Transduktor, der h invertiert und auf Eingaben aus S nur logarithmischen Raum benötigt. Da I_h raumoptimal ist, muss I_h auf S das Inverse auch in logarithmischem Raum finden. Folglich hat A_h auf S auch nur logarithmischen Raumbedarf. Da S beliebig gewählt war, akzeptiert A_h jede L-Teilmenge von W_h in logarithmischem Raum. ■

Behauptung 3.11. $(iv) \Longrightarrow (ii)$

Beweis. Sei M ein Akzeptor für A mit logarithmischer Raumschranke auf allen L-Teilmengen von A und $S \in \mathrm{L}$ eine beliebige Teilmenge von A. Das FL-Beweissystem h_M für A kann auf S invertiert werden, indem der Speicherbedarf von M notiert wird. Für ein $s \in S$ wird $M(s)$ simuliert, wobei zwei zusätzliche Zähler den aktuell verwendeten Speicher sowie das bisherige Maximum m speichern. Am Ende der Simulation kann $\langle s, 0^{2^m}\rangle$ ausgegeben werden. Da dann $h_M(\langle s, 0^{2^m}\rangle) = s$ gilt, ist ein Inverses von h_M so berechenbar.

Um m zu berechnen, ist $\mathcal{O}(\mathrm{space}_M(s))$ Speicher nötig. Um 0^{2^m} zu berechnen, ist ein m-Bit Zähler ausreichend. Zusammengenommen erhält man also einen Algorithmus, der h_M-Beweise in $\mathcal{O}(\mathrm{space}_M)$ Raum berechnet. Da M auf S logarithmisch raumbeschränkt ist, ist h_M also in FL invertierbar. ■

Behauptung 3.12. $(ii) \Longrightarrow (i)$

Beweis. Sei $A \times \Sigma^* \leq^{\log}_{m,\mathrm{L}-\mathrm{L}} A$ mit einer Funktion pad $\in$ FL und sei h ein FL-Beweissystem für A. Damit lässt sich nun eine Funktion f wie folgt konstruieren:

$$f(\langle y, x, z\rangle) = \begin{cases} y & \text{falls } h(z) = \mathrm{pad}(y, x) \\ \bot & \text{sonst} \end{cases}$$

Den Vergleich zur Fallunterscheidung kann man im gleichen Raum berechnen, wie die verglichenen Funktionen, da diese höchstens logarithmischen Raum in ihrer Ausgabelänge benötigen. Somit kann ihre Ausgabe nur polynomiell lang werden, ein Zeiger auf die aktuell zu vergleichende Stelle hat also nur logarithmisch viele Bits. Folglich ist $f \in \mathrm{FL}$.

Weiterhin lässt sich zeigen, dass $W_f = A$ gilt, dass also f ein FL-Beweissystem für A ist: Für beliebige $y \in A$ existiert ein h-Beweis z für $\mathrm{pad}(y, \varepsilon) \in A$. Daher gilt $W_f \supseteq A$. Für $y \in W_f$ existieren außerdem $x, z \in \Sigma^*$ mit $\mathrm{pad}(y, x) = h(z)$. Wegen $W_h = A$ gilt daher $\mathrm{pad}(y, x) \in A$. Da pad die Menge $A \times \Sigma^*$ auf A reduziert, folgt $(y, x) \in A \times \Sigma^*$ und somit $y \in A$. Also gilt auch $W_f \subseteq A$.

Es bleibt zu zeigen, dass f l-optimal ist, wenn h auf L-Teilmengen in FL invertiert werden kann. Sei dazu g ein beliebiges FL-Beweissystem für A. Eine Menge H_g soll nun die Paare an Ein- und Ausgaben von g enthalten:

$$H_g = \{\langle g(x), x\rangle \mid x \in D_g\}$$

Da g in FL auf der im Tupel definierte Eingabe berechnet werden kann, gilt $H_g \in \mathrm{L}$. Da pad L-Teilmengen wieder auf Teilmengen abbildet, ist dann auch $H'_g = \mathrm{pad}(H_g) \subseteq A$ in L und somit ist h nach Annahme auf H'_g mit einer Funktion $t \in \mathrm{FL}$ invertierbar. Dann gilt

$$h(t(\mathrm{pad}(g(x), x))) = \mathrm{pad}(g(x), x)$$

und nach Definition von f folgt

$$g(x) = f(\underbrace{\langle g(x), x, t(pad(g(x), x))\rangle}_{\chi})$$

Da χ die Verkettung von FL-Funktionen ist, ist $\chi \in \mathrm{FL}$. Somit wird g von f simuliert, folglich wird g von f l-simuliert. ■

Ein Ringschluss liefert nun die behauptete Äquivalenz. □

Bekanntes Resultat 8 (Messner, Theorem 3.19)**.** Sei $A \times \Sigma^* \leq^p_{m,\mathrm{P}-\mathrm{P}} A$. Dann sind die folgenden Aussagen äquivalent.

(i) Es existiert ein optimales FP-Beweissystem für A.

(ii) Es existiert ein FP-Beweissystem für A, welches kurze Beweise auf jeder NP-Teilmenge von A hat.

(iii) Es existiert ein FP-Beweissystem für A, welches kurze Beweise auf jeder P-Teilmenge von A hat.

Die Implikationen $(i) \implies (ii)$ und $(ii) \implies (iii)$ gelten außerdem für beliebige Sprachen.

Satz 3.13. *Sei* $A \times \Sigma^* \leq^{\log}_{m,\mathrm{L}-\mathrm{L}} A$. *Dann sind die folgenden Aussagen äquivalent.*

(i) Es existiert ein optimales FL*-Beweissystem für* A.

(ii) Es existiert ein FL*-Beweissystem für* A*, welches kurze Beweise auf jeder* NL*-Teilmenge von* A *hat.*

(iii) Es existiert ein FL*-Beweissystem für* A*, welches kurze Beweise auf jeder* L*-Teilmenge von* A *hat.*

Die Implikationen $(i) \Longrightarrow (ii)$ *und* $(ii) \Longrightarrow (iii)$ *gelten außerdem für beliebige Sprachen.*

Beweis.

Behauptung 3.14. $(i) \Longrightarrow (ii)$

Beweis. Sei h ein optimales FL-Beweissystem für A und $S \subseteq A$ beliebig in NL. Sei M eine NL-Maschine, die S akzeptiert.

$$h'(x) = \begin{cases} h(w) & \text{falls } x = 1w \\ z & \text{falls } x = 0\,\langle w, z\rangle \text{ und } w \text{ ist akzeptierender Pfad von } M(z) \\ \bot & \text{sonst} \end{cases}$$

h' ist ein FL-Beweissystem für A. Zusätzlich kann für jedes $y \in S$ ein polynomiell langer Beweis x mit $h'(x) = y$ gefunden werden, da Rechenwege von M nur polynomiell lang sein können. Da h nach Annahme optimal ist, hat auch h auf S polynomiell lange Beweise. ■

Behauptung 3.15. $(iii) \Longrightarrow (i)$

Beweis. Sei $A \times \Sigma^* \leq^{\log}_{m,\mathrm{L}-\mathrm{L}} A$ mit einer Reduktionsfunktion $\mathrm{pad} \in \mathrm{FL}$ und sei h ein FL-Beweissystem für A mit höchstens polynomiell langen Beweisen für beliebige L-Teilmengen von A. Sei erneut

$$f(\langle y, x, z\rangle) = \begin{cases} y & \text{falls } h(z) = \mathrm{pad}(y, x) \\ \bot & \text{sonst} \end{cases}$$

Für jedes beliebige FL-Beweissystem g für A gilt nun:

$$H_g = \{\langle g(x), x\rangle \mid x \in D_g\} \in \mathrm{L}$$

Nach Annahme ist somit auch $H'_g = \mathrm{pad}(H_g)$ in L, es existieren also Beweise von höchstens polynomieller Länge auf H'_g. Sei i eine Funktion, die solche polynomiell langen Beweise angibt. Für $y \in H'_g$ gilt folglich $y = h(i(y))$. Man erhält

$$g(x) = f(\underbrace{\langle g(x), x, i(\mathrm{pad}(g(x), x))\rangle}_{\chi})$$

Da χ die Verkettung von höchstens polynomiell Verlängernden Funktionen ist, ist χ selbst höchstens polynomiell länger als x. Somit wird g von f simuliert, f ist also ein optimales FL-Beweissystem für A. ■

Da mit $\mathrm{L} \subseteq \mathrm{NL}$ bereits $(ii) \Longrightarrow (iii)$ gilt, folgt die behauptete Äquivalenz. □

3.5. Beliebig komplexe Sprachen ohne l-optimales Beweissystem

Im Folgenden werden wir zunächst Mengen konstruieren, für die kein l-optimales Beweissystem existiert. Dazu können die Implikationen aus Satz 3.7 verwendet werden. Wir diagonalisieren gegen alle deterministischen Turingmaschinen und erreichen so, dass für die konstruierte Sprache kein raumoptimaler Akzeptor existieren kann. Dann erhalten wir aus der Kontraposition von Implikation $(i) \Rightarrow (iii)$, dass kein l-optimales Beweissystem für die Sprache existieren kann.

Bekanntes Resultat 9 (Messner, Theorem 3.20)**.** Sei $t : \mathbb{N} \to \mathbb{N}$ superpolynomiell und zeitkonstruierbar. Dann existiert eine Menge $A \in \mathrm{DTIME}(t)$ ohne zeitoptimalen deterministischen Akzeptor.

Satz 3.16. *Sei $s : \mathbb{N} \to \mathbb{N} \in \omega(\log n)$ raumkonstruierbar. Dann existiert eine Menge $A \in \mathrm{DSPACE}(s)$ ohne raumoptimalen deterministischen Akzeptor.*

Beweis. Sei $M_1, M_2, \ldots$ eine Aufzählung aller deterministischen Turingmaschinen und sei U eine TM, die für Eingaben der Form 0^i1x die Arbeit von M_i auf 0^i1x simuliert. Nach Neary und Woods [NW12] existiert eine solche universelle Maschine, sodass die Simulation mit linearem Overhead im benötigten Raum möglich ist. Daher gilt:

$$\mathrm{space}_U(0^i1x) \leq c_i \cdot \mathrm{space}_{M_i}(0^i1x) + c_i$$

Für jede Maschine M_i definieren wir nun eine Menge A_i, wobei wir die Sprache zum regulären Ausdruck 0^i10^* mit $L(0^i10^*)$ bezeichnen.

$$A_i = \left\{ x \in L(0^i10^*) \mid U \text{ akzeptiert } x \text{ nicht in Raum } s(|x|) \right\}$$

Wir erhalten nun mit $A = \bigcup_{i \in \mathbb{N}} A_i$ die gewünschte Sprache. Um A zu entscheiden, kann eine Maschine zunächst die Form der Eingabe prüfen. Mit einem endlichen Automaten geht das in konstantem Raum. Für die Simulation von U ist zusätzlich ein Schrittzähler nötig, um Endlosschleifen zu erkennen, in denen die Raumschranke nie verletzt wird. Dieser Zähler kann, genau wie die Simulation selbst, in $\mathcal{O}(s)$ Raum implementiert werden. Daher liegt A in $\mathrm{DSPACE}(s)$.

Behauptung 3.17. *Für jeden Akzeptor M_j von A kann A_j in konstantem Raum entschieden werden.*

Beweis. Es muss $A_j = L(0^j10^*)$ gelten. Nach Definition gilt bereits die Inklusion $A_j \subseteq L(0^j10^*)$. Wären die beiden Mengen verschieden, so hätten wir also einen Zeugen $w \in L(0^j10^*)$, der von M_j nicht akzeptiert wird. Da U auf w die Arbeit von M_j simuliert, kann w von U nicht akzeptiert werden. Dann akzeptiert U auch nicht innerhalb der Raumschranke s und wir erhalten $w \in A_j$ im Widerspruch zur Bedingung $L(M_j) = A$. Für den regulären Ausdruck 0^j10^* existiert ein endlicher Automat [HU79] D, der in konstantem Raum $L(0^j10^*)$ entscheidet. Kombiniert man nun M_j mit D, so erhält man einen Akzeptor von A, der auf A_j nur konstanten Raumbedarf hat. ■

Ein Akzeptor M_j von A benötigt auf A_j mindestens $\Omega(s)$ Speicher, denn für $w \in A_j$ gilt $s(|w|) < \text{space}_U(w) \leq c_j \cdot \text{space}_{M_j}(w) + c_j$. Ein raumoptimaler Akzeptor kann wegen Behauptung 3.17 für Eingaben aus A_j nur $\mathcal{O}(\log n)$ Raum verwenden. Da nach Annahme $s \in \omega(\log n)$ ist, kann M_j kein optimaler Akzeptor sein. □

Bekanntes Resultat 10 (Messner, Korollar 3.22)**.** Sei $t : \mathbb{N} \to \mathbb{N}$ superpolynomiell und zeitkonstruierbar. Dann hat keine Sprache, die $\leq_m^p$-hart für DTIME(t) ist, ein p-optimales Beweissystem.

Korollar 3.18. *Sei $s : \mathbb{N} \to \mathbb{N} \in \omega(\log n)$ raumkonstruierbar. Dann hat keine Sprache, die $\leq_m^{\log}$-hart für* DSPACE(s) *ist, ein l-optimales Beweissystem.*

Beweis. Wir erhalten diese Aussage direkt aus den Abschlusseigenschaften, da die eben konstruierte Sprache auf jede für DSPACE(s) $\leq_m^{\log}$-harte Sprache reduziert werden kann. □

Bekanntes Resultat 11 (Messner, Korollar 3.23)**.** Keine $\leq_m^p$-harte Sprache für E hat ein p-optimales Beweissystem.

Korollar 3.19. *Keine $\leq_m^{\log}$-harte Sprache für* DSPACE($\log^2 n$) *hat ein l-optimales Beweissystem. Insbesondere hat auch* QBF *kein l-optimales Beweissystem.*

Beweis. Der erste Teil der Aussage ist ein Spezialfall von Korollar 3.18, da $\log^2 \in \omega(\log)$. Außerdem ist QBF eine bekannte $\leq_m^{\log}$-vollständige Menge für PSPACE. Beweise dafür liefern unter anderem Stockmeyer [Sto76] sowie Arora und Barak [AB09]. QBF ist folglich auch $\leq_m^{\log}$-hart für DSPACE($\log^2$). □

In leicht abgewandelter Form können wir die Technik aus dem Beweis zu Satz 3.16 sogar dafür verwenden, beliebig komplexe tally-Sprachen ohne l-optimales Beweissystem zu konstruieren.

Bekanntes Resultat 12 (Messner, Theorem 3.26)**.** Seien $t, f : \mathbb{N} \to \mathbb{N}$ zeitkonstruierbar, f injektiv und t superpolynomiell. Dann existiert eine f-tally Menge $A \in$ DTIME(t) ohne zeitoptimalen Akzeptor.

Satz 3.20. *Seien $s : \mathbb{N} \to \mathbb{N} \in \omega(n)$ raumkonstruierbar und $f : \mathbb{N} \to \mathbb{N}$ injektiv und zeitkonstruierbar. Dann existiert eine f-tally Menge $A \in$* DSPACE(s) *ohne raumoptimalen Akzeptor.*

Beweis.

Behauptung 3.21. *Für injektive zeitkonstruierbare f ist auch $g : 0^n \to 0^{f(n)}$ injektiv. Außerdem kann g in linearem Raum invertiert werden.*

Beweis. Da f nach Annahme zeitkonstruierbar ist, existiert eine Maschine, die auf Eingaben 1^n den Funktionswert $f(n)$ in genau $f(n)$ Schritten berechnet. Da f injektiv ist, muss die Maschine die gesamte Eingabe einlesen. Dazu sind bereits n Schritte nötig. Wir erhalten daher $f(n) \geq n$ und können so einschränken, welche Werte als Inverses in Frage kommen. Auf diesen kann dann eine inkrementelle Suche ausgeführt werden.

Algorithmus 3 : f^{-1}

```
   Eingabe : 0^m
1  für i = 0 bis m tue
2  |  wenn f(i) = m dann
3  |  |  gib i aus
```

In Schritten 1 und 3 ist je $\log m$ Raum ausreichend, um i und einen Zeiger auf die Ausgabe von f zu speichern. Außerdem muss in Schritt 2 f berechnet werden. Da f zeitkonstruierbar ist, ist das in $\mathcal{O}(m)$ Raum möglich, denn der Speicherbedarf ist durch den berechneten Funktionswert beschränkt. Falls die Berechnung von $f(i)$ mehr als m Speicher benötigt, so auch mehr als m Takte. Da f zeitkonstruierbar ist, kann also nicht $f(i) = m$ gelten, weshalb die Berechnung frühzeitig abgebrochen werden kann. Insgesamt findet man das Inverse in $\mathcal{O}(m)$ Raum. Falls $m \notin W_f$, so braucht die Suche auch nur $\mathcal{O}(m)$ Raum und terminiert erfolglos. ■

Sei U eine universelle TM, die auf Eingaben $\langle i, x\rangle$ die Arbeit von $M_i(x)$ simuliert. Dabei soll $\text{space}_U(\langle i, x\rangle) \leq c_i \cdot \text{space}_{M_i}(x) + c_i$ für eine Konstante c_i gelten. Sei nun A_i definiert als

$$A_i = \left\{ x = 0^{f(\langle i,n\rangle)} \mid n \in \mathbb{N},\ U \text{ akzeptiert } \langle i, x\rangle \text{ nicht in Raum } s(|x|) \right\}$$

(A_i ist wohldefiniert, da f injektiv ist und somit für jedes Wort das Prädikat von nur einer Maschinennummer abhängt). Zunächst beobachten wir, dass $A = \bigcup_{i\in\mathbb{N}} A_i$ tatsächlich f-tally ist. Außerdem können wir für ein $0^{f(\langle i,n\rangle)}$ in $\mathcal{O}(f(\langle i, x\rangle))$ Raum die Maschinennummer i berechnen und danach M_i auf der Eingabe mit Raumschranke s simulieren. Da s asymptotisch superlinear wächst, erhalten wir insgesamt $A \in \text{DSPACE}(s)$.

Wie im Beweis zu Satz 3.16 gilt für jeden Akzeptor M_j von A die Ungleichung $s(|x|) \leq \text{space}_U(\langle j, x\rangle) \leq c_j \cdot \text{space}_{M_j}(x) + c_j$ für beliebige Eingaben x. Damit erhalten wir $A_j = \{0^{f(\langle j,n\rangle)} \mid n \in \mathbb{N}\}$. A_j kann daher in linearem Raum entschieden werden, indem für Eingaben 0^m das eindeutige n mit $m = f(\langle j, n\rangle)$ berechnet wird. Da f in linearem Raum invertiert werden kann, ist dafür $\mathcal{O}(n)$ Speicher ausreichend. Falls ein solches n existiert, so gilt bereits $0^m \in A_j$. Es existiert also ein Akzeptor für A, der auf A_j höchstens linearen Speicher benötigt. Es gilt jedoch auch $s(x) \leq c_j \cdot \text{space}_{M_j}(x) + c_j$, unter der Annahme $s \in \omega(n)$ hat also M_j einen superlinearen Raumbedarf auf A_j . Somit kann M_j kein optimaler Akzeptor sein. □

Sprachen ohne optimales Beweissystem Es ist zunächst unklar, ob auch beliebig komplexe Sprachen ohne optimales FL-Beweissystem existieren. Messner [Mes01] zeigt, dass die Existenz von optimalen FP-Beweissystemen äquivalent zur Existenz eines nichtdeterministischen zeitoptimalen Akzeptors ist. Für optimale FL-Beweissysteme ist ein Zusammenhang mit nichtdeterministischen raumoptimalen Akzeptoren jedoch nicht so leicht möglich. Obwohl wir die gezeigte Konstruktion auch für den nichtdeterministischen Fall anpassen könnten, weist das also nicht die Existenz von beliebig komplexen Sprachen ohne optimales Beweissystem nach. Tatsächlich werden wir aber aus Satz 4.4 dennoch solche Sprachen erhalten, da wir die Nichtexistenz von optimalen FP-Beweissystemen für die von Messner konstruierten Sprachen direkt auf die Nichtexistenz von optimalen FL-Beweissystem übertragen können.

3.6. Beliebig komplexe Sprachen mit raumoptimalem Akzeptor

Wir versuchen nun im Folgenden ein Indiz für die Existenz beliebig komplexer Sprachen mit l-optimalem FL-Beweissystem zu finden. Es ist unklar, ob die hier konstruierte Sprache den Anforderungen aus Satz 3.13 genügt, daher ist dieses Resultat für sich nicht ausreichend, um eine definitive Antwort zu erhalten.

Bekanntes Resultat 13 (Messner, Theorem 3.32)**.** Sei $t : \mathbb{N} \to \mathbb{N}$ monoton steigend und raumkonstruierbar. Es existiert eine Menge $A \in \mathrm{DTIME}(n^3 \cdot t(n) \log t(n)) \setminus \mathrm{DTIME}(t(n))$, sodass für jede TM M mit $L(M) = A$ mit höchstens endlich vielen Ausnahmen für $x \in A$ gilt: $\mathrm{time}_M(x) > t(|x|)$.

Satz 3.22. *Sei $s : \mathbb{N} \to \mathbb{N} \in \Omega(n \log n)$ raumkonstruierbar und monoton steigend. Dann existiert eine Menge $A \in \mathrm{DSPACE}(s(n))$ mit raumoptimalem deterministischem Akzeptor.*

Beweis. Sei $M_1, M_2, \ldots$ eine Aufzählung aller Turingmaschinen, wobei eine universelle Turingmaschine $U(\langle i, x\rangle)$ die Arbeit von M_i auf der Eingabe x mit $\mathrm{space}_U(\langle i, x\rangle) \leq c_i \cdot \mathrm{space}_U(x) + c_i$ simulieren kann. Wir definieren außerdem eine Menge S als

$$S = \{\langle i, 0^n\rangle \mid U \text{ akzeptiert } \langle i, 0^n\rangle \text{ in Raum höchstens } s(n)\} \in \mathrm{DSPACE}(s)$$

und verwenden S als „Ausschlusskriterium“, um nun die Menge A stufenweise zu konstruieren. Dabei entscheiden wir in jeder Stufe n, ob das Wort 0^n zu A gehört, sodass $0^n \in A_n \iff 0^n \in A$. Damit jeder Akzeptor von A für höchstens endlich viele Worte unterhalb der Raumschranke arbeiten kann, betrachten wir während der Konstruktion eine endliche Menge B_n von solchen Maschinen, die noch als Akzeptor in Frage kommen. Akzeptiert eine Maschine aus B_n ein Wort 0^n unterhalb der Raumschranke, so kann sie als Akzeptor ausgeschlossen werden, indem $0^n \notin A_n$ gesetzt wird.

Zu Beginn der Konstruktion setzen wir $A_0 = B_0 := \emptyset$ sowie $i_0 = 0$. Weiterhin unterscheiden wir für $n \in \mathbb{N}$ zwei Fälle. Falls ein $i \in B_n$ mit $\langle i, 0^n \rangle \in S$ existiert, so setzen wir

$$\begin{aligned} A_{n+1} &= A_n \\ B_{n+1} &= B_n \setminus \{i \in B_n \mid \langle i, 0^n \rangle \in S\} \\ i_{n+1} &= i_n \end{aligned}$$

und setzen sonst

$$\begin{aligned} A_{n+1} &= A_n \cup \{0^n\} \\ B_{n+1} &= B_n \cup \{i_n + 1\} \\ i_{n+1} &= i_n + 1 \end{aligned}$$

Wir erhalten nun A durch $A = \bigcup_{i \in \mathbb{N}} A_i$.

Behauptung 3.23. *A ist eine unendliche Menge und es gilt $\{i_n \mid n \in \mathbb{N}\} = \mathbb{N}$.*

Beweis. Für einen beliebigen Schritt $n \in \mathbb{N}$ ist B_n eine endliche Menge. Im ersten Fall gilt $|B_{n+1}| < |B_n|$. Daher kann höchstens für endlich viele m aufeinanderfolgende Schritte der erste Fall eintreten, bevor $B_{n+m} = \emptyset$ gilt. Spätestens dann wird ein weiteres Element zu A_{n+m+1} hinzugefügt. Mit $0^{n+m+1} \in A_{n+m+1} \implies 0^{n+m+1} \in A$ folgt, dass auch nach keinem endlichen Konstruktionsschritt $A_i = A$ gelten kann, da nach endlich vielen Schritten ein weiteres Element hinzugefügt wird. Mit dem gleichen Argument erhalten wir auch $\{i_n \mid n \in \mathbb{N}\} = \mathbb{N}$. ■

Behauptung 3.24. *A ist* DSPACE($\mathcal{O}(s)$)*-immun.*

Beweis. Sei M_i eine TM mit $\text{space}_{M_i} \in \mathcal{O}(s)$, sodass $L(M_i)$ unendlich ist. Es existiert ein minimaler Schritt n, für den $i_n = i$ gilt. Wir erhalten zudem $c_i \cdot \text{space}_{M_i}(0^m) + c_i \leq s(m)$ für genügend große m, da M_i einen asymptotischen Raumbedarf kleiner s hat. Da M_i eine unendliche Menge akzeptiert, akzeptiert M_i auch 0^m für genügend große m. Sei $m \geq n$ nun minimal, sodass beides gilt. Dann gilt insbesondere auch $\text{space}_U(i, 0^m) \leq s(m)$ und wir wählen in der Konstruktion daher $0^m \notin A$. Folglich akzeptiert M_i keine Teilmenge von A. ■

Behauptung 3.25. *A kann in* DSPACE(s) *entschieden werden.*

Beweis. Um $0^n \overset{?}{\in} A$ zu entscheiden, ist es ausreichend die Konstruktionsschritte bis A_n durchzuführen. A_n hängt dabei nicht von A_{n-1} ab, daher muss $A \cap \Sigma^{<n}$ nicht gespeichert werden. Der Raumbedarf ist dann gegeben durch

$$\underbrace{\log n}_{\text{aktuelle Stufe}} + \underbrace{\sum_{i=1}^{n} \log i}_{B_n} + \underbrace{s(n)}_{\text{Anfragen an } S} + \underbrace{\log n}_{\text{Zeiger in Mengen}}$$

Wegen $\sum_{i=0}^{n} \log i = \log n! \leq \log n^n = n \log n$ ist A_n folglich in $\mathcal{O}(s(n) + n \log n)$ Raum berechenbar. Nach Annahme gilt $s \in \Omega(n \log n)$. ■

Den DSPACE(s)-Akzeptor für A nennen wir O. Für jede Maschine M mit $L(M) = A$ existiert eine Konstante c mit $c \cdot \text{space}_M(x) \geq s(|x|)$ für fast alle $x \in A$, da A DSPACE($\mathcal{O}$(s))-immun ist. Wählt man c ausreichend groß, so gilt sogar $c \cdot (\text{space}_M(x) + |x|) \geq s(|x|)$ für alle $x \in A$. Gleichzeitig benötigt O nur $\text{space}_O(x) \leq c's(|x|) \leq cc' \cdot (\text{space}_M(x) + |x|)$ Raum. Somit ist O ein optimaler Akzeptor von A. □

Die so konstruierten Sprachen haben zwar einen raumoptimalen deterministischen Akzeptor aber dennoch ist unklar, ob Satz 3.13 anwendbar ist. Denn die konstruierte Sprache lässt kein padding zu, da sie keine unendliche in L entscheidbare Teilmenge hat. Das ist aber für padding nötig, da jede L-Teilmenge von $A \times \Sigma^*$ injektiv auf A abgebildet werden müsste. Es fehlt daher ein Beweis ob dennoch $A \times \Sigma^* \leq^{\log}_{m,\mathrm{L}-\mathrm{L}} A$. Daher geben wir noch eine hinreichende Bedingung für die Existenz l-optimaler FL-Beweissysteme an.

Satz 3.26. *Aus* L = NP *folgt die Existenz beliebig komplexer Sprachen mit l-optimalem Beweissystem.*

Beweis. Aus L = P erhalten wir nach [Wra76] zunächst FL = FP und können daher die Behauptung auf die Existenz beliebig komplexer FP-Beweissysteme zurückführen. Mit P = NP folgt dann nach Messner [Mes01] die Behauptung. □

3.7. Ein natürliches Beweissystem für GAP

Wir wissen nach 3.4, dass ein l-optimales FL-Beweissystem für eine vollständige Sprache einer Komplexitätsklasse bereits impliziert, dass jede Sprache dieser Klasse ein l-optimales Beweissystem hat. Daher sind solche Sprachen besonders spannend. Eine sehr natürliche Komplexitätsklasse für solche Betrachtungen ist NL. Wir wissen bereits, dass für deterministische Raumklassen oberhalb von L keine $\leq^{\log}_m$-vollständigen Sprachen mit l-optimalem FL-Beweissystem existieren. Für nichtdeterministische Raumklassen ist jedoch kein solches Resultat bekannt. Wir definieren daher im Folgenden ein natürliches Beweissystem für das NL-vollständige Problem GAP und untersuchen, ob dieses l-optimal ist.

$$\text{gap}(\langle G, s, t, p\rangle) = \begin{cases} \langle G, s, t\rangle & \text{falls } p \text{ ein } s\text{-}t\text{-Pfad im gerichteten Graphen } G \text{ ist} \\ \bot & \text{sonst} \end{cases}$$

Offensichtlich ist $W_{\text{gap}} = \text{GAP}$. Weiterhin kann ein gegebener Pfad in logarithmischem Raum durchlaufen werden, da nur der aktuell besuchte Knoten durch Zeiger in die Eingabe gespeichert werden muss. Dann kann im Pfad für jede Kante geprüft werden, ob eine entsprechende Kante auch im Graphen existiert. Zusätzlich müssen nur Start- und Endknoten verifiziert werden. Daher folgt auch gap ∈ FL.

Wir können eine Gemeinsamkeit mit sat beobachten. Die Schwierigkeit von GAP und SAT liegt jeweils in einer existenziellen Quantifizierung für eine effizient verifizierbare Lösung. Dieser Existenzquantor wird in der Definition von gap sowie von sat umgangen, indem Lösungen explizit angegeben werden müssen. Ist ein solches Beweissystem l- bzw. p-optimal, so kann jeder beliebige Beweis in einen konstruktiven umgewandelt werden. Kann man also ausrechnen, ob eine Lösung existiert, so kann man auch mit geringem Mehraufwand die Lösung explizit berechnen. Diesen Zusammenhang zeigen wir nun für gap formal.

Bekanntes Resultat 14 (Messner, Theorem 5.2)**.** Die folgenden Aussagen sind äquivalent:

1. Für jede nichtdeterministische Polynomialzeitturingmaschine M mit $L(M) = \text{SAT}$ existiert eine Funktion $f \in \text{FP}$, sodass für jede Codierung w eines akzeptierenden Rechenwegs von $M(\varphi)$ die Auswertung von $f(w)$ eine erfüllende Belegung von φ berechnet.
2. Für jede nichtdeterministische Turingmaschine M mit $L(M) = \text{SAT}$ existiert eine Funktion $f \in \text{FP}$, sodass für jede Codierung w eines akzeptierenden Rechenwegs von $M(\varphi)$ die Auswertung von $f(w)$ eine erfüllende Belegung von φ berechnet.
3. sat ist ein p-optimales FP-Beweissystem.

Satz 3.27. *Die folgenden Aussagen sind äquivalent:*

(i) *Für jede NPTM M mit $L(M) = \text{GAP}$ existiert $f \in \text{FL}$, sodass für jede Codierung w eines akzeptierenden Rechenwegs von $M(\langle G, s, t\rangle)$ ein s-t-Pfad in G durch $f(w)$ gegeben ist.*

(ii) *Für jede nichtdeterministische TM M mit $L(M) = \text{GAP}$ existiert $f \in \text{FL}$, sodass für jede Codierung w eines akzeptierenden Rechenwegs von $M(\langle G, s, t\rangle)$ ein s-t-Pfad in G durch $f(w)$ gegeben ist.*

(iii) gap *ist l-optimal.*

Beweis.

Behauptung 3.28. $(iii) \Longrightarrow (ii)$

Beweis. Sei M eine nichtdeterministische TM, die GAP akzeptiert. Man erhält ein FL-Beweissystem für GAP wie folgt:

$$h(w) = \begin{cases} \langle G, s, t\rangle & \text{falls } w \text{ ein akzeptierender RW von } M(\langle G, s, t\rangle) \text{ ist} \\ \bot & \text{sonst} \end{cases}$$

Da nach Annahme gap ein l-optimales Beweissystem ist, existiert ein $f' \in \text{FL}$, mit welchem $\text{gap}(f'(w)) = h(w)$ gilt. Dann ist $f'(w) = \langle G, s, t, p\rangle$. Durch anschließende

Projektion auf das letzte Listenelement erhält man die geforderte Übersetzung von akzeptierenden Rechenwegen in s-t-Pfade. ■

Behauptung 3.29. $(ii) \Longrightarrow (iii)$

Beweis. Sei h ein FL-Beweissystem für GAP. Dann kann eine nichtdeterministische TM M einen möglichen h-Beweis x für ein gegebenes y raten und danach deterministisch prüfen, ob $h(x) = y$ gilt. M akzeptiert so GAP. Nach Annahme existiert also ein $f \in \mathrm{FL}$, sodass für Codierungen w von akzeptierenden Rechenwegen von M $f(w)$ einen s-t-Pfad liefert. Da aus einem h-Beweis auch direkt ein akzeptierender Rechenweg von M folgt, existiert auch ein $f' \in \mathrm{FL}$, welches aus h-Beweisen einen s-t-Pfad liefert. Dann gilt

$$h(x) = \mathrm{gap}(c(h(x), f'(x)))$$

wobei c die Konkatenation zweier Listen leistet. ■

Behauptung 3.30. $(i) \Longrightarrow (iii)$

Beweis. Der Beweis von Behauptung 3.29 ist für diese Implikation noch zu schwach, da h keine kurzen Beweise garantiert. Somit kann M h-Beweise nicht unbedingt in Polynomialzeit raten. Dieses Problem kann umgangen werden, indem in einem weiteren Beweissystem einige ausgegebene Instanzen vergrößert werden. Dazu sei $g(\langle G, i\rangle)$ eine Kopie von G, in die $|i|$ zusätzliche isolierte Knoten eingefügt wurden. Da dazu nur G kopiert und ein Zähler der Länge $\log |i|$ nötig sind, ist $g \in \mathrm{FL}$. Es gilt $|g(G, i)| \geq |i|$ für beliebige G, i. Sei nun h' ein Beweissystem wie folgt:

$$h'(x) = \begin{cases} \langle g(G, w), s, t\rangle & \text{falls } x = 1w \text{ und } h(w) = \langle G, s, t\rangle \\ \langle G, s, t\rangle & \text{falls } x = 0w \text{ und } \mathrm{gap}(w) = \langle G, s, t\rangle \\ \bot & \text{sonst} \end{cases}$$

Ausgaben von h' sind mindestens so lang wie die Eingabe. Folglich arbeitet M aus dem Beweis zu $(ii) \Longrightarrow (iii)$ in nichtdeterministischer Polynomialzeit. Aus dem gleichen Beweis erhält man dann auch unter Annahme von (i) ein solches $f' \in \mathrm{FL}$, welches aus h'-Beweisen s-t-Pfade berechnet. Da ein korrekter Pfad für $h'(1x)$ auch ein korrekter Pfad für $h(x)$ ist, gilt schließlich

$$h(x) = \mathrm{gap}(c(h(x), f'(1x)))$$

■

Da nach Definition bereits $(ii) \Longrightarrow (i)$ gilt, folgt mit einem Ringschluss die behauptete Äquivalenz. □

4. Besonderheiten von FL-Beweissystemen

4.1. L-Optimalität von gap impliziert den Kollaps L = NL

Ein Beweis von Pudlák [Pud17] zeigt, dass das Faktorisieren von Zahlen in deterministischer Polynomialzeit gelöst werden kann, falls das Standardbeweissystem von SAT p-optimal ist. Für den Beweis werden aussagenlogische Formeln genutzt, um auszudrücken, dass eine Zahl n entweder prim ist oder einen echten Teiler besitzt. Da eine solche Formel immer wahr ist, erhält man ein Beweissystem für SAT mit kurzen Beweisen für diese Formeln. Ist sat aber p-optimal, so lassen sich diese kurzen Beweise in sat-Beweise übersetzen, woraus ein echter Faktor für die betrachtete Zahl berechnet werden könnte.

Das gleiche Prinzip soll nun auf gap erweitert werden. Dazu wird zunächst eine Funktion definiert, durch welche die Rechnung einer Turing-Maschine als Graph dargestellt werden kann. Diese Konstruktion wird üblicherweise dazu verwendet, zu zeigen, dass GAP NL-vollständig ist.

Definition 4.1 ([Pap94])**.** *Für eine* NL*-Rechnung $M(x)$ wird der entsprechende* Konfigurationsgraph *$G_M(x)$ genannt. Die Knoten dieses Graphen stellen die möglichen Konfigurationen der Maschine dar. Auf einem Arbeitsband sind $|\Sigma|^{c\cdot\log|x|}$ verschiedene Inhalte und $\log|x|$ Kopfpositionen möglich. Weiterhin sind auf dem Eingabeband $|x|$ verschiedene Kopfpositionen möglich. Zusätzlich hat die Maschine konstant viele Zustände und Arbeitsbänder. Insgesamt erhält man daher $p(|x|)$ mögliche Konfigurationen für ein Polynom p. Diese sind daher jeweils durch $c\cdot\log|x|$ Bits codierbar, folglich ist die Codierung der Knotenmenge von $G_M(x)$ nur polynomiell lang. Wegen $|E| \leq |V|^2$ hat damit die Codierung des gesamten Konfigurationsgraphen nur polynomielle Größe.*

Die Kanten des Graphen beschreiben, ob ein Übergang zwischen zwei Konfigurationen existiert. Für zwei Konfigurationen c_1, c_2 existiert also die Kante (c_1, c_2) genau dann, wenn c_1 in einem Schritt in c_2 übergehen kann. Diese Kantenmenge kann man in logarithmischem Raum berechnen, da man je zwei Konfigurationen gleichzeitig im Speicher halten kann. Für jede Konfiguration gibt es nur konstant viele Folgekonfigurationen, diese müssen mit der zweiten Konfiguration verglichen werden. Iteriert man so über alle möglichen Konfigurationen, so erhält man die Kantenmenge. Da jede Konfiguration nur logarithmischen Raum benötigt, gilt folglich $G_M \in$ FL.

Satz 4.2. gap *ist genau dann l-optimal, wenn* L = NL *gilt.*

Beweis. Sei $A \in$ NL beliebig. Nach Immerman und Szelepcsényi [Imm88] erhalten wir mit NL = coNL die Existenz von zwei NLTMs M_A und $M_{\overline{A}}$ mit $L(M_A) = A$ und $L(M_{\overline{A}}) = \overline{A}$. Ohne Beschränkung der Allgemeinheit kann angenommen werden, dass

M_A und $M_{\overline{A}}$ auf dem gleichen Arbeitsalphabet arbeiten, gleich viele Arbeitsbänder verwenden und jeweils nur eine akzeptierende Konfiguration haben. Durch Anpassen der Zustandsmengen kann zusätzlich sichergestellt werden, dass beide Maschinen die gleiche akzeptierende Konfiguration Acc einnehmen, sonst aber disjunkte Konfigurationsmengen aufweisen.

Eine dritte NLTM M kann nun aus ihrer Startkonfiguration S nichtdeterministisch in die Startkonfigurationen von M_A und $M_{\overline{A}}$ übergehen. Dadurch ist garantiert, dass M auf jeder Eingabe akzeptiert, da $A \cup \overline{A} = \Sigma^*$. Weiterhin sind A und $\overline{A}$ disjunkt, daher kann in M immer nur eine der beiden simulierten Maschinen akzeptieren. Aus der zweiten Konfiguration eines akzeptierenden Rechenwegs von M kann abgelesen werden, ob ein Pfad der Rechnung von M_A oder von $M_{\overline{A}}$ vorliegt. Auf keinem Rechenweg von M wird die Simulierte Maschine nach dem ersten Takt geändert. Daher kann bereits aus der zweiten Konfiguration eines akzeptierenden Rechenweges entschieden werden, ob die Eingabe in A liegt.

Wegen $L(M) = \Sigma^*$ ist in $G_M(x)$ Acc immer von S aus erreichbar. Daher gilt $W_{G'_M} \subseteq$ GAP, wobei $G'_M(x) = \langle G_M(x), S, \text{Acc}\rangle$. Sei nun g ein FL-Beweissystem für GAP wie folgt:

$$g(x) = \begin{cases} \text{gap(w)} & \text{falls } x = 1w \\ G'_M(w) & \text{falls } x = 0w \end{cases}$$

Da gap l-optimal ist, existiert $t \in$ FL, welches g-Beweise in gap-Beweise übersetzt. Da $G'_M(w)$ nach Konstruktion in GAP liegt, ist $0w$ bereits ein g-Beweis für $G'_M(w)$. Aus $t(0w)$ erhält man folglich einen gap-Beweis für $G'_M(w)$. Nach Definition von gap enthält $t(0w)$ einen S-Acc-Pfad in $G_M(w)$. Eine L-Maschine kann nun wie folgt A entscheiden:

Algorithmus 4 : Entscheidungsalgorithmus für A

```
Eingabe : x
Berechne y := t(0x)
Setze p auf den in y codierten S-Acc-Pfad
wenn der Nachfolgeknoten von S in p die Startkonfiguration von M_A ist dann
    akzeptiere
sonst
    lehne ab
```

Es ist ausreichend, nur eine Konfiguration des Pfades im Speicher zu halten. Daher muss y nie vollständig auf ein Arbeitsband geschrieben werden. Die Codierung einzelner Konfigurationen kann in logarithmischem Raum abgespeichert werden. Da auch t und das Inverse der Listencodierung in FL berechenbar sind, arbeitet der gegebene Algorithmus in logarithmischem Raum. Folglich ist $A \in$ L.

Angenommen, es gilt L = NL. Sei GAP$'$ definiert als

$$\text{GAP}' = \{(G, s, t, k) \mid \text{ in gerichtetem Graph } G \text{ existiert ein } s\text{-}t\text{-Pfad der Länge } k\}$$

Da ein Pfad der Länge k in einem gerichteten Graphen von einer NLTM geraten und

dann geprüft werden kann, ist GAP′ ∈ NL = L. Für ein gegebenes $(G, s, t) \in$ GAP kann eine L TM nun wie folgt arbeiten:

Algorithmus 5 : GAPSuche

```
   Eingabe : G, s, t
1  (V, E) = G
2  Bestimme mittels binärer Suche das kleinste k mit (G, s, t, k) ∈ GAP′
3  Knoten v = s
4  solange k > 0 tue
5      für u mit (v, u) ∈ E tue
6          wenn (G, v, t, k − 1) ∈ GAP′ dann
7              Schreibe v auf das Ausgabeband
8              v = u
9              k = k − 1
```

Für diesen Algorithmus müssen nur Anfragen an GAP′ beantwortet und zwei Konfigurationen sowie die Restpfadlänge gespeichert werden. Das ist insgesamt möglich, da GAP′ nach Annahme in L entscheidbar ist und da der kürzeste Pfad jeden Knoten höchstens einmal durchläuft, also in seiner Länge polynomiell beschränkt ist. Sollte kein Pfad existieren, so erkennen wir das bereits in Zeile 2. Hier wird dann kein geeignetes $k \leq |V|$ gefunden, auch hierfür ist nur logarithmischer Raum nötig. Folglich können Pfade in G über ein $g \in$ FL berechnet werden. Sei nun f ein FL-Beweissystem für GAP. Aus einem f-Beweis x erhält man einen gap-Beweis, indem man den Pfad $g(f(x))$ berechnet und mit der Instanz $f(x)$ konkateniert. Es folgt $f \leq_s^l$ gap. Also ist gap l-optimal. □

4.2. Zusammenhang zwischen FL- und FP-Beweissystemen

Unsere Definition von FL-Beweissystemen scheint stärkere Anforderungen zu stellen, als die übliche, da FL ⊆ FP gilt und die umgekehrte Inklusion nicht bekannt ist. Üblicherweise wird FL ≠ FP angenommen. Gleichzeitig ist dadurch aber der Optimalitätsbegriff auf FL-Beweissystemen scheinbar schwächer, da ein optimales FP-Beweissystem auch jedes FL-Beweissystem simulieren muss, umgekehrt aber nicht. Es ist daher zunächst unklar, ob zwischen den Optimalitätsbegriffen für FL- und FP-Beweissysteme überhaupt Beziehungen gelten. Wir werden in diesem Abschnitt eine Technik zeigen, mit der viele dieser Beziehungen gezeigt werden können. Hierfür zeigen wir, dass zu beliebigen FP-Beweissystemenen ein FL-Beweissystem konstruiert werden kann, welches die gleiche Sprache beweist. Es ist sogar eine Übersetzung der Beweise in FP möglich.

Lemma 4.3. *Für jedes* FP*-Beweissystem f existiert ein* FL*-Beweissystem ℓ, sodass $W_f = W_\ell$ und $f \leq_s^p \ell$.*

Beweis. Es existiert eine TM M_f, welche zu beliebigen f-Beweisen x in polynomieller Zeit ein $y = f(x)$ berechnet. Folglich existiert ein Polynom t, sodass die Rechenzeit von

M_f durch t beschränkt ist. Ein Rechenweg von $M_f(x)$ kann daher höchstens $t(|x|)$ Konfigurationen durchlaufen. Jede dieser Konfigurationen kann dabei höchstens $t(|x|)$ Raum verwenden. Daher existiert ein Polynom q, sodass Rechenwege von M_f auf x inklusive der vollständigen Bandinhalte mit $q(|x|)$ Bits codiert werden können. Für einen so codierten Rechenweg muss ein Verifier nur überprüfen, ob die einzelnen Übergänge korrekt sind. Dazu sind zwei Zeiger in die Eingabe ausreichend, die in aufeinanderfolgende Konfigurationen zeigen. Ausgehend von einem Fixpunkt kann der Verifier die Unterschiede in den Konfigurationen prüfen und testen, ob die Änderung durch eine korrekte Regelanwendung entstehen konnte. Für diese Prüfung müssen nur Zeiger in die Eingabe gespeichert werden, daher kann ein Verifier die Rechenwege in logarithmischem Raum prüfen. Sei ℓ nun definiert als

$$\ell(w) = \begin{cases} y & \text{falls } w \text{ einen Rechenweg von } M_f(x) \text{ mit Ausgabe } y \text{ codiert} \\ \bot & \text{sonst} \end{cases}$$

Da nur der Rechenweg geprüft und die Ausgabe imitiert werden muss, ist $\ell \in \mathrm{FL}$. Da außerdem für jeden f-Beweis x ein Rechenweg von M_f auf x existiert, der höchstens polynomiell länger ist, gilt $f \leq_s \ell$.

Um schließlich einen f-Beweis x in einen ℓ-Beweis zu überführen ist es ausreichend, M_f auf diesem x zu simulieren und nach jedem Takt seine Konfiguration inklusive der Bandinhalte auszugeben. Das ist in FP möglich. Durch den Rechenweg ist dann ein ℓ-Beweis gegeben. □

Die Aussage kann sogar noch verschärft werden. Die Konstruktion in Lemma 4.3 fordert nur, dass Rechenzeit sowie Speicher polynomiell beschränkt sind und dass ein codierter Rechenweg in Logspace auf Korrektheit geprüft werden kann. Beides ist für NPSV-Beweissysteme der Fall. Für die Übersetzungsfunktion t ist es daher ausreichend, einen akzeptierenden Rechenweg zu suchen. Falls ein solcher existiert, so ist die Ausgabe von f bereits eindeutig, t kann also einen beliebigen akzeptierenden Rechenweg wählen. Falls kein Rechenweg eine Ausgabe macht, so ist f an dieser Stelle nicht definiert. Es ist daher ausreichend, dass t einen festen Wert ausgibt, der keine korrekte Codierung eines Rechenwegs ist. Nur die gefolgerte P-Simulation funktioniert nicht für alle $f \in \mathrm{NPSV}$.

Auch für Relationen $f \in \mathrm{NPMV}$ existiert ein FL-Beweissystem ℓ, sodass für alle x, y mit $y \in f\{x\}$ ein x' existiert, sodass x' höchstens polynomiell länger als x ist und $\ell(x') = y$. Allerdings existiert in diesem Kontext keine Funktion t, die f-Beweise in ℓ-Beweise übersetzt, da ein f-Beweis nicht eindeutig charakterisiert, welches Element bewiesen werden soll. Dadurch ist die Übersetzung t nicht wohldefiniert.

Im Kontext von noch schwächeren Berechnungsmodellen ist die gezeigte Technik nicht anwendbar. Um einen codierten Rechenweg zu prüfen, ist es notwendig, die ursprüngliche Eingabe vollständig in der Eingabe codiert zu haben. Solange für das Prüfen des Rechenwegs Zeiger in die Eingabe gespeichert werden müssen, benötigt die Prüfung eines Rechenwegs w daher $\Omega(\log |w|)$ Speicher. Da die Codierung eines Rechenwegs von $M(x)$ bereits eine Länge von $p(|x|)$ haben kann, ist die Prüfung der codierten Rechenwege folglich nicht unterhalb von L möglich.

Optimale Beweissysteme Aus Lemma 4.3 folgt nun direkt, dass die Existenz optimaler FL-Beweissysteme genau durch die von optimalen FP-Beweissystemen charakterisiert werden kann.

Satz 4.4. *Eine Menge $A \subseteq \Sigma^*$ hat genau dann ein optimales* FL*-Beweissystem, wenn sie ein optimales* FP*-Beweissystem hat.*

Beweis. Seien g ein optimales FL-Beweissystem und f ein beliebiges FP-Beweissystem für A. Lemma 4.3 garantiert ein FL-Beweissystem ℓ für A mit $f \leq_s \ell$. Da g optimal ist, gilt weiterhin $\ell \leq_s g$. Aus der Transitivität der Simulation (Satz 2.25) folgt $f \leq_s g$.

Sei nun f ein optimales FP-Beweissystem für A. Mit ℓ sei das in Lemma 4.3 garantierte FL-Beweissystem bezeichnet. Sei g ein FL-Beweissystem für A. Wegen FL $\subseteq$ FP gilt $g \leq_s f$. Da ℓ wiederum f simuliert, gilt nach der Transitivität der Simulation (Satz 2.25) auch $g \leq_s \ell$. □

Dieser Satz ermöglicht es uns, die Resultate von Messner [Mes01] zur Existenz von optimalen FP-Beweissystemen in den Kontext von FL-Beweissystemen zu übertragen.

Korollar 4.5 (Verbesserung von Satz 3.2)**.** *Jedes $A \in$ NP hat ein optimales* FL*-Beweissystem.*

Beweis. Nach Messner [Mes01] ist bekannt, dass beliebige Sprachen aus NP optimale FP-Beweissysteme haben. □

Korollar 4.6. *Es existieren beliebig komplexe Sprachen ohne optimales* FL*-Beweissystem.*

Beweis. Für superpolynomielle zeitkonstruierbare $t : \mathbb{N} \to \mathbb{N}$ konstruiert Messner [Mes01] Sprachen in coNTIME(t) ohne optimales FP-Beweissystem. □

Korollar 4.7. *Der Kollaps* NP = coNP *impliziert die Existenz beliebig komplexer Sprachen mit optimalem* FL*-Beweissystem.*

Beweis. Messner zeigt diese Implikation für optimale FP-Beweissysteme. □

l-optimale Beweissysteme sind mindestens so schwer zu finden wie p-optimale

Satz 4.8. *Ist f ein l-optimales* FL*-Beweissystem für eine Sprache A, so ist f auch ein p-optimales* FP*-Beweissystem für A.*

Beweis. Sei f ein l-optimales FL-Beweissystem und g ein beliebiges FP-Beweissystem. Nach Satz 4.4 gilt bereits $g \leq_s f$. Weiterhin kann beobachtet werden, dass g von dem in Lemma 4.3 konstruierte FL-Beweissystem ℓ sogar p-simuliert wird. Die Übersetzung kann berechnet werden, indem ein Rechenweg von g auf der Eingabe simuliert und die dabei durchlaufenen Konfigurationen ausgegeben werden. Die Simulation an sich ist in Polynomialzeit möglich, da nach Wahl $g \in$ FP. Für die Ausgabe der einzelnen Konfigurationen ist dann nur noch polynomieller Overhead nötig. Da weiterhin $\ell \leq_s^l f$ gilt und FP unter Verkettung abgeschlossen ist, folgt nun $g \leq_s^p f$. □

Es ist unklar, ob auch die Umkehrung dieser Aussage gilt. Die reine Verwendung von Lemma 4.3 ist hier allerdings nicht ausreichend, da dafür beliebige FP-Berechnungen in FL simuliert und die durchlaufenen Konfigurationen berechnet werden müssten. Ein hinreichendes Kriterium wäre L = P, da dann nach einem Resultat von Jenner und Torán [JT95] FP $\subseteq$ FL folgt.

Wir erhalten auch ein Indiz für eine untere Schranke. Wegen GAP $\in$ P können s-t-Pfade in FP berechnet werden. Daher ist klar, dass GAP bereits ein p-optimales Beweissystem hat. Implizieren p-optimale FP-Beweissysteme nun auch l-optimale FL-Beweissysteme, so erhalten wir ein l-optimales FL-Beweissystem g für GAP. Grundsätzlich wissen wir nicht, wie dieses Beweissystem funktioniert. Insbesondere ist unklar, ob das Standardbeweissystem gap dann l-optimal ist. Lassen sich aber aus g-Beweisen Rückschlüsse auf mindestens einen inneren Knoten eines s-t-Pfades ziehen, so können wir bereits durch den Konfigurationsgraphen wie im Beweis zu Satz 4.2 L = NL folgern. Um diesen Kollaps zu umgehen wäre eine Beweistechnik erforderlich, durch die man keine Informationen über die inneren Knoten eines verbindenden Pfades erhält. Im Gegensatz zu den anderen Richtungen könnte man dann nicht das l-optimale FL-Beweissystem direkt aus einem p-optimalen ausrechnen.

5. Offene Fragen

Wir haben eine enge Verbindung zwischen FL-Beweissystemen und FP-Beweissystemen beobachtet, insbesondere implizieren positive Antworten auf die Existenzfragen für FL-Beweissysteme jeweils auch positive Antworten auf die Existenzfragen bei FP-Beweissystemen. Diese Implikationen könnten dabei helfen, die hinreichenden Kollapsbedingungen von Messner abzuschwächen, da für raumbeschränkte Komplexitätsklassen manche Resultate wie die Abgeschlossenheit unter Komplement bereits bekannt sind. Dadurch stehen im Kontext von FL-Beweissystemen mehr Techniken zur Verfügung, über die eine Verbesserung erzielt werden kann.

Speziell für die Existenz optimaler FL-Beweissysteme wäre eine zusätzliche Charakterisierung über einen raumoptimalen Akzeptor wünschenswert. Der analoge Beweis von Messner lässt sich nicht direkt übertragen, aber ein solcher Zusammenhang ist dennoch nicht ausgeschlossen. Da wir zumindest wissen, dass Funktionen in FL raumoptimal invertiert werden können, scheint ein solcher Zusammenhang nicht ausgeschlossen. Könnte man die Existenz optimaler FL-Beweissysteme so charakterisieren, so würden außerdem die Sprachen mit raumoptimalem Akzeptor mit den Sprachen mit zeitoptimalem Akzeptor zusammenfallen.

Mit Blick auf die $L \stackrel{?}{=} P$ Frage ist schließlich die Umkehrung der Implikation aus Satz 4.8 interessant. Man könnte zeigen, dass diese Implikation nicht gilt, indem eine Sprache mit p-optimalem FP-Beweissystem und ohne l-optimales FL-Beweissystem konstruiert wird. Da unklar ist, für welche Komplexitäten noch Sprachen mit p-optimalem FP-Beweissystem existieren, ist diese Suche vermutlich sehr schwer. Man könnte aber explizit nach einer Sprache in P suchen, die kein l-optimales FL-Beweissystem hat. Findet man eine solche Sprache, so zeigt man damit, dass $L \neq P$. Auch daher ist davon auszugehen, dass dieser Ansatz sehr schwierig ist. Könnte man stattdessen zeigen, dass diese Implikation tatsächlich auch gilt, so könnte man darüber einen Hinweis erhalten, ob gap oder ein ähnliches FL-Beweissystem bereits l-optimal ist. Dass daraus $L = P$ folgt, haben wir bereits in Satz 4.2 gezeigt.

A. Beweissysteme mit konstantem Speicherbedarf

Bisher haben wir beobachtet, dass die Frage nach universellen Beweissystemen auch im schwächeren Berechnungsmodell nicht leichter als mit der üblichen Definition von Cook und Reckhow zu beantworten ist. Daher schwächen wir die betrachteten Beweissysteme im Folgenden noch weiter ab und beschränken uns auf solche Funktionen, die mit dem Berechnungsmodell der *endlichen Automaten* berechnet werden können.

Im Berechnungsmodell von endlichen Automaten wird im Gegensatz zu Turing-Maschinen auf Arbeitsbänder verzichtet. Außerdem ist das Eingabeband ein *read-once* Band, sodass die Eingabe nicht mehrfach durchlaufen werden kann. Eine genaue Definition von endlichen Automaten kann bei Hopcroft und Ullman [HU79] gefunden werden. Endliche Automaten können auch Funktionen berechnen, indem in jedem Übergang zusätzlich eine Ausgabe getätigt wird. Die Konkatenation der Ausgaben ergibt dann einen Funktionswert. Die Menge der Funktionen, die von einem endlichen Automaten berechnet werden können, nennen wir FDEA.

Satz A.1. *Für eine Sprache $A \subseteq \Sigma^*$ sind die folgenden Aussagen äquivalent:*

(i) $A \in$ REG

(ii) *A hat ein* FDEA-*Beweissystem*

(iii) *A hat ein l-optimales* FDEA-*Beweissystem*

Beweis.

Behauptung A.2. $(ii) \Longrightarrow (i)$

Beweis. Gegeben sei ein FDEA-Beweissystem f für A. Sei F ein deterministischer endlicher Automat (DEA), der f berechnet. Ohne Beschränkung der Allgemeinheit können wir annehmen, dass in jedem Übergang von F nur ein Symbol ausgegeben wird. Dazu sind technisch ε-Transitionen nötig, die längere Eingaben über Blockierzustände aufbrechen. Wir erhalten dann einen nichtdeterministischen endlichen Automaten N, der den Wertebereich der von F berechneten Funktion entscheidet, indem für jeden Übergang der Form $z_a s_a \to z_b s_b$ von F ein Übergang $z_a s_b \to z_b$ in N angelegt wird. Durch dieses Vorgehen prüft N, ob eine Eingabe existiert, für die F das bisher gelesene Wort ausgeben kann. Nach einem Satz von Rabin und Scott [RS59] kann N in einen äquivalenten DEA überführt werden. Folglich ist der Wertebereich durch einen endlichen Automaten entscheidbar und daher eine reguläre Sprache [HU79]. ■

Behauptung A.3. $(i) \Longrightarrow (iii)$

Beweis. Sei $A \in \text{REG}$. Folglich existiert ein DEA D mit $L(D) = A$ [HU79]. Wir erhalten einen DEA F, der genau wie D arbeitet und in jedem Übergang das gerade eingelesene Symbol ausgibt. Offenbar berechnet F das FDEA-Beweissystem

$$h(w) = \begin{cases} w & \text{falls } w \in A \\ \perp & \text{sonst} \end{cases}$$

Der Wertebereich von h entspricht nach Definition genau A, da $h(w) = w$ für $w \in A$. Gleichzeitig gilt dann für jedes FDEA-Beweissystem g für A, dass $g(x) = h(g(w))$ für jedes beliebige $x \in \Sigma$. Wegen $g \in \text{FDEA} \subseteq \text{FL}$ ist h also l-optimal. ■

Da $(iii) \Longrightarrow (ii)$ bereits per Definition gilt, folgt nach einem Ringschluss die behauptete Äquivalenz. □

Literaturverzeichnis

[AB09] Sanjeev Arora und Boaz Barak: *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009, 10.1017/CBO9780511804090.

[BDJ11] José L. Balcázar, Josep Díaz und Gabarró Joaquim: *Structural Complexity I*. Springer Berlin, Heidelberg, 2. Auflage, 2011, https://doi.org/10.1007/978-3-642-79235-9.

[Bey12] Olaf Beyersdorff: *Non-classical Aspects in Proof Complexity*. Cuvillier, 2012.

[Bon18] Ilario Bonacina: *Space in Weak Propositional Proof Systems*. Springer Cham, 1. Auflage, 2018.

[Bus12] Samuel R. Buss: Towards NP–P via proof complexity and search. *Annals of Pure and Applied Logic*, 163(7):906--917, 2012, https://doi.org/10.1016/j.apal.2011.09.009, ISSN 0168-0072.

[CR79] Stephen A. Cook und Robert A. Reckhow: The relative efficiency of propositional proof systems. *The Journal of Symbolic Logic*, 44(1):36–50, 1979, 10.2307/2273702.

[FFNR03] Stephen. A Fenner, Lance Fortnow, Ashish V. Naik und John D. Rogers: Inverting onto functions. *Information and Computation*, 186(1):90--103, 2003, https://doi.org/10.1016/S0890-5401(03)00119-6, ISSN 0890-5401.

[Fre23] Anton Freund: An Introduction to Mathematical Logic, 2023. https://doi.org/10.48550/arXiv.2310.09921.

[HS11] Steven Homer und Alan L. Selman: *Computability and Complexity Theory*. Springer New York, NY, 2. Auflage, 2011.

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

[Imm88] Neil Immerman: Nondeterministic Space is Closed under Complementation. *SIAM Journal on Computing*, 17(5):935--938, 1988, 10.1137/0217058. `https://doi.org/10.1137/0217058`.

[JT95] Birgit Jenner und Jacobo Torán: Computing functions with parallel queries to NP. *Theoretical Computer Science*, 141(1):175--193, 1995, https://doi.org/10.1016/0304-3975(94)00080-3, ISSN 0304-3975.

[Lev73] Leonid A. Levin: Universal Sequential Search Problems. *Problems of Information Transmission*, 9(3):265--266, 1973.

[Mes01] Jochen Messner: *On the simulation order of proof systems.* Dissertation, Januar 2001.

[NW12] Turlough Neary und Damien Woods: The Complexity of Small Universal Turing Machines: A Survey. In: Mária Bieliková, Gerhard Friedrich, Georg Gottlob, Stefan Katzenbeisser und György Turán (Herausgeber): *SOFSEM 2012: Theory and Practice of Computer Science*, Seiten 385--405, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg, ISBN 978-3-642-27660-6.

[Pap94] Christos H. Papadimitriou: *Computational complexity.* Addison-Wesley, 1994.

[Pud17] Pavel Pudlák: Incompleteness in the finite domain. *Bulletin of Symbolic Logic*, 23(4):405–441, 2017, 10.1017/bsl.2017.32.

[RS59] M. O. Rabin und D. Scott: Finite Automata and Their Decision Problems. *IBM Journal of Research and Development*, 3(2):114--125, 1959, 10.1147/rd.32.0114.

[Sip80] Michael Sipser: Halting space-bounded computations. *Theoretical Computer Science*, 10(3):335--338, 1980, https://doi.org/10.1016/0304-3975(80)90053-5.

[Sto76] Larry J. Stockmeyer: The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1--22, 1976, https://doi.org/10.1016/0304-3975(76)90061-X, ISSN 0304-3975.

[Wra76] Celia Wrathall: Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):23--33, 1976, https://doi.org/10.1016/0304-3975(76)90062-1, ISSN 0304-3975.