# Blattsprachen und Blattfunktionen

Von der Fakultät für Elektrotechnik und Informatik
der Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des Grades

Doktor der Naturwissenschaften
Dr. rer. nat.

genehmigte Dissertation
von

**Dipl.-Inform. Matthias Galota**

geboren am 10.6.1974 in Würzburg

2006

Referent: Prof. Dr. Heribert Vollmer

Koreferent: Prof. Dr. Ulrich Hertrampf

Tag der Promotion: 16. August 2006

## Kurzzusammenfassung

Diese Dissertation ist im Bereich der Komplexitätstheorie angesiedelt. Sie befasst sich dort mit *Blattsprachen*, einer Verallgemeinerung des Begriffs des *Nichtdeterminismus*. Blattsprachen in verschiedenen Berechnungsmodellen werden untersucht, im Einzelnen:

- Es wird erkundet, wie Blattsprachen dabei helfen können, Trennungen zwischen Komplexitätsklassen zu erzielen:

  Die Trennung zweier Komplexitätsklassen, also der Beweis, dass die beiden Klassen tatsächlich verschieden sind, ist ein schwieriges Problem der Komplexitätstheorie.

  Es wird untersucht, wie man bestimmte Komplexitätsklassen, die durch Blattsprachen beschrieben werden, trennen kann, indem man die Eigenschaften der Blattsprachen direkt ausnutzt.

- Es werden auch Blattsprachen für nichtdeterministische endliche Automaten untersucht. Diese Blattsprachen für endliche Automaten stehen im Zusammenhang mit Phänomenen aus der mathematischen Logik.

  Schon lange ist bekannt, dass ein Zusammenhang zwischen Logik und endlichen Automaten besteht: Monadische logische Formeln der zweiten Stufe erkennen genau die regulären Sprachen. Und so wie die Blattsprachen eine Erweiterung des Nichtdeterminismus in der Komplexitätstheorie sind, sind die *Lindströmquantoren* in der Logik eine Verallgemeinerung der Existenzquantoren.

  Es wird untersucht, wie die Lindströmquantoren mit den Blattsprachen für endliche Automaten im Zusammenhang stehen.

- Auch Komplexitätsklassen von Funktionen werden betrachtet. Dort gibt es einen Begriff, der den Blattsprachen entspricht: die *Blattfunktionen*.

  Eine interessante Erkenntnis auf dem Gebiet der Blattsprachen war die Tatsache, dass es eine reguläre Blattsprache gibt, mithilfe derer die Komplexitätsklasse PSPACE dargestellt werden kann.

  Es wird untersucht, ob es, in Anlehnung an die einfache Blattsprache für PSPACE, eine einfache Blattfunktion für die Funktionenklasse FPSPACE gibt.

## Abstract

This dissertation deals with problems from complexity theory. We take a look at *leaf languages*, which are a generalization of *non-determinism*. Leaf languages in several computation paradigms are examined, in detail:

- We take a look at how leaf languages could help separating certain complexity classes.

  The separation of two complexity classes – the proof that both classes indeed are distinct – usually is a hard problem in complexity theory.

  We examine how certain complexity classes, which are defined by leaf languages, can be separated by directly exploiting properties of the leaf languages.

- We also examine leaf languages for non-deterministic finite automata. Those leaf languages for automata are linked with certain notions from mathematical logic.

  It is long known that there exists a connection between logic and finite automata: monadic second order logical formulas recognize exactly the regular languages. And as the leaf languages are a generalization of non-determinism in complexity theory, the *Lindström quantifiers* are a generalization of the existential quantifier in logic.

  We study how Lindström quantifiers relate to leaf languages for finite automata.

- We also take a look at complexity classes of functions. There, a notion which resembles the leaf languages can be defined: *leaf functions*.

  An interesting result from the field of leaf languages was the fact that there is a regular leaf language which describes the class PSPACE.

  We try to find, in the style of the simple leaf language for PSPACE, a simple leaf function for the function complexity class FPSPACE.

## Schlagworte

- Komplexitätstheorie
- Blattsprachen
- Orakel

## Keywords

- Complexity Theory
- Leaf Languages
- Oracles

Ich möchte all jenen danken, die durch Rat und Tat zum Entstehen dieser Arbeit beigetragen haben. Mein Dank gilt insbesondere meinem Doktorvater Heribert Vollmer für seine Unterstützung und Geduld und Steffen Reith für seine Hilfe und Ungeduld. Außerdem möchte ich Klaus W. Wagner, Sven Kosub, Christian Glaßer, Elmar Böhler und Michael Bauland für Diskussionen, Anregungen und Aufmunterungen danken.

# Inhaltsverzeichnis

# Kapitel 1

# Einführung

Die Komplexitätstheorie beschäftigt sich mit Problemen, die ein Computer bearbeiten kann, den Algorithmen, die diese Probleme lösen – und dem Rechenaufwand, den diese Algorithmen haben. Dabei interessieren wir uns hauptsächlich für den Aufwand an Rechenzeit und Speicherplatz. Um Rechenzeit und Speicherplatz vergleichbar messen zu können, muss man sich zuerst auf eine Rechnerarchitektur einigen, auf der die Algorithmen ausgeführt werden sollen. Um die Aufwandsanalyse möglichst einfach zu machen, sollte diese Rechnerarchitektur möglichst wenige, einfach strukturierte Komponentenen besitzen. Andererseits muss sie auch die volle Berechnungsstärke besitzen, die Computer besitzen können. Ein solches Modell, das wir im Weiteren benutzen werden, ist die *Turing-Maschine* (kurz: *TM*). Sie besteht aus einem unendlich langen Band, das in Felder unterteilt ist, einem Schreib-/Lesekopf, der sich über das Band bewegen kann, und einer Kontrolleinheit, in der ein Programm gespeichert werden kann, das angibt, wie sich der Kopf in Abhängigkeit des Bandinhalts bewegen soll.

Bei der Untersuchung von mathematischen Problemen und Algorithmen für deren Lösung traf man immer wieder auf Probleme, für die man keine effizienten Algorithmen finden konnte. Eines dieser Probleme war das so genannte *Problem des Handlungsreisenden*: Gegeben sind eine Liste von Städten und eine Obergrenze in Kilometern. Die Frage ist nun, ob es eine Rundreise durch alle Städte gibt, die nicht länger ist als die gegebene Obergrenze. Der einzige Algorithmus, den man für dieses Problem fand, besteht im Wesentlichen darin, alle möglichen Rundreisen auszuprobieren und für jede einzelne zu testen, ob sie kurz genug ist. Da die Menge der Rundreisen exponentiell zu der Menge der Städte ist, ist dieser Algorithmus ab einer bestimmten Anzahl von Städten nicht mehr praktikabel. Man traf immer wieder auf solche Probleme, die sich einer effektiven Lösung entzogen. Sie hatten alle die Gemeinsamkeit, dass die einzige Lösungsmöglichkeit das Aus-

probieren schien: Man muss eine in der Eingabe exponentiell große Anzahl von möglichen Lösungen daraufhin testen, ob sie eine bestimmte Bedingung erfüllen, wobei das Abtesten der Bedingung selbst effektiv möglich ist. All diese Probleme sind durch so genannte *nichtdeterministische* Turing-Maschinen effektiv lösbar. Diese nichtdeterministischen Maschinen sind hypothetische Rechnermodelle, die nicht realisierbar sind. Sie sind, grob gesprochen, Parallelrechner, die beliebig viele Prozessoren zur Verfügung haben. Die Rechnung einer nichtdeterministischen Turing-Maschine läuft meistens genau so ab wie die einer üblichen, *deterministischen* TM. Es gibt aber bestimmte Situationen, in denen sich die nichtdeterministische TM *verzweigen* kann; aus einer werden zwei TM, die auf verschiedene Weise weiterrechnen. Auf diese Weise entsteht statt eines Berechnungspfades ein *Berechnungsbaum*. Die vorhin genannten Probleme lassen sich mit diesen hypothetischen Maschinen effektiv lösen. Um z.B das Handlungsreisendenproblem zu lösen, muss die nichtdeterministische Maschine auf den verschiedenen Pfaden ihres Berechnungsbaums die verschiedenen Reiserouten auswählen. Die Antwort auf die Frage „Gibt es eine Reiseroute, die kürzer als die Obergrenze ist?“, ist nun „ja“, falls mindestens ein Berechnungspfad existiert, auf dem eine Reiseroute ausgewählt wurde, die kurz genug ist. Allgemein akzeptiert eine nichtdeterministische TM die Eingabe genau dann, wenn mindestens ein Pfad des Berechnungsbaumes akzeptiert.

Die Menge aller Probleme für die ein effektiver Algorithmus für eine nichtdeterministische TM existiert, nennt man *NP*, dies steht für *nichtdeterministische Polynomialzeit*. Die Menge der Probleme, für die sogar ein effektiver Algorithmus für eine deterministische TM existiert, nennt man *P*. Dies ist die Menge aller Probleme, die wir als effektiv lösbar ansehen. Die vielleicht wichtigste Frage der Komplexitätstheorie ist nun, ob die Mengen P und NP verschieden sind.

Nichtdeterministische Maschinen akzeptieren eine Eingabe, wenn mindestens ein Pfad des Berechnungsbaums die Eingabe akzeptiert. Nun stellte man sich die Frage, was passieren würde, wenn man die Akzeptanzbedingung der Maschine ändern würde. Man kann z.B. festlegen, dass die Eingabe dann akzeptiert wird, wenn die Anzahl der akzeptierenden Pfade gerade ist. Alle Probleme, die durch Polynomialzeit-TM mit einer solchen Akzeptanzbedingung gelöst werden können, fasst man in der Klasse $\oplus$P zusammen. Die Akzeptanzbedingung „mindestens die Hälfte aller Pfade muss akzeptieren“ führt zur Klasse PP. Die Akzeptanzbedingung „der erste Pfad muss akzeptieren“ führt zur Klasse P.

Man kann die Akzeptanzbedingung nun beliebig verallgemeinern, indem man die Ausgaben der Maschine auf den einzelnen Pfaden des Berechnungsbaumes als Zeichenkette interpretiert und akzeptiert, wenn die Zeichenkette

in einer bestimmten Sprache liegt, der so genannten *Blattsprache*. Dadurch kann jeder Blattsprache eine Komplexitätsklasse zugeordnet werden. Diese Idee der Blattsprachen haben zu Beginn der neunziger Jahre des vorigen Jahrhunderts Bovet, Creszenzi und Silvestri in [BCS92] und parallel dazu Vereshchagin in [Ver93] eingeführt.

In den drei Hauptteilen dieser Arbeit werden wir uns mit verschiedenen Aspekten von Blattsprachen beschäftigen.

- In Kapitel 3 wollen wir erkunden, wie Blattsprachen dabei helfen können, Trennungen zwischen Komplexitätsklassen zu erzielen:

  Die Trennung zweier Komplexitätsklassen, also der Beweis, dass die beiden Klassen tatsächlich verschieden sind, ist ein schwieriges Problem der Komplexitätstheorie. Um dies zu bewerkstelligen, muss man mindestens eine Menge finden, die in einer der beiden Klassen liegt, aber nicht in der anderen. Der Beweis der Zugehörigkeit zu einer Klasse ist der einfachere Teil: Es genügt, einen Algorithmus für das Problem anzugeben, der die Ressourcenbeschränkung der Klasse einhält. Der Nachweis, dass eine Menge nicht in einer Komplexitätsklasse liegt, kann jedoch nur gelingen, wenn man zeigt, dass alle Algorithmen, die die Menge akzeptieren, die Ressourcenbeschränkung nicht erfüllen. D.h. also, man muss eine *untere Schranke* für die Komplexität eines Problems finden. Es gibt nur wenige Methoden, mit denen ein solcher Nachweis gelingt, und diese Methoden funktionieren auch nur in wenigen Fällen.

  In diesem Kapitel soll nun eine Methode beschrieben werden, mit der man bestimmte Komplexitätsklassen trennen kann, die durch Blattsprachen beschrieben werden. Diese Trennung beruht darauf, dass die Blattsprachen, die die Klassen beschreiben, bestimmte Eigenschaften besitzen.

  Wesentliche Inhalte dieses Kapitels wurden schon in [GKV03] veröffentlicht.

- In Kapitel 4 wollen wir Blattsprachen nicht für Turing-Maschinen, sondern für nichtdeterministische endliche Automaten untersuchen. Diese Blattsprachen für endliche Automaten stehen im Zusammenhang mit Phänomenen aus der mathematischen Logik:

  Da die Blattsprachen eine Verallgemeinerung der nichtdeterministischen TM sind, liegt es nahe, auch die nichtdeterministischen endlichen Automaten mithilfe von Blattsprachen zu verallgemeinern. Es stellt sich dabei heraus, dass endliche Automaten, denen als Blattsprache wiederum eine reguläre Sprache zur Verfügung steht, genau so mächtig sind wie gewöhnliche nichtdeterministische Automaten.

Schon lange ist bekannt, dass ein Zusammenhang zwischen Logik und endlichen Automaten besteht: Monadische logische Formeln der zweiten Stufe erkennen genau die regulären Sprachen. Und so wie die Blattsprachen eine Erweiterung des Nichtdeterminismus in der Komplexitätstheorie sind, sind die *Lindströmquantoren* in der Logik eine Verallgemeinerung der Existenzquantoren.

In diesem Kapitel wollen wir zeigen, wie die Lindströmquantoren mit den Blattsprachen für endliche Automaten im Zusammenhang stehen.

Wesentliche Inhalte dieses Kapitels wurden schon in [GV01] veröffentlicht.

- Schließlich wollen wir in Kapitel 5 noch einen Blick auf Komplexitätsklassen für Funktionen werfen. Dort gibt es einen Begriff, der den Blattsprachen entspricht: die *Blattfunktionen*. In diesem Abschnitt wollen wir die Funktionenklasse FPSPACE mithilfe von Blattfunktionen beschreiben:

  Eine interessante Erkenntnis auf dem Gebiet der Blattsprachen war die Tatsache, dass es eine reguläre Blattsprache gibt, mithilfe derer die Komplexitätsklasse PSPACE dargestellt werden kann.

  Allerdings betrachtet die Komplexitätstheorie nicht nur Entscheidungsprobleme, sondern auch Funktionen. Es werden also auch Komplexitätsklassen von Funktionen untersucht. Dort wurde auch der Versuch unternommen, einen den Blattsprachen entsprechenden Begriff zu definieren: die Blattfunktionen. Die Ausgaben auf den Blättern des Berechnungsbaumes einer TM werden als Eingaben einer Funktion – der Blattfunktion – verwendet und die Ausgabe dieser Funktion ist dann die Ausgabe der TM auf ihrer Eingabe.

  In diesem Kapitel wollen wir nun, in Anlehnung an die einfache Blattsprache für PSPACE, eine einfache Blattfunktion für die Funktionenklasse FPSPACE angeben. Diese einfache Blattfunktion wird die Matrixmultiplikation sein. Im Beweis werden wir darüber hinaus noch eine ganze Menge weiterer Darstellungen von FPSPACE angeben.

  Wesentliche Inhalte dieses Kapitels wurden schon in [GV05] veröffentlicht.

# Kapitel 2

# Grundlagen

Wir verwenden als Berechnungsmodelle die Turing-Maschine (TM) und den Transducer. Diese Modelle unterscheiden sich nur in der Ausgabe: TM geben auf jede Eingabe nur ein Zeichen aus, das durch den Endzustand bestimmt ist, während Transducer ein Wort ausgeben, das durch den Inhalt des Berechnungsbandes am Ende der Berechnung bestimmt ist.

Bei der Definition von Blattsprachen geht man von Polynomialzeit-Turing-Maschinen aus. Jeder Sprache $A$ über einem Alphabet $\Delta$ wird eine Komplexitätsklasse auf folgende Weise zugeordnet:

Eine Sprache $L \subseteq \Sigma^*$ ist in der Klasse, wenn es eine nichtdeterministische Polynomialzeit-TM $M$ mit folgenden Eigenschaften gibt: Die Eingabe von $M$ sind Wörter $x$ über $\Sigma$. Auf jedem ihrer Berechnungspfade gibt $M$ einen Buchstaben aus $\Delta$ aus. Auf eine Eingabe spannt $M$ einen Berechnungsbaum auf. An den Blättern des Berechnungsbaum steht jeweils ein Zeichen aus $\Delta$. Die Zeichen an allen Blättern hintereinander gelesen ergeben das *Blattwort* von $M$ auf $x$. Dann liegt $x$ genau dann in $L$, wenn das Blattwort von $M$ auf Eingabe $x$ in $A$ liegt.

Diese Definition werden wir nun formaler beschreiben: Eine Blatt-Turing-Maschine (BTM) ist ein Tupel $M = (Z, \Sigma, \Gamma, \delta, z_0, \Box, E, \Delta, \beta)$, wobei $Z$ die endliche Zustandsmenge, $\Sigma$ das Eingabealphabet, $\Gamma \supset \Sigma$ das Arbeitsalphabet, $\delta : Z \times \Gamma \to (Z \times \Gamma \times \{L, N, R\})^*$ die Überführungsfunktion, $z_0 \in Z$ der Startzustand, $\Box \in \Gamma \setminus \Sigma$ das Leersymbol, $E \subseteq Z$ die Menge der Endzustände, $\Delta$ das Blattalphabet und $\beta : E \to \Delta$ eine Funktion ist, die jedem Endzustand $z$ ein Zeichen des Blattalphabets $\beta(z)$ zuordnet.

Auf Eingabe $x$ spannt $M$ nun einen Berechnungsbaum $T_M(x)$ auf, wobei die Anordnung der Äste des Baumes durch die Anordnung der Nachfolger bei $\delta$ bestimmt ist. Falls $M$ auf jedem Pfad schließlich in einen Endzustand übergeht, dann endet also jeder Pfad in einem Blatt. Jedem der Blätter des Berechnungsbaumes wird nun durch die Funktion $\beta$ ein Zeichen aus $\Delta$ zu-

geordnet. Die Zeichen an allen Blättern von $T_M(X)$ zusammengenommen bilden nun eine Zeichenkette, die wir $\text{leafstring}^M(x)$ nennen. Falls $M$ bei Eingabe $x$ auf einem seiner Pfade nicht hält, so ist $\text{leafstring}^M(x)$ nicht definiert.

Sei nun $A$ eine Sprache über $\Delta$. Dann ist $\text{Leaf}(A)$ eine wie folgt definierte Komplexitätsklasse: Eine Sprache $L$ ist in $\text{Leaf}(A)$, wenn es eine BTM $M$ mit Blattalphabet $\Delta$ und ein Polynom $p$ gibt, so dass $M$ auf jede Eingabe $x$ auf jedem Berechnungspfad nach höchstens $p(|x|)$ Schritten anhält und falls gilt $x \in L \Leftrightarrow \text{leafstring}^M(x) \in A$. Wir haben nun also eine Komplexitätsklasse mithilfe der *Blattsprache* $A$ beschrieben.

Wir hatten schon in der Einführung einige Komplexitätklassen genannt. und behauptet, dass sie mithilfe von Blattsprachen beschreibbar wären. Wir wollen nun Blattsprachen über dem Alphabet $\{0,1\}$ für diese Klassen angeben:

P wird durch die Blattsprache $A =_{\text{def}} \left\{ w \in \{0,1\}^* \;\middle|\; \text{es existiert ein } u \in \{0,1\}^* \text{ mit } w = u1 \right\}$ beschrieben. Dies gilt, da in $A$ nur das letzte Zeichen von Interesse ist. Übertragen auf die BTM heißt das aber, dass nur der letzte Berechnungspfad beachtet werden muss. Damit wird die BTM aber effektiv zu einer deterministischen TM.

NP wird durch die Blattsprache $A =_{\text{def}} \left\{ w \in \{0,1\}^* \;\middle|\; \text{es existiertn } u, v \in \{0,1\}^* \text{ mit } w = u1v \right\}$ beschrieben. Ein Wort ist in $A$, wenn es mindestens eine 1 enthält. Aus einer nichtdeterministischen TM, die eine Sprache $L \in \text{NP}$ akzeptiert, lässt sich also eine BTM $M'$ für die Blattsprache $A$ erzeugen, indem man jedem akzeptierenden Endzustand eine 1 und jedem ablehnenden Endzustand eine 0 zuordnet.

$\oplus$P wird durch $A =_{\text{def}} \left\{ w \in \{0,1\}^* \;\middle|\; |w|_1 \equiv 0 \pmod 2 \right\}$ beschrieben, wobei für ein Wort $w \in \Sigma^*$ und ein Zeichen $x \in \Sigma$ der Ausdruck $|w|_x$ die Anzahl der Vorkommen von $x$ in $w$ beschreibt. Eine Sprache $L$ ist in $\oplus$P, wenn eine nichtdeterministische TM $M$ existiert, für die gilt, dass $x$ genau dann in $L$ ist, wenn die Anzahl der akzeptierenden Pfade von $M$ auf Eingabe $x$ gerade ist. Man kann nun wieder eine BTM $M'$ für die Blattsprache $A$ erzeugen, indem man jedem akzeptierenden Endzustand eine 1 und jedem ablehnenden Endzustand eine 0 zuordnet.

Die Blattsprache für PSPACE lässt sich zwar nicht so einfach beschreiben, ist aber immer noch regulär (siehe [HLS+93]).

Es gibt noch Komplexitätsklassen, die mithilfe von NPTM definiert werden, die aber nicht mit dem von uns verwendeten Blattsprachenbegriff beschrieben werden können, die *Promise-Klassen*. Sie zeichnen sich dadurch aus, dass die Akzeptanzbedingung für eine Eingabe nicht das Gegenteil der Ablehnungsbedingung ist. So ist zum Beispiel die Klasse UP folgendermaßen

definiert: Eine Menge $L$ ist in UP, falls eine NPTM $M$ existiert, so dass für alle $x$ die beiden folgenden Bedingungen gelten:

1. $x \in L \Leftrightarrow M(x)$ besitzt genau einen akzeptierenden Berechnungspfad,
2. $x \notin L \Leftrightarrow M(x)$ besitzt keinen akzeptierenden Berechnungspfad.

Promise-Klassen unterscheiden sich von den bisher behandelten Klassen dadurch, dass es für jede Promise-Klasse NPTM gibt, die keine Menge aus dieser Klasse akzeptieren, da sie für irgendeine Eingabe weder die Akzeptanz- noch die Ablehnungsbedingung erfüllen. Das „Promise“ dieser Klassen bezieht sich also auf das Versprechen der NPTM, auf jede Eingabe entweder die Akzeptanz- oder die Ablehnungsbedingung zu erfüllen.

Im Blattsprachenzusammenhang kann man Promise-Klassen definieren, indem man nicht eine sondern zwei Blattsprachen definiert, die akzeptierende und die ablehnende Sprache. Der bisherige Fall ist dann einfach der Spezialfall, bei dem die beiden Sprachen Komplemente sind.

Seien also nun $A$ und $B$ Sprachen über $\Delta$ mit $A \cap B = \emptyset$. Dann ist $\mathrm{Leaf}(A, B)$ eine wie folgt definierte Komplexitätsklasse: Eine Sprache $L$ ist in $\mathrm{Leaf}(A, B)$, wenn es eine BTM $M$ mit Blattalphabet $\Delta$ und ein Polynom $p$ gibt, so dass $M$ auf jede Eingabe $x$ auf jedem Berechnungspfad nach höchstens $p(|x|)$ Schritten anhält und folgende zwei Bedingungen gelten:

1. $x \in L \Leftrightarrow \mathrm{leafstring}^M(x) \in A$,
2. $x \notin L \Leftrightarrow \mathrm{leafstring}^M(x) \in B$.

Wir können jetzt auch für UP eine Blattsprachendefinition geben: $\mathrm{UP} = \mathrm{Leaf}(A, B)$ mit $A =_{\mathrm{def}} \left\{ x \in \{0,1\}^* \;\middle|\; |x|_1 = 1 \right\}$ und $B =_{\mathrm{def}} \{0\}^*$.

Es existiert noch eine andere Art von Promise-Klassen, bei denen nicht die Akzeptanz- und Ablehnungsbedingungen eingeschränkt sind, sondern die Form des Berechnungsbaums der NPTM: Wir werden uns mit den *balancierten* TM beschäftigen. Dies sind NPTM, deren Berechnungsbäume binäre Bäume sind, bei denen alle Knoten die folgenden Eigenschaften besitzen:

1. Der linke Teilbaum ist ein vollständiger Binärbaum.
2. Die Höhe des linken Teilbaums ist mindestens so groß wie die Höhe des rechten Teilbaums.

Solche TM zeichnen sich dadurch aus, dass sowohl die Anzahl ihrer Pfade auf eine bestimmte Eingabe in Polynomialzeit berechenbar ist als auch das Ergebnis ihrer Berechnung für einen Pfad mit gegebener Nummer.

Wir bezeichnen als $\mathrm{BLeaf}(A)$ die Klasse aller Mengen, die durch balancierte NPTM mit Blattsprache $A$ akzeptiert werden können. Die Klasse $\mathrm{BLeaf}(A, B)$ ist analog definiert.

# Kapitel 3

# Blattsprachen und Generische Orakel

## 3.1 Einführung

Ein großes Problem der Komplexitätstheorie war schon immer der Versuch, Komplexitätsklassen zu *trennen*, also von zwei Komplexitätsklassen zu zeigen, dass sie tatsächlich verschieden sind. Das wohl berühmteste Beispiel sind die Klassen P und NP: Wir wissen nur, dass $\mathrm{P} \subseteq \mathrm{NP}$ gilt, aber nicht, ob P eine echte Teilmenge von NP ist.

Betrachtet man allerdings *Orakelklassen*, so waren Trennungen im Allgemeinen leichter zu erzielen: Schon 1975 fanden Baker, Gill und Solovay ein Orakel $B$ mit der Eigenschaft $\mathrm{P}^B \neq \mathrm{NP}^B$ [BGS75].

Nun gibt es andererseits ein Orakel $A$ mit der Eigenschaft $\mathrm{P}^A = \mathrm{NP}^A$. (Jede PSPACE-vollständige Menge ist ein solches Orakel.) Wir wissen natürlich nicht, welche dieser beiden Aussagen dem nicht-relativierten Fall entspricht.

Man hat sich nun die Frage gestellt, ob es Orakel gibt, die sozusagen den nicht-relativierten Fall „simulieren“, d.h., bei denen die Trennungsergebnisse dem nicht-relativierten Fall entsprechen. Eine Idee in diese Richtung waren die *zufälligen* Orakel, also Mengen, bei denen die Mitgliedschaft durch einen Münzwurf entschieden wird. Bennett und Gill [BG81] stellten die *Zufallsorakelhypothese* auf, die besagt, dass jede Trennung, die bezüglich eines zufälligen Orakels erzielt wird, auch im unrelativierten Fall gilt. Leider stellte sich im Jahr 1994 heraus, dass diese Hypothese falsch ist [CCG$^+$94].

Ein weiterer Versuch trennende Orakel zu finden waren die *generischen Orakel*. Sie wurden von Blum und Impagliazzo [BI87] in die Komplexitätstheorie eingeführt. Die generischen Orakel beruhen auf dem Prinzip der *Stufenkonstruktion*. Generische Orakel besitzen alle Eigenschaften, die durch eine

Stufenkonstruktion erzwungen werden können. Diese Stufenkonstruktion war in der Berechenbarkeitstheorie (siehe z.B. [Soa87]) eine Methode, um Orakel mit bestimmten gewünschten Eigenschaften zu erhalten. In ihrem Beweis für die Orakeltrennung von P und NP benutzen Baker, Gill und Solovay ebenfalls diese Methode. Deshalb trennt ein generisches Orakel auch P und NP, und man kann sogar zeigen, dass die gesamte Polynomialzeithierarchie durch generische Orakel zu einer echten Hierarchie wird.

Leider wurde gezeigt, dass die *generische Orakelhypothese* ebenfalls falsch ist [Fos93], dass es also Trennungen von Komplexitätsklasssen bezüglich generischer Orakel gibt, die im unrelativierten Fall nicht gelten. Dennoch ist es sinnvoll, die generischen Orakel zu studieren, denn sie könnten helfen, die Frage „P = NP“ zu beantworten. Denn Blum und Impagliazzo zeigten, dass P $\neq$ NP gilt, falls P $\neq$ UP bezüglich eines generischen Orakels gilt.

Es gibt nun Ergebnisse, die Orakeltrennungen und Blattsprachen verbinden. In ihren Arbeiten, die das Gebiet der Blattsprachen begründeten, haben Bovet, Crescenzi und Silvestri [BCS92] und parallel dazu Vereshchagin [Ver93] gezeigt, dass die Trennung von Blattsprachenklassen bezüglich eines Orakels durch das Betrachten der Blattprachen entschieden werden kann. In diesem Kapitel werden wir zeigen, wie dieses Ergebnis auf generische Orakel umgesetzt werden kann.

## 3.2 Grundlagen

### 3.2.1 Orakel-Turing-Maschinen

Wir betrachten Turing-Maschinen, die auf eine Informationsquelle neben ihrem Programm und ihrer Eingabe zurückgreifen können, ein *Orakel*.

Ein Orakel $O$ ist eine totale Funktion $O : \mathbb{N} \to \{0, 1\}$. (Wenn man diese Funktion als charakteristische Funktion ansieht, kann man ein Orakel auch als eine Menge natürlicher Zahlen ansehen.) Ein *endliches Orakel* $v$ ist eine partielle Funktion $v : \mathbb{N} \to \{0, 1\}$ mit endlichem Definitionsbereich $D(v)$. Wir werden endliche Orakel, deren Definitionsbereich ein Anfangsbereich der natürlichen Zahlen, also eine Menge $\{0, 1, \ldots, n\}$, ist, auch mit binären Zeichenketten $v(0)v(1)\ldots v(n)$ gleichsetzen. Ebenso werden wir Orakel $O$, die ja den Definitionsbereich $\mathbb{N}$ besitzen, mit unendlichen binären Zeichenketten gleichsetzen.

Eine Orakel-Turing-Maschine $M$ ist nun eine Turing-Maschine mit einem zusätzlichen Band $B$ (dem *Orakelband*) und vier speziellen Zuständen $z_q$, $z_+$, $z_-$ und $z_?$. Sei $v$ nun ein Orakel oder ein endliches Orakel. Falls $M$ nun auf $v$

zurückgreift, so schreiben wir dafür $M^v$. Dabei kann $M$ auf folgende Weise auf $v$ zurückgreifen: Die TM kann während ihrer Berechnung die Binärdarstellung einer Zahl auf das Orakelband schreiben. Wenn auf dem Orakelband nun die Binärdarstellung der Zahl $n$ steht und die TM in den Zustand $z_q$ geht, geschieht im nächsten Schritt folgendes: Das Orakelband wird gelöscht und die TM geht in einen der Zustände $z_+$, $z_-$ oder $z_?$ über. Falls $n$ nicht im Definitionsbereich von $v$ liegt, geht $M^v$ in den Zustand $z_?$ über; falls $v(n) = 1$ gilt, geht $M^v$ in den Zustand $z_1$ über, und falls $v(n) = 0$ gilt, so geht $Mv$ in den Zustand $z_0$ über.

Nichtdeterministische Polynomialzeit-Orakel-Turing-Maschinen bezeichnen wir als $NPOTM$.

Orakel-Transducer sind analog definiert.

### 3.2.2 Generizität

Wir verwenden den Begriff der *Generizität* von Orakeln so, wie er von Blum und Impagliazzo in [BI87] eingeführt wurde:

Ein endliches Orakel $w$ *erweitert* ein endliches Orakel $v$, in Zeichen $v \sqsubseteq w$, falls $D(v) \subseteq D(w)$ gilt und beide Orakel auf $D(v)$ übereinstimmen. Ein Orakel $O$ *erweitert* ein endliches Orakel, falls $O$ und $v$ auf $D(v)$ übereinstimmen. In diesem Fall nennen wir $v$ auch ein *Präfix* von $O$.

Eine Menge $D$ von endlichen Orakeln ist *dicht*, falls für jedes endliche Orakel $v$ ein $w \in D$ existiert, das $v$ erweitert.

Sei $\mathcal{C} = \{D_1, D_2, \ldots\}$ eine aufzählbare Menge dichter Mengen von endlichen Orakeln. Wir nennen ein Orakel $O$ $\mathcal{C}$-generisch, falls $O$ für jedes $i \in \mathbb{N}$ ein endliches Präfix $w_i \in D_i$ besitzt. Eine Menge von endlichen Orakeln ist *arithmetisch*, falls sich Zugehörigkeit zu ihr als endliche First-Order-Formel über rekursiven Prädikaten ausdrücken lässt. Sei $\mathcal{A}$ nun die Menge aller dichten, arithmetischen Mengen von endlichen Orakeln. Ein Orakel heißt dann *generisch*, wenn es $\mathcal{A}$-generisch ist.

### 3.2.3 Blattsprachenreduktionen

In den Arbeiten von Bovet, Crescenzi und Silvestri [BCS92] und Vereshchagin [Ver93] wurde eine spezielle Reduktion zwischen Sprachen vorgestellt, die in diesen Arbeiten auf Blattsprachen angewandt wurde, die *Polylogzeit-Reduktion*.

Wir nennen eine Funktion $f\colon \Sigma^* \to \Sigma^*$ *Polylogzeit-Bit-berechenbar*, falls zwei Polynomialzeit-Orakel-Transducer $R\colon \mathbb{N}^2 \to \Sigma$ und $l\colon \mathbb{N} \to \mathbb{N}$ existieren,

so dass für alle $x \in \Sigma^*$ gilt:

$$f(x) = R^x(|x|, 1)R^x(|x|, 2) \cdots R^x(|x|, l^x(|x|)).$$

D.h. also, die einzelnen Bits der Ausgabe von $f(x)$ lassen sich in polylogarithmischer Zeit berechnen.

Für eine Funktion $f : M \to N$ und eine Menge $A \subseteq M$ definieren wir $f(A) =_{\text{def}} \{ f(x) \mid x \in A \}$.

Seien nun $(A, R)$ und $(A', R')$ zwei Paare von Sprachen. $(A, R)$ ist *Polylogzeit-Bit-reduzierbar* auf $(A', R')$ (kurz: $(A, R) \leq_{\text{m}}^{\text{plt}} (A', R')$ ), falls eine Polylogzeit-Bit-berechenbare Funktion $f$ existiert, für die sowohl $f(A) \subseteq A'$ als auch $f(R) \subseteq R'$ gilt. (Für den Fall $R = \overline{A}$ und $R' = \overline{A'}$ heißt dies $x \in A \Leftrightarrow f(x) \in A'$.)

Nun wurde in [BCS92] und [Ver93] ein Zusammenhang zwischen Polylogzeit-Bit-Reduktionen zwischen Blattsprachen und der Trennung von Orakelklassen, die mithilfe dieser Blattsprachen definiert sind, hergestellt. Dieses Resultat lautet:

**Satz 3.2.1 [BCS92, Ver93].** *Seien $(A, R)$ und $(A', R')$ Paare von Blattsprachen. Dann gilt*

$\text{BLeaf}^O(A, R) \subseteq \text{BLeaf}^O(A', R')$ *für alle Orakel $O$* $\iff$ $(A, R) \leq_{\text{m}}^{\text{plt}} (A', R')$.

## 3.3 Generische Orakel und PLT-Reduktionen

Bevor wir zum zentralen Ergebnis dieses Kapitels kommen können, müssen wir noch mehrere technische Lemmata zeigen.

**Lemma 3.3.1.** *Sei $M$ eine balancierte NPOTM mit Eingabealphabet $\{0, 1\}$ und Orakelalphabet $\Sigma$. Für $x \in \Sigma^*$ ist die Funktion $f(x) =_{\text{def}} \text{leafstring}_{M^x}(|x|)$ Polylogzeit-Bit-berechenbar.*

*Beweis.* Der Transducer $l$ aus der Definition von $\leq_{\text{m}}^{\text{plt}}$ muss in unserem Fall die Anzahl der Pfade des balancierten Berechnungsbaums von $M$ auf Eingabe $|x|$ und Orakel $x$ berechnen. Da $l$ selbst die Eingabe $|x|$ und das Orakel $x$ zur Verfügung hat, kann einfach $M$ simuliert werden.

Der Transducer $R$ muss dann auf Eingabe $(|x|, k)$ die Ausgabe von $M$ auf Berechnungspfad $k$ bei Eingabe $|x|$ und Orakel $x$ berechnen. Da auch $R$ sowohl $|x|$ als auch $x$ direkt benutzen kann, kann der Berechnungspfad $k$ von $M$ direkt simuliert werden. Da der Berechnungsbaum von $M$ balanciert ist, ist es auch einfach, den Berechnungspfad $k$ zu finden. □

**Definition 3.3.2.** Wir definieren die Funktion $\delta : \mathbb{N} \to \mathbb{N}$ als $\delta(n) =_{\text{def}} \frac{n(n-1)}{2}$.

Diese Funktion besitzt die Eigenschaft $\delta(n+1) = \delta(n) + n$.

**Definition 3.3.3.** Für jede Sprache $A$ und jedes Orakel $O$ definieren wir eine „Testsprache“ $L_A(O) =_{\text{def}} \left\{ x \;\middle|\; O(\delta(x)+1) \dots O(\delta(x)+x) \in A \right\}$.

**Lemma 3.3.4.** *Seien $A$ und $B$ arithmetische Mengen und sei* $\text{BLeaf}^G(A) \subseteq \text{BLeaf}^G(B)$ *für alle generischen Orakel $G$. Dann existiert eine balancierte NPOTM $\hat{M}$, so dass für alle generischen Orakel $G$ und für alle $x$ gilt $x \in L_A(G) \Leftrightarrow \text{leafstring}_{\hat{M}^G}(x) \in B$.*

*Beweis.* Zuerst eine Beobachtung:

Für alle Orakel $O$ gilt aufgrund der Definition von $L_A(O)$, dass $L_A(O) \in \text{BLeaf}^O(A)$. Für alle generischen Orakel $G$ gilt dann $L_A(G) \in \text{BLeaf}^G(B)$, da wir $\text{BLeaf}^G(A) \subseteq \text{BLeaf}^G(B)$ vorausgesetzt haben. Sei $M_k$ die $k$-te Maschine in einer Aufzählung aller *balancierten* NPOTM. Dann gibt es also für jedes generische Orakel $G$ ein $k$, so dass für alle $x$ gilt $x \in L_A(G) \Leftrightarrow \text{leafstring}_{M_k^G}(x) \in B$.

Nun können wir das Lemma beweisen. Wir führen dazu einen Widerspruchsbeweis durch. Die folgende Annahme ist das Gegenteil der Aussage des Lemmas:

**Annahme Y:** Für alle $k$ existieren ein $x$ und ein generisches Orakel $G$, so dass

$$x \in L_A(G) \Leftrightarrow \text{leafstring}_{M_k^G}(x) \notin B.$$

Für den Widerspruch zeigen wir, dass unter Annahme Y die Familie $\mathcal{C} = \{C_0, C_1, \dots, C_{2m}, C_{2m+1}, \dots\}$ mit

- $C_{2m} =_{\text{def}} \left\{ z \;\middle|\; \text{es existiert ein } x \text{ mit } x \in L_A(z) \Leftrightarrow \text{leafstring}_{M_m^z}(x) \notin B \right\}$
- $C_{2m+1} =_{\text{def}} D_m$ (die $m$-te dichte arithmetische Menge)

eine Familie dichter Mengen ist. Dann gilt für jedes $\mathcal{C}$-generische Orakel $G$, dass

- $G$ generisch ist (da $A$ und $B$ arithmetisch sind, sind also auch die Mengen $C_{2m}$ arithmetisch),
- für alle $m$ ein $x$ existiert mit $x \in L_A(G) \Leftrightarrow \text{leafstring}_{M_m^G}(x) \notin B$.

Dies steht im Widerspruch zu der zu Beginn dieses Beweises gemachten Beobachtung, dass für alle generischen Orakel $G$ ein $k$ existiert, so dass für alle $x$ gilt $x \in L_A(G) \Leftrightarrow \text{leafstring}_{M_k^G}(x) \in B$. Also muss Annahme Y falsch sein.

Nun müssen wir aber noch zeigen, dass unter Annahme Y die Mengen $C_{2m}$ dicht sind.

Wir zeigen zuerst, dass folgendes gilt:

**Behauptung Z:** Falls Annahme Y gilt, dann gibt es für jedes $m$ und für jedes $\hat{s}$ mit $|\hat{s}| \in \delta(\mathbb{N})$ ein Orakel $H \in \hat{s} \cdot \{0,1\}^\omega$ und ein $x$ mit $x \in L_A(H) \Leftrightarrow \text{leafstring}_{M_m^H}(x) \notin B$.

Dies geschieht durch einen Widerspruchsbeweis. Wir nehmen an, dass Y gilt, es aber ein $m$ gibt, so dass für alle Orakel $H \in \hat{s} \cdot \{0,1\}^\omega$ und für alle $x$ gilt $x \in L_A(H) \Leftrightarrow \text{leafstring}_{M_m^H}(x) \in B$. Dann definieren wir eine NPOTM $M'$ folgendermaßen: Sei $G$ ein generisches Orakel und $x \in \mathbb{N}$. Dann setzen wir

$$\text{leafstring}_{M'^O}(x) = \begin{cases} \text{leafstring}_{M_m^{\hat{G}}}(x), & \text{falls } |\hat{s}| \leq \delta(x), \\ \text{ein Wort aus } B, & \text{falls } |\hat{s}| > \delta(x) \text{ und} \\ & G(\delta(x)+1)\dots G(\delta(x)+x) \in A \\ \text{ein Wort aus } \overline{B}, & \text{falls } |\hat{s}| > \delta(x) \text{ und} \\ & G(\delta(x)+1)\dots G(\delta(x)+x) \notin A \end{cases}$$

wobei $\hat{G} =_{\text{def}} \hat{s} \cdot G(\delta(r)+1)G(\delta(r)+2)\cdots$. Dann ist $M'$ eine NPOTM und für jedes generische Orakel $G$ gilt $x \in L_A(G) \Leftrightarrow \text{leafstring}_{M'^G}(x) \in B$, was im Widerspruch zu Annahme Y steht. Damit haben wir Behauptung Z bewiesen.

Nun können wir zeigen, dass unter Annahme Y die Mengen $C_{2m}$ dicht sind, d.h., dass für alle endlichen Orakel $s$ ein $z \in C_{2m}$ mit $s \sqsubseteq z$ existiert.

Wir wählen ein $\hat{s}$ so, dass $s \sqsubseteq \hat{s}$ und $|\hat{s}| = \delta(r)$ für ein geeignetes $r$ gelten. Sei $m$ beliebig. Wähle $x$ als lexikographisch kleinstes Wort, für das ein Orakel $H$ wie in Behauptung Z beschrieben existiert. Sei $q_x$ die lexikographisch größte Orakelfrage, die während der Berechnung von $M_m^H(x)$ an $H$ gestellt wird. Wir definieren $z =_{\text{def}} H(0)H(1)\dots H(q_x)$. Dies ergibt $\hat{s} \sqsubseteq z$ und $z \in C_{2m}$. Also ist $C_{2m}$ dicht. □

Das Hauptresultat dieses Kapitels setzt, wie schon Satz 3.2.1, die Trennung von Blattsprachenklassen bezüglich Orakeln mit der $\leq_{\text{m}}^{\text{plt}}$-Reduzierbarkeit der Blattsprachen selbst in Beziehung; allerdings geht es im Folgenden um generische Orakel:

**Satz 3.3.5.** *Seien $A$ und $B$ arithmetische Mengen. Dann gilt*

$$\text{BLeaf}^G(A) \subseteq \text{BLeaf}^G(B) \text{ für alle generischen Orakel } G \iff A \leq_{\text{m}}^{\text{plt}} B.$$

*Beweis.* Die Folgerung von rechts nach links folgt mit $R = \overline{A}$, $A' = B$ und $R' = \overline{B}$ direkt aus Satz 3.2.1. Wir müssen also nur noch die Richtung von links nach rechts zeigen.

Seien $A$ und $B$ arithmetische Mengen und sei $\mathrm{BLeaf}^G(A) \subseteq \mathrm{BLeaf}^G(B)$ für alle generischen Orakel. Wir müssen nun eine $\leq_{\mathrm{m}}^{\mathrm{plt}}$-Reduktion von $A$ nach $B$ angeben.

Aufgrund von Lemma 3.3.4 wissen wir, dass eine balancierte NPOTM $\hat{M}$ existiert, so dass für alle generischen Orakel $G$ und für alle $x$ gilt $x \in L_A(G) \Leftrightarrow \mathrm{leafstring}_{\hat{M}^G}(x) \in B$.

Die Maschine $\hat{M}$ können wir nun benutzen, um eine $\leq_{\mathrm{m}}^{\mathrm{plt}}$-Reduktion von $A$ nach $B$ anzugeben:

**Eingabe:** Ein Wort $w$, $|w| = x$, $w = w(0)w(1)\dots w(x-1)$.

**Ausgabe:** Ein Wort $v$ mit $w \in A \Leftrightarrow v \in B$.

**Algorithmus:** Sei $s =_{\mathrm{def}} 0^{\delta(x)}w0^m$ der Präfix eines generischen Orakels $G$, wobei $m$ so groß ist, dass bei einer Simulation von $\hat{M}^G(x)$ alle Fragen an $G$ im Bereich des Anfangsstücks $s$ liegen. Dann gilt $x \in L_A(G) \Leftrightarrow w \in A$.

Nun können wir eine NPOTM $M'$ konstruieren, die auf Eingabe $x$ mit Orakel $w$ das Verhalten von $\hat{M}^G(x)$ simuliert. (Dies ist möglich da wir $G$ gerade so konstruiert haben, dass das Wissen über $w$ ausreicht, um $G$ zu simulieren.) $M'$ ist also balanciert und es gilt $\mathrm{leafstring}_{\hat{M}^G}(x) = \mathrm{leafstring}_{M'^w}(x)$. Aufgrund von Lemma 3.3.1 wissen wir aber, dass $v =_{\mathrm{def}} \mathrm{leafstring}_{M'^w}(x)$ Polylogzeit-Bit-berechenbar ist. Außerdem wissen wir wegen Lemma 3.3.4, dass $x \in L_A(G) \Leftrightarrow v \in B$.

Insgesamt erhalten wir also $w \in A \Leftrightarrow v \in B$.

□

## 3.4 Folgerungen

### 3.4.1 Universelle Orakel

Das folgende Resultat ist aus der Berechenbarkeitstheorie bekannt, siehe z.B. [BI87]:

**Proposition 3.4.1.** *Sei* $\Pi$ *irgendeine Eigenschaft von Orakeln, die durch eine First-Order-Formel mit berechenbaren Prädikaten und einem Prädikat* $O(x)$ *(das das Enthaltensein im Orakel ausdrückt) und die nicht von einer*

*endlichen Menge von Bits des Orakels abhängt. Dann haben entweder alle generischen Orakel diese Eigenschaft oder keines.*

Daraus ergibt sich folgende Folgerung aus dem Satz 3.3.5:

**Korollar 3.4.2.** *Seien $A$ und $B$ arithmetische Mengen. Dann gilt*

$$\mathrm{BLeaf}^G(A) \not\subseteq \mathrm{BLeaf}^G(B) \text{ für alle generischen Orakel } G \iff A \not\leq_{\mathrm{m}}^{\mathrm{plt}} B.$$

Daraus folgern wir also, dass jede Orakeltrennung zwischen Klassen, die mithilfe von arithmetischen Blattsprachen definiert werden, relativ zu jedem festen generischen Orakel gilt.

**Korollar 3.4.3.** *Seien $A$ und $B$ arithmetische Mengen, und sei $\widehat{G}$ ein festes generisches Orakel. Dann sind die folgenden drei Aussagen äquivalent:*

*(1) Es existiert ein Orakel $O$ mit* $\mathrm{BLeaf}^O(A) \not\subseteq \mathrm{BLeaf}^O(B)$.

*(2)* $\mathrm{BLeaf}^{\widehat{G}}(A) \not\subseteq \mathrm{BLeaf}^{\widehat{G}}(B)$.

*(3)* $\mathrm{BLeaf}^G(A) \not\subseteq \mathrm{BLeaf}^G(B)$ *für alle generischen Orakel $G$.*

*Beweis.* Es existiere ein Orakel $O$ mit $\mathrm{BLeaf}^O(A) \not\subseteq \mathrm{BLeaf}^O(B)$. Wegen Satz 3.2.1 folgt daraus, dass $A \not\leq_{\mathrm{m}}^{\mathrm{plt}} B$. Aufgrund von Korollar 3.4.2 folgt daraus, dass $\mathrm{BLeaf}^G(A) \not\subseteq \mathrm{BLeaf}^G(B)$ für alle generischen Orakel $G$. Daraus folgt trivialerweise, dass $\mathrm{BLeaf}^{\widehat{G}}(A) \not\subseteq \mathrm{BLeaf}^{\widehat{G}}(B)$ und daraus folgt wiederum trivialerweise, dass $\mathrm{BLeaf}^O(A) \not\subseteq \mathrm{BLeaf}^O(B)$ für ein Orakel $O$ ist. □

Das vorhergehende Resultat bedeutet insbesondere, dass alle relativierten Trennungen, die zwischen Klassen $\mathrm{BLeaf}(A)$ (wobei $A$ arithmetisch ist) gelten, „gleichzeitig“ gelten:

**Korollar 3.4.4.** *Alle Orakeltrennungen zwischen Klassen, die eine relativierbar vollständige Menge besitzen, gelten gleichzeitig bezüglich desselben festen generischen Orakels $\widehat{G}$.*

*Beweis.* Die Tatsache, dass eine Klasse $\mathcal{C}$ als $\mathrm{BLeaf}(A, R)$ mit $R = \overline{A}$ charakterisiert werden kann (d.h. $\mathcal{C} = \mathrm{BLeaf}(A)$), ist äquivalent mit der Aussage, dass $\mathcal{C}$ eine vollständige Menge in allen relativierten Welten besitzt [BCS92, Satz 4.2]. Wenn nun aber eine Klasse $\mathcal{C}$ eine arithmetische relativierbar vollständige Menge besitzt, ist sie natürlich auch durch eine arithmetische Blattsprache charakterisierbar. Deshalb ist jede Klasse, die relativierbar eine arithmetische vollständige Menge besitzt, in der Form $\mathrm{BLeaf}(A)$ mit arithmetischem $A$ darstellbar. Das Resultat folgt nun unmittelbar aus Korollar 3.4.3. □

Da $\leq_{\mathrm{m}}^{\mathrm{plt}}$ reflexiv und transitiv ist, können wir einen *plt-Grad* als die Klasse aller Mengen definieren, die äquivalent bezüglich $\leq_{\mathrm{m}}^{\mathrm{plt}}$ sind. Dann können wir das obige Ergebnis so auffassen, dass die Ordnung der plt-Grade isomorph zu der Ordnung der relativierten Komplexitätsklassen ist. Dies drücken wir in folgendem Korollar aus, wobei *arithmetische plt-Grade* plt-Grade sind, die einen arithmetischen Repräsentanten besitzen.

**Korollar 3.4.5.** *Sei $\hat{G}$ irgendein festes generisches Orakel. Dann ist die Familie*

$$\left\{ \mathrm{BLeaf}^{\hat{G}}(A) \;\middle|\; A \text{ ist arithmetisch} \right\},$$

*partiell geordnet bezüglich Teilmengenbeziehungen, isomorph zu der partiellen Ordnung aller arithmetischen plt-Grade.*

Blum und Impagliazzo haben gezeigt, dass zwei ressourcenbeschränkte Klassen, deren Beschränkungen „polynomiell unscharf“ sind, identisch sind, wenn sie relativ zu einem generischen Orakel zusammenfallen [BI87]. Genauer gesagt haben Blum und Impagliazzo folgendes gezeigt:

**Proposition 3.4.6.** *Sei $f$ eine berechenbare Funktion, $L$ eine entscheidbare Menge. Falls $L$ durch eine Orakel-TM in Zeit $f(|x|)$ (bzw. Platz $f(|x|)$) berechenbar ist, dann ist sie auch ohne Orakel in Zeit $O(f(|x|))$ (bzw. Platz $O(f(|x|))$) berechenbar.*

Wir können Korollar 3.4.2 umformen und damit ihr Resultat auf alle Blattsprachen-beschreibbaren Klassen ausdehnen.

**Korollar 3.4.7.** *Seien $A$ und $B$ arithmetische Mengen. Falls ein generisches Orakel $G$ mit* $\mathrm{BLeaf}^{G}(A) \subseteq \mathrm{BLeaf}^{G}(B)$ *existiert, dann gilt* $\mathrm{BLeaf}(A) \subseteq \mathrm{BLeaf}(B)$.

### 3.4.2 Ressourcenbeschränkte Generizität

Beinahe alle interessanten syntaktischen Komplexitätsklassen zwischen P und PSPACE können durch Blattsprachen ausgedrückt werden (dabei ist PP eine prominente Ausnahme, siehe [Vol99b]). Wenn man den Beweis des Satzes 3.3.5 betrachtet, erkennt man, dass die relevanten Klassen dichter Mengen von endlichen Orakeln (d.h. die Mengen $C_{2m}$ aus dem Beweis) in $\mathrm{QP} =_{\mathrm{def}} \mathrm{DTIME}\big(n^{\log^{O(1)} n}\big)$, also in quasipolynomieller Zeit, entschieden werden können. Deshalb ist das entstehende Orakel nicht nur $\mathcal{A}$-generisch (wobei $\mathcal{A}$ wieder die Menge aller dichten, arithmetischen Mengen von endlichen Orakeln ist, wie in der Definition der generischen Orakel erwähnt), sondern sogar QP-generisch. Damit ist das folgende Korollar gezeigt:

**Korollar 3.4.8.** *Sei $A$ eine reguläre Menge. Dann gilt*

$$\mathrm{BLeaf}^O(A) \not\subseteq \mathrm{BLeaf}^O(B) \text{ für ein Orakel } O$$
$$\iff \mathrm{BLeaf}^G(A) \not\subseteq \mathrm{BLeaf}(B)^G \text{ für ein QP-generisches Orakel } G.$$

Darüber hinaus genügt, um eine Komplexitätsklasse von P zu trennen, ein P-generisches Orakel.

**Korollar 3.4.9.** *Sei $A$ eine reguläre Menge. Dann gilt*

$$\mathrm{BLeaf}^O(A) \not\subseteq \mathrm{P}^O \text{ für ein Orakel } O$$
$$\iff \mathrm{BLeaf}^G(A) \not\subseteq \mathrm{P}^G \text{ für ein P-generisches Orakel } G.$$

### 3.4.3 Typ-2-Komplexität

In [CIY97] zeigten Cook, Impagliazzo, und Yamakami, dass zwei beliebige Komplexitätsklassen, die gewisse allgemeine Bedingungen erfüllen, genau dann relativ zu einem generischen Orakel verschieden sind, wenn ihre korrespondierenden Typ-2-Klassen verschieden sind. Wir werden sehen, dass dies eine Verbindung zwischen Blattsprachentheorie und Typ-2-Komplexitätstheorie erzeugt.

Zum Begriff *Typ-2-Komplexität*: Als *Typ-0-Objekte* bezeichnen wir Zahlen oder Zeichenketten. *Typ-1-Objekte* sind Funktionen auf Typ-0-Objekten – und damit, als Spezialfälle, auch Mengen und Relationen. Somit sind die Komplexitätsklassen, die wir bis jetzt behandelt haben, Typ-1-Komplexitätsklassen. Typ-2-Objekte wiederum sind Funktionen auf Typ-1- und Typ-0-Objekten.

Eine $k$-stellige *Typ-2-Funktion* $F$ über $\Sigma^*$ ordnet allen $k$-tupeln $\overrightarrow{x}$ von Wörtern über $\Sigma^*$ und Mengen $X \subseteq \Sigma^*$ einen Wert in $\Sigma^*$ zu. Eine *Typ-2-Relation* ist eine 0-1-wertige Typ-2-Funktion. Sei $R$ eine Typ-2-Relation und sei $\mathfrak{C}$ eine Klasse von Typ-2-Relationen. Für eine feste Menge $X$ definieren wir $R[X] =_{\mathrm{def}} \{ \overrightarrow{x} \mid R(\overrightarrow{x}, X) \}$ und $\mathfrak{C}[X] =_{\mathrm{def}} \{ R[X] \mid R \in \mathfrak{C} \}$.

Wir nennen eine Typ-2-Funktion $F$ Polynomialzeit-berechenbar, falls ein deterministischer Orakel-Transducer $M$ und ein Polynom $p$ existieren, so dass $M$ mit Eingabe $\overrightarrow{x}$ auf dem Eingabeband und Orakel $X$ den Wert $F(\overrightarrow{x}, X)$ nach höchstens $p(|\overrightarrow{x}|)$ Schritten ausgibt, wobei jede Orakelfrage als ein Schritt zählt.

Eine einstellige Typ-2-Relation $R$ ist genau dann auf eine einstellige Typ-2-Relation $S$ Polynomialzeit-Many-One-reduzierbar (kurz: $R \leq_m^{\mathrm{p}} S$), wenn eine Polynomialzeit-berechenbare einstellige Typ-2-Funktion $F$ und eine Polynomialzeit-berechenbare zweistellige Typ-2-Relation $Q$ existieren, so dass

für alle Wörter $x$ und Mengen $X$ gilt $R(x,X) = S(F(x,X), Q[x,X])$, wobei $Q[x,X] =_{\text{def}} \{z \mid Q((x,z),X)\}$.

Eine von Cook, Impagliazzo und Yamakami untersuchte Frage war, welche Inklusionsbeziehung zwischen Typ-2-Klassen gelten. Für unter $\leq^{\text{p}}_m$ abgeschlossene Klassen wurde dies zu relativierter Inklusion bezüglich eines generischen Orakels in Beziehung gesetzt [CIY97, Theorem 3.2]:

**Proposition 3.4.10.** *Seien $\mathfrak{C}$ und $\mathfrak{D}$ Klassen berechenbarer Typ-2-Relationen. Wenn wir nun annehmen, dass $\mathfrak{C}$ und $\mathfrak{D}$ unter $\leq^{\text{p}}_m$ abgeschlossen sind, dann gilt für jedes generische Orakel:*

$$\mathfrak{C} \subseteq \mathfrak{D} \iff \mathfrak{C}[G] \subseteq \mathfrak{D}[G].$$

Im Rahmen von Blattsprachen können Typ-2-Komplexitätsklassen wie folgt definiert werden: Gegeben seien zwei Mengen $A, R \subseteq \Sigma^*$ mit $A \cap R = \emptyset$. Dann gehört die Typ-2-Relation $S$ zur Typ-2-Klasse $\mathbf{BLeaf}(A,R)$ genau dann, wenn eine NPOTM $M$ existiert, so dass für alle Worte $x$ und Mengen $X$ gilt:

$$S(x,X) \Longrightarrow \text{leafstring}_{M^X}(x) \in A$$

und

$$\neg S(x,X) \Longrightarrow \text{leafstring}_{M^X}(x) \in R.$$

Da leicht zu sehen ist, dass jede Klasse $\mathbf{BLeaf}(A,R)$ unter $\leq^{\text{p}}_m$ abgeschlossen ist, erhalten wir mithilfe von Satz 3.3.5 und Proposition 3.4.10 eine vollständige kombinatorische Charakterisierung der Inklusionsbeziehungen zwischen sinnvoll beschreibbaren syntaktischen Typ-2-Klassen.

**Korollar 3.4.11.** *Seien $A$ und $B$ arithmetische Mengen. Dann sind die folgenden Aussagen äquivalent.*

*(1)* $A \leq^{\text{plt}}_{\text{m}} B$.

*(2)* $\text{BLeaf}^G(A) \subseteq \text{BLeaf}^G(B)$ *für ein generisches Orakel $G$.*

*(3)* $\mathbf{BLeaf}(A)[G] \subseteq \mathbf{BLeaf}(B)[G]$ *für ein generisches Orakel $G$.*

*(4)* $\mathbf{BLeaf}(A) \subseteq \mathbf{BLeaf}(B)$.

*Beweis.* Die Äquivalenz von (1) und (2) ist einfach Satz 3.3.5 und die Äquivalenz von (3) und (4) ist nur Proposition 3.4.10. Die Äquivalenz von (2) und (3) folgt aus Satz 3.4.13, den wir weiter unten angeben werden. □

Typischerweise besitzt eine Typ-1-Komplexitätsklasse $\mathcal{C}$ eine natürliche Typ-2-Entsprechung, die mithilfe der gleichen Ressourcen definiert wird, die zur Beschreibung von $\mathcal{C}$ benutzt wurden. Cook, Impagliazzo und Yamakami [CIY97] untersuchten die Frage, für welche Klassen $\mathcal{C}$ und $\boldsymbol{\mathcal{C}}$ ihre relativierten Versionen übereinstimmten, d.h. $\mathcal{C}^O = \boldsymbol{\mathcal{C}}[O]$ für alle Orakel $O$. Betrachten wir z.B. die Klasse BPP aller Sprachen, die von probabilistischen Polynomialzeit-TM mit zweiseitig beschränkter Fehlerwahrscheinlichkeit akzeptiert werden. Da nicht jede probabilistische TM die letzte Eigenschaft besitzt, ist BPP eine Promise-Klasse. Die relativierte Klasse $\mathrm{BPP}^O$ enthält alle Mengen, die relativ zu einem Orakel $O$ von einer probabilistischen TM akzeptiert werden, die relativ zu $O$ beschränkten Fehler hat. Die Typ-2-Klasse **BPP** andererseits verlangt, dass die Maschinen beschränkten Fehler für alle Eingaben und *alle* Orakel besitzen. Deshalb besteht $\mathbf{BPP}[O]$ aus allen Mengen, die relativ zu Orakel $O$ von einer probablistischen TM akzeptiert werden, die relativ zu allen Orakeln beschränkten Fehler besitzt. Deshalb gilt $\mathbf{BPP}[O] \subseteq \mathrm{BPP}^O$, aber die umgekehrte Richtung ist nicht klar. Es gibt sogar ein Orakel $O$, für das $\mathbf{BPP}[O] \neq \mathrm{BPP}^O$ gilt [CIY97]. Bei generischen Orakeln ändert sich die Situation jedoch. Cook u.a. zeigten, dass $\mathbf{BPP}[G] = \mathrm{BPP}^G$ für jedes generische Orakel $G$ gilt. In [CIY97] wurde auch von mehreren anderen Klassen nachgewiesen, dass sie sich wie BPP verhalten. Eine komplette Charakterisierung der Klassen $\mathcal{C}$ mit Typ-2-Entsprechung $\boldsymbol{\mathcal{C}}$, die die Eigenschaft $\mathcal{C}^G = \boldsymbol{\mathcal{C}}[G]$ für generisches $G$ besitzen, wurde nicht erreicht.

Wir können, indem wir die Methode des Forcierens anwenden, eine beinahe vollständige Antwort auf diese Frage für Klassen geben, die durch Paare von Blattsprachen $(A, R)$ definiert werden. Die hinreichende Bedingung, die wir im Satz 3.4.13 erhalten, nämlich dass $A \cup R$ co-rekursiv-aufzählbar (d.h. das Komplement einer rekursiv-aufzählbaren Menge) sein muss, ist sehr allgemein und wird von allen uns bekannten Blattsprachen-definierbaren Sprachen erfüllt. Insbesondere sind natürlich komplementäre Paare von Blattsprachen $(A = \overline{R})$ co-rekursiv-aufzählbar. Für alle Blattsprachen $A$ gilt deshalb $\mathrm{BLeaf}^G(A) = \mathbf{BLeaf}(A)[G]$. Dies beweist die Äquivalenz von (2) und (3) in Korollar 3.4.11.

Bevor wir zum Satz 3.4.13 kommen können, benötigen wir einige Definitionen und Resultate aus der Literatur. Eine *Orakeleigenschaft* ist eine Typ-2-Relation $S(x, X)$, die nicht von den Worten $x$, sondern nur von den Mengen $X$ abhängt. Deshalb bezeichnen wir Orakeleigenschaften mit $S(X)$. Wir sagen, ein endliches Orakel $v$ *forciert eine Orakeleigenschaft* $S$, falls $S(X)$ für alle Orakel $X$ gilt, die $v$ erweitern. Das folgende Lemma ist aus der Literatur bekannt. Ein Beweis findet sich z.B. in [CIY97, Lemma 2.1].

**Lemma 3.4.12.** *Sei $S$ eine $\Pi_1^0$-Orakeleigenschaft und sei $G$ ein generisches*

*Orakel. Dann gilt $S(G)$ genau dann, wenn irgendein endliches Präfix von $G$ die Eigenschaft $S$ forciert.*

Nun können wir den folgenden Satz beweisen:

**Satz 3.4.13.** *Sei $(A, R)$ ein Paar von Blattsprachen, wobei $A \cup R$ co-rekursiv-aufzählbar ist. Dann gilt* $\mathrm{BLeaf}^G(A, R) = \mathbf{BLeaf}(A, R)[G]$ *für jedes generische Orakel $G$.*

*Beweis.* Wir müssen nur die Inklusion $\subseteq$ zeigen. Unser Beweis folgt dem Beweis von Proposition 4.2 in [CIY97]. Sei $L \in \mathrm{BLeaf}^G(A, R)$. Also existiert eine balancierte NPOTM $M$, so dass für alle $x$ gilt, $\mathrm{leafstring}_{M^G}(x) \in A \cup R$ und $x \in L \Leftrightarrow \mathrm{leafstring}_{M^G}(x) \in A$. Wir definieren die Orakeleigenschaft $Q$ durch

$$Q(X) \Longleftrightarrow_{\mathrm{def}} \text{für alle } x,\ \mathrm{leafstring}_{M^X}(x) \in A \cup R.$$

Da $A \cup R$ co-rekursiv-aufzählbar ist, ist $Q$ eine $\Pi_1^0$-Orakeleigenschaft, für die $Q(G)$ gilt. Deshalb existiert aufgrund von Lemma 3.4.12 ein endliches Präfix $v$ von $G$, so dass für alle Orakel $O$, die $v$ erweitern, $Q(O)$ gilt. Wir definieren nun eine Typ-2-Relation $S'$ durch

$$S'(x, X) \Longleftrightarrow_{\mathrm{def}} \mathrm{leafstring}_{M^{X^v}}(x) \in A,$$

wobei wir mit $X^v$ das Orakel bezeichnen, das dadurch entsteht, dass in $X$ das Präfix mit Länge $|v|$ durch $v$ ersetzt wird. Offensichtlich gilt $S' \in \mathbf{BLeaf}(A, R)$ und wegen $G^v = G$ erhalten wir $S'[G] = L$. Damit haben wir $L \in \mathbf{BLeaf}(A, R)[G]$ gezeigt. □

## 3.5 Das P-NP-Problem

In der Einführung zu diesem Kapitel hatten wir erwähnt, dass generische Orakel möglicherweise dabei helfen könnten, eine Lösung für das P-NP-Problem zu finden und das mit der Tatsache begründet, dass Blum und Impagliazzo gezeigt haben, dass $\mathrm{P} \neq \mathrm{NP}$, falls $\mathrm{P} \neq \mathrm{UP}$ bezüglich eines generischen Orakels. Wir wissen außerdem, dass es ein Orakel gibt, das P und UP trennt. Nun könnte man hoffen, dass man mithilfe von Korollar 3.4.3 ein generisches trennendes Orakel finden kann. Dazu müsste aber UP durch eine Blattsprache beschreibbar sein. Leider gibt es aber keine Blattsprache $A$, so dass $\mathrm{UP} = \mathrm{BLeaf}(A)$ relativierbar gilt (siehe [BCS92]). UP kann als echte Promise-Klasse nur mit Blattsprachenpaaren beschrieben werden und

für diese gilt Satz 3.3.5 nicht (während Satz 3.2.1 sehr wohl für Paare von Blattsprachen gilt.)

Das Hauptproblem im Beweis des Satzes 3.3.5, das den Sprachenpaaren $(A, B)$ entgegensteht, ist, die Dichtheit für die Mengen $C_{2m}$ im Beweis des Lemmas 3.3.4 nachzuweisen. Eine interessante Frage ist nun, welche Bedingung für die Sprachenpaare hinreichend ist, um die Dichtheit der $C_{2m}$ nachzuweisen. Eine Bedingung haben wir schon gefunden: $A = \overline{B}$. Vielleicht gibt es ja allgemeinere Bedingungen und vielleicht erfüllt ja ein Paar, das UP beschreibt, eine dieser Bedingungen. Dann wäre P $\neq$ NP bewiesen.

# Kapitel 4

# Lindström-Quantoren und Blattsprachen

## 4.1 Einführung

Man kann die Klasse NP nicht nur durch nichtdeterministische Turing-Maschinen, sondern auch mithilfe der Klasse P und Existenzquantoren darstellen: Eine Menge $A$ ist genau dann in NP, wenn eine Menge $B \in \mathrm{P}$ und ein Polynom $p$ existieren, so dass gilt $A = \left\{ x \in \mathbb{N} \;\middle|\; \exists y \in \mathbb{N}(|y| \leq p(|x|) \wedge (x, y) \in B) \right\}$. Mengen in NP sind also *Projektionen* von Mengen aus P. Da die Blattsprachen eine Verallgemeinerung des Nichtdeterminismus sind und eine Parallele zwischen Nichtdeterminismus und Existenzquantoren besteht, kann man also Blattsprachen als eine Art von verallgemeinerten Quantoren bezeichnen.

In der Logik werden auch Existenzquantoren verwendet. So, wie es bei Turing-Maschinen eine Verallgemeinerung – nämlich die Blattsprachen – gibt, existieren in der Logik ebenfalls Verallgemeinerungen des Existenzquantors. Eine solche Verallgemeinerung sind die *Lindström-Quantoren*, siehe [EF95, Kap. 10.1].

*Reguläre Sprachen* werden unter anderem mithilfe von *endlichen Automaten* beschrieben. Hier unterscheidet man deterministische endliche Automaten (DEA) und nichtdeterministische endliche Automaten (NEA). So, wie man nichtdeterministische TM mithilfe von Blattsprachen verallgemeinern kann, kann man auch nichtdeterministische endliche Automaten zu *Blattautomaten* (BA) verallgemeinern. Blattsprachen für endliche Automaten wurden zum ersten Mal in [PV01] betrachtet.

Ein bekanntes Resultat ist die Tatsache, dass NEA nicht mächtiger sind als deterministische DEA, dass also beide Maschinenmodelle die gleichen

Sprachen akzeptieren können. Ein Ergebnis in [PV01] war nun, dass man sogar mit regulären Blattsprachen die Mächtigkeit des Maschinenmodells nicht vergrößert.

Der Satz von Büchi-Elgot-Trachtenbrot stellt einen Zusammenhang zwischen logischen Formeln mit Existenzquantoren und regulären Mengen dar. Wir wollen in diesem Kapitel zeigen, dass auch zwischen den verallgemeinerten logischen Quantoren und den verallgemeinerten endlichen Automaten ein Zusammenhang besteht.

## 4.2 Grundlagen

### 4.2.1 Endliche Automaten und Blattsprachen

In diesem Kapitel werden wir *endliche Automaten mit verallgemeinerten Akzeptanzkriterien*, wie sie in [PV01] eingeführt wurden, definieren.

Sie beruhen auf dem Modell der nichtdeterministischen endlichen Automaten. Ein solcher Automat erzeugt auf Eingabewort $w$ einen Baum möglicher Berechnungen. Wir wollen die Blätter des Baumes auf natürliche Weise ordnen. Deshalb machen wir folgende Definition:

Ein *endlicher Blattautomat (BA)* ist ein Tupel $M = (Z, \Sigma, \delta, s, \Gamma, \beta)$, wobei $Z$ die endliche *Zustandsmenge*, $\Sigma$ das *Eingabealphabet*, $\delta : Z \times \Sigma \to Z^+$ die *Übergangsfunktion*, $s \in Z$ der *Startzustand*, $\Gamma$ das *Blattalphabet* und $\beta : Z \to \Gamma$ eine Funktion ist, die jedem Zustand $z$ einen Wert $\beta(z)$ zuordnet. Für $z \in Z$ und $a \in \Sigma$ enthält das Zustandswort $\delta(z, a)$ alle möglichen Nachfolgerzustände von $M$ wenn der Buchstabe $a$ im Zustand $z$ gelesen wird, und die Anordnung der Zustände in diesem Wort definiert eine *totale Ordnung auf den Nachfolgezuständen*. Laut Definition ist es möglich, dass ein Zustand im Wort der Nachfolgezustände mehrmals vorkommen kann.

Sei $M$ wie oben. Der Berechnungsbaum $T_M(w)$ von $M$ auf Eingabe $w$ ist ein markierter gerichteter Baum, der wie folgt definiert ist:

- Die Wurzel von $T_M(w)$ ist mit $(s, w)$ markiert.
- Sei $v$ ein Knoten in $T_M(w)$ mit der Markierung $(z, x)$, wobei $x \neq \varepsilon$, d.h. $x = ay$ für ein $a \in \Sigma$ und $y \in \Sigma^*$. Sei $\delta(z, a) = z_1 \dots z_k$. Dann hat $v$ $k$ Kinder in $T_M(w)$, die die Markierungen $(z_1, y), \dots, (z_k, y)$, in dieser Reihenfolge, besitzen.

Jedes Blatt des Baumes hat eine Markierung der Form $(z, \varepsilon)$ mit $z \in Z$. Wenn wir einem Blatt mit der Markierung $(z, \varepsilon)$ das Zeichen $\beta(z)$ zuordnen, dann nennen wir die Zeichenkette der den Blättern zugeordneten Zeichen,

von links nach rechts in der durch $\delta$ induzierten Anordnung gelesen, den leafstring$^M(w)$.

**Definition 4.2.1.** Für ein $A \subseteq \Gamma^*$ besteht die Klasse Leaf$^{\mathrm{FA}}(A)$ aus allen Sprachen $B \subseteq \Sigma^*$, für die ein Blattautomat $M$ mit Eingabealphabet $\Sigma$ und Blattalphabet $\Gamma$ existiert, so dass für alle $w \in \Sigma^*$ gilt: $w \in B \Leftrightarrow$ leafstring$^M(w) \in A$.

Das folgende Resultat aus [PV01] über die Mächtigkeit regulärer Blattsprachen für endliche Automaten werden wir für eines unserer Resultate benötigen:

**Proposition 4.2.2.** Leaf$^{FA}$(REG) = REG.

### 4.2.2 Logik

Wir folgen der Standardnotation für die monadische Logik zweiter Stufe, wie sie z.B. in [Str94] eingeführt wird. Wir betrachten nur *String-Signaturen*, d.h. Signaturen der Form $\langle P_{a_1}, \ldots, P_{a_s} \rangle$, wobei alle Prädikate $P_{a_i}$ unär sind und in jeder Struktur $\mathcal{A}$ folgendes gilt: $\mathcal{A} \models P_{a_i}(j)$ gdw. das $j$-te Zeichen der Eingabe ist der Buchstabe $a_i$. Solche Strukturen sind also Wörter über dem Alphabet $\{a_1, \ldots, a_s\}$. Variablen erster Stufe erstrecken sich über Positionen in solch einem Wort, d.h. von 1 bis zur Wortlänge $n$. Variablen zweiter Stufe erstrecken sich über Untermengen von $\{1, \ldots, n\}$. Das lineare Ordnungssymbol der Logik bezieht sich auf numerische Ordnung auf $\{1, \ldots, n\}$. Aus technischen Gründen nehmen wir außerdem an, dass jedes Alphabet eine ihm innewohnende lineare Ordnung besitzt und wir schreiben deshalb Alphabete als Symbolfolgen. So schreiben wir z.B. im obigen Fall $(a_1, \ldots, a_s)$.

Die Basisformeln sind in der üblichen Weise aus Variablen erster und zweiter Ordnung, Booleschen Konnektoren $\{\wedge, \vee, \neg\}$, den relevanten Prädikaten $P_{a_i}$, den Symbolen $\{=, <\}$, den Konstanten min und max, den Quantoren $\{\exists, \forall\}$ erster und zweiter Stufe und Klammern aufgebaut. Für eine Formel $\varphi$ bezeichnen wir die durch $\varphi$ erzeugte Sprache (d.h. die Menge aller Modelle von $\varphi$) mit $L_\varphi$. Also ist $L_\varphi$ eine Menge von Wörtern. Wir benutzen das Zeichen „$\triangleq$“ um Formeln zu benennen, z.B. $\varphi \triangleq \exists x P_1(x)$.

SOM ist die Klasse aller durch die gerade beschriebenen Formeln darstellbaren Sprachen. Die Buchstaben „SOM“ stehen für „Second Order Monadic Logic“, also “monadische Logik zweiter Stufe“. In der Literatur wird diese Logik manchmal mit „MSO“ bezeichnet. FO ist die Unterklasse von SOM, die nur Sprachen enthält, die durch Formeln erster Ordnung beschrieben werden

können. Es ist bekannt [MP71], dass FO der Klasse der sternfreien Sprachen entspricht. Wir werden uns in diesem Kapitel mit dem folgenden bekannten Resultat beschäftigen (siehe auch [BE58, Büc62, Tra61]):

**Proposition 4.2.3 (Satz von Büchi-Elgot-Trachtenbrot).** *Die Klassen* SOM *und* REG *sind identisch.*

Nun werden wir unsere Logik um verallgemeinerte Quantoren erweitern.

**Definition 4.2.4.** Wir betrachten eine Sprache $L$ über dem Alphabet $\Sigma = (a_1, a_2, \ldots, a_s)$. Mithilfe dieser Sprache kann ein Lindström-Quantor $Q_L$ definiert werden, der auf eine Sequenz von $s-1$ Formeln wie folgt angewendet werden kann:

Sei $\overline{x}$ ein $k$-Tupel von Variablen (deren Wertebereich, wie gesehen, von 1 bis zur „Eingabelänge“ $n$ reicht). Wir nehmen die lexikographische Ordnung auf $\{1, 2, \ldots, n\}^k$ an und schreiben $\overline{x}^{(1)} < \overline{x}^{(2)} < \cdots < \overline{x}^{(n^k)}$ für die Folge möglicher Werte, die $\overline{x}$ annehmen kann. Seien $\varphi_1(\overline{x})$, …, $\varphi_{s-1}(\overline{x})$ Formeln und komme jede der Variablen aus $\overline{x}$ in jeder Formel $\varphi_i(\overline{x})$ als freie Variable vor. Dann kann man mithilfe des $k$-adischen *Lindström-Quantors* $Q_L$ eine neue Formel $Q_L\overline{x}\big[\varphi_1(\overline{x}), \varphi_2(\overline{x}), \ldots, \varphi_{s-1}(\overline{x})\big]$ definieren, in der $\overline{x}$ gebunden ist. Diese Formel wird auf einem Wort $w = w_1 \cdots w_n$ genau dann wahr, wenn das Wort $v = v_1 \ldots v_{n^k}$ der Länge $n^k$ zu $L$ gehört, wobei der $i$-te Buchstabe $v_i$ von $v$ folgendermaßen definiert ist:

$$v_i =_{\text{def}} \begin{cases} a_1 & \text{falls } w \models \varphi_1(\overline{x}^{(i)}), \\ a_2 & \text{falls } w \models \neg\varphi_1(\overline{x}^{(i)}) \wedge \varphi_2(\overline{x}^{(i)}), \\ & \vdots \\ a_{s-1} & \text{falls } w \models \neg\varphi_1(\overline{x}^{(i)}) \wedge \cdots \wedge \neg\varphi_{s-2}(\overline{x}^{(i)}) \wedge \varphi_{s-1}(\overline{x}^{(i)}), \\ a_s & \text{falls } w \models \neg\varphi_1(\overline{x}^{(i)}) \wedge \cdots \wedge \neg\varphi_{s-2}(\overline{x}^{(i)}) \wedge \neg\varphi_{s-1}(\overline{x}^{(i)}). \end{cases}$$

Wir bezeichnen $v$ auch als $f^{\overline{x}}_{\varphi_1 \cdots \varphi_{s-1}}(w)$.

Man kann so z.B. auch den schon bekannten Existenzquantor erster Ordnung darstellen. Dafür benötigt man $s = 2$, $\Sigma = \{0, 1\}$ und $L_\exists = 0^*1(0+1)^*$. Der Allquantor kann analog durch die Sprache $L_\forall =_{\text{def}} 1^*$ dargestellt werden. Die Quantoren $Q_{L_{\text{mod } p}}$ für $p > 1$ sind unter dem Namen Modulo-Zähl-Quantoren bekannt.

Die Lindström-Quantoren der Definition 4.2.4 sind genau das, was als „Lindström-Quantoren auf Strings“ bezeichnet wurde [BV98]. Die ursprüngliche, generellere Definition aus [Lin66] benutzt Transformationen auf beliebigen Strukturen, nicht notwendigerweise mit String-Signatur. In diesem

Kapitel werden wir uns nur mit Reduktionen auf Sprachen oder algebraischen Wortproblemen beschäftigen; deshalb scheint uns die obige Definition in unserem Zusammenhang die natürlichste.

Sei $M$ ein endliches Monoid. Dann definiert jedes $S \subseteq M$ ein $M$-Wortproblem, d.h. eine Sprache $\mathcal{W}(S, M)$ die aus allen Worten $w$ über dem Alphabet $M$ besteht, die „ausmultipliziert“ ein Element von $S$ ergeben.

Die folgende Definition stammt von Barrington, Immerman und Straubing [BIS90].

**Definition 4.2.5.** Wenn $L$ Wortproblem über einem endlichen Monoid ist, dann nennen wir den Lindströmquantor $Q_L$ einen *Monoidquantor*.

Es ist bekannt, dass jede reguläre Sprache ein homomorphes Urbild eines Wortproblems über einem Monoid ist und dass umgekehrt jedes Wortproblem über einem Monoid regulär ist. Deshalb ist ein Monoidquantor nichts anderes als ein Lindströmquantor $Q_L$ mit regulärem $L$.

Die Klasse $Q_{\mathrm{Mon}}\mathrm{FO}$ ist die Klasse aller Sprachen, die durch die Anwendung eines Monoidquantoren auf ein passendes Tupel von FO-Formeln definiert werden kann. Alle Sprachen, die durch FO-Formeln mit dem zusätzlichen Monoidquantor definierbar sind, nennen wir $\mathrm{FO}(Q_{\mathrm{Mon}})$.

Es ist schon länger bekannt [BIS90], dass $Q_{\mathrm{Mon}}\mathrm{FO}$ den regulären Sprachen entspricht. Dieses Ergebnis wurde vor kurzem in [LMSV01] erweitert:

**Satz 4.2.6.** $\mathrm{FO}(Q_{\mathrm{Mon}}) = Q_{\mathrm{Mon}}\mathrm{FO} = \mathrm{REG}$.

Eines der Resultate dieses Kapitels wird sein, dass in diesem Zusammenhang die Quantoren erster Stufe durch monadische Quantoren zweiter Stufe ersetzt werden können, ohne die Ausdrucksstärke zu erhöhen.

Ein *Gruppoid* ist eine endliche Multiplikationstabelle mit Identität. Für ein festes Gruppoid $G$ definiert jedes $S \subseteq G$ ein $G$-Wortproblem, also eine Sprache $\mathcal{W}(S, G)$, die aus den Worten $w$ über $G$ besteht, die in einer solchen Weise geklammert werden können, dass $w$ zu einem Element aus $S$ ausmultipliziert. Gruppoid-Wortprobleme stehen zu kontextfreien Sprachen in derselben Beziehung wie Monoid-Wortprobleme zu regulären Sprachen: Jedes Wortproblem ist kontextfrei und jede kontextfreie Sprache ist ein homomorphes Urbild eines Wortproblem. Dieses Ergebnis wird in [BLM93] Valiant zugeordnet.

Die folgende Definition stammt von Bédard, Lemieux und McKenzie [BLM93]:

**Definition 4.2.7.** Falls $L$ ein Wortproblem über einem endlichen Gruppoid ist, so nennen wir den Lindströmquantor $Q_L$ einen *Gruppoidquantor*.

Der Gebrauch von Gruppoidquantoren wird hier durch das Symbol $Q_{\mathrm{Grp}}$ dargestellt, das genau so benutzt wird, wie $Q_{\mathrm{Mon}}$ oben.

Lindströmquantoren zweiter Stufe über Zeichenketten wurden in [BV98] eingeführt. Hier sind wir hauptsächlich an den Quantoren interessiert, die Mengenvariablen binden, so genannte *monadische Quantoren*.

**Definition 4.2.8.** Wir betrachten eine Sprache $L$ über Alphabet $\Sigma = (a_1, a_2, \ldots, a_s)$. Sei $\overline{X} = (X_1, \ldots, X_k)$ ein $k$-Tupel von unären Variablen zweiter Stufe (d.h. Mengenvariablen), deren Werte, wie oben erwähnt, Untermengen von {1,...,n} sind. Es gibt $2^{nk}$ verschiedene Instanzen (Belegungen) von $\overline{X}$. Wir nehmen die folgende Ordnung auf diesen Instanzen an: Sei jede Instanz eines einzelnen $X_i$ durch die Bitfolge $s_1^i \cdots s_n^i$ kodiert, wobei gelten soll $s_j^i = 1 \iff j \in X_i$. Dann kodieren wir eine Instanz von $\overline{X}$ durch die Bitfolge $s_1^1 s_1^2 \cdots s_1^k s_2^1 s_2^2 \cdots s_2^k \cdots s_n^1 s_n^2 \cdots s_n^k$ und ordnen die Instanzen lexikographisch nach ihren Kodierungen. Wir schreiben $\overline{X}^{(1)} < \overline{X}^{(2)} < \cdots < \overline{X}^{(2^{nk})}$ für die geordnete Folge aller Instanzen.

Seien $\varphi_1(\overline{X}), \varphi_2(\overline{X}), \ldots, \varphi_{s-1}(\overline{X})$ Formeln und komme jede der Variablen aus $\overline{X}$ in jeder Formel $\varphi_i(\overline{X})$ als freie Variable vor. Dann kann man mithilfe des *monadischen Lindströmquantors zweiter Stufe* $Q_L$ eine neue Formel $\varphi = Q_L\overline{X}\big[\varphi_1(\overline{X}), \varphi_2(\overline{X}), \ldots, \varphi_{s-1}(\overline{X})\big]$ definieren, in der $\overline{X}$ gebunden ist. Diese Formel wird auf einem Wort $w = w_1 \cdots w_n$ genau dann wahr, wenn das Wort $v = v_1 \ldots v_{2^{nk}}$ der Länge $2^{nk}$ zu $L$ gehört, wobei der $i$-te Buchstabe $v_i$ von $v$ folgendermaßen definiert ist:

$$v_i =_{\mathrm{def}} \begin{cases} a_1 & \text{falls } w \models \varphi_1(\overline{X}^{(i)}), \\ a_2 & \text{falls } w \models \neg\varphi_1(\overline{X}^{(i)}) \wedge \varphi_2(\overline{X}^{(i)}), \\ & \vdots \\ a_{s-1} & \text{falls } w \models \neg\varphi_1(\overline{X}^{(i)}) \wedge \cdots \wedge \neg\varphi_{s-2}(\overline{X}^{(i)}) \wedge \varphi_{s-1}(\overline{X}^{(i)}), \\ a_s & \text{falls } w \models \neg\varphi_1(\overline{X}^{(i)}) \wedge \cdots \wedge \neg\varphi_{s-2}(\overline{X}^{(i)}) \wedge \neg\varphi_{s-1}(\overline{X}^{(i)}). \end{cases}$$

Wir nennen das Wort $v$ auch $f^{\overline{X}}_{\varphi_1 \cdots \varphi_{s-1}}(w)$.

Wenn wir als Beispiele wieder die Sprachen $L_\exists$ und $L_\forall$ betrachten, so erhalten wir die üblichen Existenz- und Allquantoren zweiter Stufe.

Falls das obige $L$ ein Monoid-Wortproblem ist, dann nennen wir $Q_L$ einen monadischen Monoidquantor zweiter Stufe; falls $L$ ein Gruppoid-Wortproblem ist, dann nennen wir $Q_L$ einen monadischen Gruppoidquantor zweiter Stufe.

Die Klasse mon-$Q_L^1$FO enthält alle Sprachen, die durch die Anwendung eines bestimmten monadischen Monoidquantors $Q_L$ zweiter Stufe auf ein

passendes Tupel von Formeln, die selbst keine Quantoren zweiter Stufe enthalten, definierbar sind. Die Klasse mon-$Q^1_{\mathrm{Mon}}$FO wird analog definiert, wobei hier beliebige monadische Monoidquantoren zweiter Stufe erlaubt sind. Die Klasse mon-$Q^1_{\mathrm{Mon}}$SOM wird analog definiert, wobei hier Tupel von SOM-Formeln erlaubt sind. Die Klasse SOM(mon-$Q^1_{\mathrm{Mon}}$) wird analog definiert, wobei aber die Monoidquantoren genau so wie alle anderen Quantoren benutzt werden können. Klassen mit mon-$Q^1_{\mathrm{Grp}}$ werden analog definiert.

Wir haben noch nicht beschrieben, wie Formeln mit freien Variablen auf String-Signaturen behandelt werden sollen. Hier folgen wir der in [Str94, pp. 14ff] eingeführten Notation: Formeln mit freien Variablen beschreiben Sprachen von Wörtern, an deren einzelne Buchstaben noch Variablen erster und zweiter Ordnung angehängt werden. Diese erweiterten Wörter werden $(\mathcal{V}_1, \mathcal{V}_2)$-*Strukturen* genannt, wobei $\mathcal{V}_1$ eine Menge von Variablen erster Stufe und $\mathcal{V}_2$ eine Menge von Variablen zweiter Stufe ist. Formal ist eine $(\mathcal{V}_1, \mathcal{V}_2)$-Struktur über dem ursprünglichen Buchstabenalphabet $\Sigma$ ein Wort über dem Alphabet $\Sigma \times 2^{\mathcal{V}_1} \times 2^{\mathcal{V}_2}$.

## 4.3 Blattsprachen und verallgemeinerte Quantoren

Wir beweisen zuerst ein technisches Lemma, das wir in den späteren Beweisen noch benötigen werden.

**Lemma 4.3.1.** *Sei $\Sigma$ ein Alphabet. Sei $\mathcal{V}_1$ eine Menge von Variablen erster Ordnung und sei $\mathcal{V}_2$ eine Menge von Variablen zweiter Ordnung. Sei $\overline{X} = (X_1, \ldots, X_k)$ ein Vektor von monadischen Variablen zweiter Stufe. Seien $\varphi_1(\overline{X}), \varphi_2(\overline{X}), \ldots, \varphi_{s-1}(\overline{X})$ logische Formeln, die reguläre Mengen von $(\mathcal{V}_1, \mathcal{V}_2 \cup \{X_1, \ldots, X_k\})$-Stukturen über $\Sigma$ beschreiben.*

*Sei $\Gamma = (a_1, a_2, \ldots, a_s)$ ein geordnetes Alphabet und sei $L \subseteq \Gamma^*$. Definiere $\varphi \triangleq Q_L \overline{X} \big[\varphi_1(\overline{X}), \varphi_2(\overline{X}), \ldots, \varphi_{s-1}(\overline{X})\big]$. Dann ist $\varphi$ eine Formel, die eine Menge $L_\varphi$ von $(\mathcal{V}_1, \mathcal{V}_2)$-Strukturen über $\Sigma$ definiert und es gibt einen Blattautomaten $M$, der die Sprache $L_\varphi$ mit Blattsprache $L$ akzeptiert.*

*Beweis.* Dass $\varphi$ eine Menge von $(\mathcal{V}_1, \mathcal{V}_2)$-Strukturen beschreibt, ist offensichtlich, da $\varphi$ laut Definition die Variablen $X_1, \ldots, X_k$ bindet. Nun müssen wir noch den Blattautomaten $M$ beschreiben.

Seien $M_1, \ldots, M_{s-1}$ deterministische Automaten, die $L_{\varphi_1}, \ldots, L_{\varphi_{s-1}}$ akzeptieren. Für $1 \leq i < s$ sei $M_i = (Q_i, \Sigma \times 2^{\mathcal{V}_1} \times 2^{\mathcal{V}_2 \cup \{X_1, \ldots, X_k\}}, \delta_i, s_i, A_i)$. Betrachte das übliche Kreuzprodukt von Automaten $M_i$ in der Form

$$M_\times =_{\mathrm{def}} \big(\times_{i=1}^{s-1} Q_i, \Sigma \times 2^{\mathcal{V}_1} \times 2^{\mathcal{V}_2 \cup \{X_1, \ldots, X_k\}}, \delta_\times, (s_1, s_2, \ldots, s_{s-1}), \times_{i=1}^{s-1} A_i\big)$$

mit Übergangsfunktion

$$\delta_\times((q_1, q_2, \ldots, q_{s-1}), (v, S, T)) = (\delta_1(q_1, (v, S, T)), \ldots, \delta_{s-1}(q_{s-1}, (v, S, T))).$$

Unser Blattautomat

$$M =_{\text{def}} \left(\times_{i=1}^{s-1} Q_i, \Sigma \times 2^{\mathcal{V}_1} \times 2^{\mathcal{V}_2}, \delta, (s_1, s_2, \ldots, s_{s-1}), \Gamma, \beta\right)$$

wird nun wie folgt definiert:

$M$ soll nur $(\mathcal{V}_1, \mathcal{V}_2)$-Strukturen als Eingabe haben. Deshalb müssen wir natürlich das Verhalten von $M_\times$ auf jeder möglichen Instanz von $\overline{X} = (X_1, \ldots, X_k)$ simulieren. Die Simulation dieser Instanzen geschieht auf den einzelnen Berechnungspfaden von $M$.

Jede Eingabeposition von $M_\times$ kann ein Element der Mengenvariable $X_i$ sein oder auch nicht. Anders ausgedrückt: Wenn $w$ die Eingabe von $M$ ist, dann muss $M$ für jeden gelesenen Buchstaben $(w_j, \mathcal{V}_1^j, \mathcal{V}_2^j)$ von $w$, wobei $w_j \in \Sigma$, $\mathcal{V}_1^j \subseteq \mathcal{V}_1$ und $\mathcal{V}_2^j \subseteq \mathcal{V}_2$ ist, $M_\times$ für die Fälle $X_i \in \mathcal{V}_2^j$ und $X_i \notin \mathcal{V}_2^j$ simulieren. Also gibt es für jeden Buchstaben von $w$ $2^k$ Möglichkeiten für simulierte Eingaben. Wir kodieren die Elemente von $\mathcal{P}(\{X_1, \ldots, X_k\})$ durch $S_i, 1 \leq i \leq 2^k$, wobei jede Menge $S_i$ auf folgende Weise definiert ist: $X_j \in S_i$ genau dann, wenn das $j$-te Bit von $\text{bin}_k(i-1)$ (die Binärkodierung der mit eventuellen führenden Nullen auf Länge $k$ gepaddeten natürlichen Zahl $i-1$) den Wert 1 besitzt.

Die Übergangsfunktion von $M$ ist nun definiert als

$$\begin{aligned}\delta((q_1, q_2, \ldots, q_{s-1}), (x, S, T)) = {} & \delta_\times((q_1, q_2, \ldots, q_{s-1}), (x, S, T \cup \{S_1\})) \\ & \delta_\times((q_1, q_2, \ldots, q_{s-1}), (x, S, T \cup \{S_2\})) \\ & \quad \vdots \\ & \delta_\times((q_1, q_2, \ldots, q_{s-1}), (x, S, T \cup \{S_{2^k}\}))\end{aligned}$$

Dabei ist zu beachten, dass die Überführungsfunktion für Blattautomaten auf Folgen von Zuständen abbildet. Schließlich definieren wir

$$\beta(q_1, \ldots, q_{s-1}) =_{\text{def}} \begin{cases} a_1, & \text{falls } q_1 \in A_1, \\ a_2, & \text{falls } q_1 \notin A_1 \wedge q_2 \in A_2, \\ & \vdots \\ a_{s-1}, & \text{falls } q_1 \notin A_1 \wedge \cdots \wedge q_{s-2} \notin A_{s-2} \wedge q_{s-1} \in A_{s-1} \\ a_s, & \text{falls } q_1 \notin A_1 \wedge \cdots \wedge q_{s-2} \notin A_{s-2} \wedge q_{s-1} \notin A_{s-1}. \end{cases}$$

$M$ auf Eingabe $w = w_1 \ldots w_n$ spannt nun einen Berechnungsbaum $T_M(w)$ der Tiefe $n$ und Grad $2^k$ auf. Ein solcher Baum besitzt $(2^k)^n = 2^{nk}$ Blätter.

Wir werden nun zeigen, dass das Blattwort auf $T_M(w)$ identisch zum Wort $f^{\overline{X}}_{\varphi_1 \cdots \varphi_{s-1}}(w)$ ist.

Jede der $2^k$ Verzweigungen pro Buchstabe der Eingabe entspricht einer Menge $S_i, 1 \leq i \leq 2^k$. Deshalb können wir jeden Zweig durch einen Bitvektor $\text{bin}_k(i-1)$ (der $S_i$ beschreibt) kodieren. Bei der Anordnung der Blätter von $T_M(w)$ sind die Verzweigungen um so wichtiger, je näher sie der Wurzel des Baumes stehen. D.h. also, die Verzweigungen bei früheren Buchstaben der Eingabe sind wichtiger als die Verzweigungen bei späteren Buchstaben. Deshalb können wir einen Pfad in $T_M(w)$ durch die Bitfolge $\text{bin}_k(i_1-1)\text{bin}_k(i_2-1)\cdots\text{bin}_k(i_n-1)$ der Länge $nk$ kodieren. Die Folge hat die Bedeutung: Auf dem $j$-ten Eingabesymbol wurde die Menge $S_{i_j}$ ausgewählt. Falls Pfad $l$ eine lexikographisch kleinere Kodierung als Pfad $l'$ besitzt, dann steht das Blatt von $l$ im Blattwort $T_M(w)$ links vom Blatt von $l'$. Da es genau $2^k$ verschieden Bitfolgen $\text{bin}_k(i)$ gibt, entspricht jede dieser Folgen einer Menge $S_i$ und somit korrespondiert sie auch zu Zweig $i$ in einer Verzweigung des Berechnungsbaumes. Darum ist die Menge aller $2^{nk}$ Binärfolgen der Länge $nk$ komplett mit Kodierungen von Berechnungspfaden ausgefüllt. Daraus folgt, dass der Pfad mit Nummer $l$ im Berechnungsbaum durch die Bitfolge der Länge $nk$ kodiert ist, die die Binärdarstellung von $l-1$ ist.

Nun wollen wir zeigen, dass auf Pfad $l$ das Mengentupel $\overline{X}^{(l)}$ (wobei die Notation von Definition 4.2.8 verwendet wird) simuliert wird. Die Kodierung von Pfad $l$ ist die Binärfolge $z$, die die Zahl $l-1$ darstellt. Wir können $z$ in $n$ Teile der Länge $k$ aufteilen: $z = z_1 z_2 \ldots z_n$. Wie schon gesagt, kodiert die Bitfolge $z_j = z_j^1 z_j^2 \ldots z_j^k$ die Simulation von Mengen $X_i$ auf folgende Weise: $z_j^i = 1$ genau dann, wenn die Menge $X_i$ bei Eingabesymbol $j$ gewählt wurde. Also hängt der Wert von $X_i$ von den Werten von $z_1^i, z_2^i, \ldots, z_n^i$ des Strings $z = z_1^1 z_1^2 \ldots z_1^k z_2^1 z_2^2 \ldots z_2^k \ldots z_n^1 z_n^2 \ldots z_n^k$ ab. Da $z$ aber die Zahl $l-1$ repräsentiert, ist dies gerade die Definition des Tupels $\overline{X}^{(l)}$.

Also können wir schließen, dass auf Pfad $l$ des Berechnungsbaumes das Verhalten der Automaten $M_1, \ldots, M_{s-1}$ auf Eingabe $w$ und $\overline{X}^{(l)}$ simuliert wird. Jeder Automat $M_k$ simuliert die Formel $\varphi_k$. Falls also bei Automat $M_k$ beim Lesen des letzten Buchstabens der Eingabe der Zustand $q_k$ erreicht wird, dann gilt $q_k \in A_k \Leftrightarrow w \models \varphi_k(\overline{X}^{(l)})$.

Wenn wir die Definition der Funktion $\beta$ betrachten, dann sehen wir, dass

der Buchstabe auf dem Blatt von Pfad $l$

$$\begin{cases} a_1, & \text{falls } w \models \varphi_1(\overline{X}^{(l)}), \\ a_2, & \text{falls } w \models \neg\varphi_1(\overline{X}^{(l)}) \wedge \varphi_2(\overline{X}^{(l)}), \\ & \vdots \\ a_{s-1}, & \text{falls } w \models \neg\varphi_1(\overline{X}^{(l)}) \wedge \cdots \wedge \neg\varphi_{s-2}(\overline{X}^{(l)}) \wedge \varphi_{s-1}(\overline{X}^{(l)}) \\ a_s, & \text{falls } w \models \neg\varphi_1(\overline{X}^{(l)}) \wedge \cdots \wedge \neg\varphi_{s-2}(\overline{X}^{(l)}) \wedge \neg\varphi_{s-1}(\overline{X}^{(l)}) \end{cases}$$

ist. Aber dies ist der $l$-te Buchstabe des Wortes $f^{\overline{X}}_{\varphi_1\cdots\varphi_{s-1}}(w)$ (siehe Definition 4.2.8). Damit haben wir bewiesen, dass das Blattwort des Berechnungsbaums $T_M(w)$ identisch zum Wort $f^{\overline{X}}_{\varphi_1\cdots\varphi_{s-1}}(w)$ ist.

Daraus folgt

$$\begin{aligned} w \models Q_L\overline{X}\big[\varphi_1(\overline{X}), \varphi_2(\overline{X}), \ldots, \varphi_{s-1}(\overline{X})\big] \quad &\Leftrightarrow \quad f^{\overline{X}}_{\varphi_1\cdots\varphi_{s-1}}(w) \in L \\ &\Leftrightarrow \quad \text{leafstring}^M(w) \in L \end{aligned}$$

Also akzeptiert $M$ mit Blattsprache $L$ die Sprache $L_\varphi$. □

Eine Sprache $L \subseteq \Gamma^*$ besitzt einen *neutralen Buchstaben* $\epsilon \in \Gamma$, falls für alle $u, v \in \Gamma^*$ gilt: $uv \in L \iff u\epsilon v \in L$. Sei $\mathcal{N}$ die Klasse aller Sprachen mit neutralem Buchstaben.

Das Hauptergebnis dieses Abschnitts lautet nun, dass endliche Automaten mit Blattsprache $L$ genau die Sprachen akzeptieren, die durch einen monadischen $Q^1_L$-Quantor definiert werden, falls $L \in \mathcal{N}$ gilt.

**Satz 4.3.2.** *Für alle $L \in \mathcal{N}$ gilt* $\text{Leaf}^{FA}(L) = \text{mon-}Q^1_L\text{FO}$.

*Beweis.* $\text{Leaf}^{\text{FA}}(L) \supseteq \text{mon-}Q^1_L\text{FO}$:

Sei $\varphi$ eine mon-$Q^1_L$FO-Formel, die eine Sprache $L_\varphi \subseteq \Sigma^*$ definiert. Dann hat $\varphi$ die Form $Q_L\overline{X}\big[\varphi_1(\overline{X}), \varphi_2(\overline{X}), \ldots, \varphi_{s-1}(\overline{X})\big]$, wobei die $\varphi_i(\overline{X})$, $1 \leq i \leq s-1$ Formeln erster Ordnung sind, mit der Ausnahme, dass in ihnen die monadischen Variablen in $\overline{X}$ vorkommen können. Also folgt aus Lemma 4.3.1, dass ein Blattautomat $M_\varphi$ existieren muss, der die Sprache $L_\varphi$ akzeptiert.

$\text{Leaf}^{\text{FA}}(L) \subseteq \text{mon-}Q^1_L\text{FO}$:

Sei $\Gamma = \{a_1, a_2, \ldots, a_{s-1}, \epsilon\}$, $L \in \mathcal{N} \cap \mathcal{P}(\Gamma^*)$, wobei $\epsilon$ neutral ist, und sei $M = (Q, \Sigma, \delta, q_1, \Gamma, \beta)$ ein Blattautomat. Wenn $M$ mit Blattsprache $L$ arbeitet, akzeptiert er eine bestimmte Sprache $A \subseteq \Sigma^*$. Wir müssen eine mon-$Q^1_L$FO-Formel $\varphi$ mit $L_\varphi = A$ konstruieren.

Zuerst ordnen wir das Blattalphabet so an, dass $\epsilon$ das letzte Zeichen in $\Gamma$ ist; also $\Gamma = (a_1, a_2, \ldots, a_{s-1}, \epsilon)$. Sei $Q = \{q_1, \ldots, q_l\}$ und sei $m =_{\text{def}}$

$\max\{\, |\delta(q,a)| \mid q \in Q, a \in \Sigma \,\}$. Die Formel $\varphi$ wird die Form $Q_L\overline{X}[\varphi_1(\overline{X}), \varphi_2(\overline{X}), \ldots, \varphi_{s-1}(\overline{X})]$ besitzen, wobei $\overline{X} =_{\text{def}} (X_m, \ldots, X_1, Y_l, \ldots, Y_1)$ gilt.

Wir wollen die Nummer eines Berechnungspfades von $M$ auf Eingabe $w$ in den Variablen $X_i$ kodieren. Dies wird auf folgende Weise geschehen: $j \in X_i$ genau dann, wenn $M$ beim Lesen des $j$-ten Zeichens von $w$ die $i$-te Alternative der Nachfolgerfunktion wählt.

In den Variablen $Y_i$ wollen wir die Folge der Zustände von $M$ auf Eingabe $w$ für einen bestimmten Berechnungspfad kodieren. Dies wird auf folgende Weise geschehen: $j \in Y_i$ genau dann, wenn $M$ (auf dem Berechnungspfad, der durch die Variablen $X_l$ gegeben ist) beim Lesen des $j$-ten Zeichens der Eingabe im Zustand $q_i$ ist.

Die Formeln $\varphi_k$ müssen also bei Eingabe $w$ und einer Instanz von $\overline{X}$ folgendes ausdrücken:

1. Alle $\varphi_k$ müssen überprüfen, ob die $X_i$ einen Berechnungspfad von $M$ auf $w$ kodieren. Falls dies nicht der Fall ist (falls z.B. ein $j$ und $i_1 \neq i_2$ mit $j \in X_{i_1}$ und $j \in X_{i_2}$ existieren), dann muss $w \not\models \varphi_k(\overline{X})$ für alle $k \in \{1, \ldots, s-1\}$ gelten. (Ein Blick auf die Definition von $f^{\overline{X}}_{\varphi_1 \cdots \varphi_{s-1}}(w)$ zeigt, dass der erzeugte Buchstabe dann $\epsilon$ sein wird.)

2. Alle $\varphi_k$ müssen überprüfen, ob die $Y_i$ eine korrekte Folge von Zuständen von $M$ auf dem durch die $X_i$ kodierten Berechnungspfad kodieren. Falls dies nicht der Fall ist, so muss $w \not\models \varphi_k(\overline{X})$ für alle $k \in \{1, \ldots, s-1\}$ gelten. (Der Buchstabe von $f^{\overline{X}}_{\varphi_1 \cdots \varphi_{s-1}}(w)$ wird dann wieder $\epsilon$ sein.)

3. Falls $\overline{X}$ eine Berechnung von $M$ kodiert, falls also die $X_i$ die Nummer eines Berechnungspfades und die $Y_i$ die korrekte Folge von Zuständen auf diesem Pfad kodieren, dann gilt $w \models \varphi_k(\overline{X})$ genau dann, wenn $M$ auf dem von $X_i$ kodierten Berechnungspfad das Blattsymbol $a_k$ erzeugt. (Der Buchstabe von $f^{\overline{X}}_{\varphi_1 \cdots \varphi_{s-1}}(w)$ wird dann identisch zu dem von $M$ auf dem durch $X_i$ kodierten Pfad sein.)

Solche FO-Formeln sind leicht zu konstuieren.

Nun zeigen wir, dass das Wort $f^{\overline{X}}_{\varphi_1 \cdots \varphi_{s-1}}(w)$ im Wesentlichen mit dem Wort $\text{leafstring}^M(w)$ identisch ist. Das heißt genauer: Falls alle Vorkommen von $\epsilon$ aus beiden Wörtern gelöscht werden, sind sie identisch. Also sind die beiden Wörter äquivalent, was das Enthaltensein in $L$ angeht.

Da jede Berechnung von $M$ durch eine bestimmte Instanz von $\overline{X}$ kodiert wird, ist auch jeder Buchstabe von $\text{leafstring}^M(w)$ in $f^{\overline{X}}_{\varphi_1 \cdots \varphi_{s-1}}(w)$ enthalten. Und da jede Instanz von $\overline{X}$, die keine korrekte Berechnung von $M$ kodiert, den Buchstaben $\epsilon$ in $f^{\overline{X}}_{\varphi_1 \cdots \varphi_{s-1}}(w)$ erzeugt, wissen wir, dass die Anzahl der Vorkommen für alle von $\epsilon$ verschiedenen Buchstaben in beiden Worten identisch

sind. Nun müssen wir nur noch zeigen, dass die Reihenfolge der Buchstaben in beiden Worten identisch ist.

Seien $p_1$ und $p_2$ Berechnungspfade in $M$, $u$ und $v$ die auf diesen Pfaden erzeugten Blattsymbole und $\overline{X_1}$ und $\overline{X_2}$ die Kodierungen der korrekten Berechnungen, die zu diesen Pfaden gehören. (Oben wurde gezeigt, dass die Buchstaben $u$ und $v$ dann auch im Wort $f^{\overline{X}}_{\varphi_1 \cdots \varphi_{s-1}}(w)$ von $\overline{X_1}$ und $\overline{X_2}$ erzeugt werden.) Wir zeigen dass $p_1$ genau dann vor $p_2$ positioniert ist, wenn die Kodierung von $\overline{X_1}$ vor der Kodierung von $\overline{X_2}$ angeordnet ist.

Der Definition von $f^{\overline{X}}_{\varphi_1 \cdots \varphi_{s-1}}(w)$ zufolge kann die Kodierung einer Instanz $\overline{X_k}$ in $n$ Bitfolgen der Länge $m + l$ aufgeteilt werden. Jede dieser Folgen hat die Form $x_m \dots x_1 y_l \dots y_1$. Die $j$-te solche Folge entspricht dem $j$-ten Eingabesymbol, wobei zum einen $x_i = 1 \leftrightarrow j \in X_i$ gilt (es gibt immer nur ein $x_i = 1$, da die $\overline{X_k}$ Kodierung korrekter Berechnungen sind), und zum anderen $y_i = 1 \leftrightarrow j \in Y_i$ (es gibt auch immer nur ein $y_i = 1$).

Zwei unterschiedliche Berechnungspfade haben eine Wortposition, an der der erste Unterschied auftritt. Wir nennen diese Position $d$. Da die Berechnung von $M$ bis zu $d$ auf beiden Pfaden identisch ist, sind nicht nur die Entscheidungen der Nachfolgerfunktion für beide Pfade bis $d$ identisch, sondern auch die Folgen der von $M$ angenommenen Zustände.

Daraus folgt aber, dass die Kodierungen der $\overline{X_k}$ bis zur Bitfolge $d$ identisch sind. Also entscheidet diese Folge darüber, welche Kodierung von höherer Ordnung ist. Sei $x^1_m \dots x^1_1 y^1_l \dots y^1_1$ der $d$-te String von $\overline{X_1}$ und $x^2_m \dots x^2_1 y^2_l \dots y^2_1$ der $d$-te String von $\overline{X_2}$.

Das eindeutige $x^1_a = 1$ ist die Wahl für die Nachfolgefunktion des $d$-ten Eingabesymbols auf Berechnungspfad $p_1$, während $x^2_b = 1$ die entsprechende Auswahl für Pfad $p_2$ ist. Dann gilt genau dann $a < b$, wenn $p_1$ vor $p_2$ in $T_M(w)$ angeordnet ist. (Dies gilt aufgrund der Ordnung, die die Nachfolgerfunktion auf dem Berechnungsbaum induziert.) Außerdem ist der String $x^1_m \dots x^1_{b+1} x^1_b x^1_{b-1} \dots x^1_{a+1} x^1_a x^1_{a-1} \dots x^1_1 y^1_l \dots y^1_1 = 0 \dots 000 \dots 010 \dots 0 y^1_l \dots y^1_1$ von geringerer Ordnung als $x^2_m \dots x^2_{b+1} x^2_b x^2_{b-1} \dots x^2_{a+1} x^2_a x^2_{a-1} \dots x^2_1 y^2_l \dots y^2_1 = 0 \dots 010 \dots 000 \dots 0 y^2_l \dots y^2_1$. Also ist die Anordnung von $\overline{X_1}$ und $\overline{X_2}$ identisch zur Anordnung von $p_1$ und $p_2$.

Dies beschließt den Beweis, dass, wenn man vom Buchstaben $\epsilon$ absieht, die Wörter $f^{\overline{X}}_{\varphi_1 \cdots \varphi_{s-1}}(w)$ und $\text{leafstring}^M(w)$ identisch sind. Deshalb gilt

$$\begin{aligned} w \in A &\Leftrightarrow \text{leafstring}^M(w) \in L \\ &\Leftrightarrow f^{\overline{X}}_{\varphi_1 \cdots \varphi_{s-1}}(w) \in L \\ &\Leftrightarrow w \in L_\varphi. \end{aligned}$$

Dies heißt aber, dass $\varphi$ die von $M$ akzeptierte Sprache $A$ akzeptiert. $\square$

## 4.4 Monoidquantoren

Wir können jetzt eine Erweiterung von Proposition 4.2.3 angeben: Falls wir den monadischen Formalismus zweiter Stufe durch Monoidquantoren erweitern, erhalten wir dadurch keine größere Ausdrucksstärke.

**Satz 4.4.1.** $\text{mon-}Q^1_{\text{Mon}}\text{FO} = \text{FO}(\text{mon-}Q^1_{\text{Mon}}) =$
$\text{mon-}Q^1_{\text{Mon}}\text{SOM} = \text{SOM}(\text{mon-}Q^1_{\text{Mon}}) = \text{REG}.$

*Beweis.* Die Inklusionen

$$\text{mon-}Q^1_{\text{Mon}}\text{FO} \subseteq \text{mon-}Q^1_{\text{Mon}}\text{SOM} \subseteq \text{SOM}(\text{mon-}Q^1_{\text{Mon}})$$

und

$$\text{mon-}Q^1_{\text{Mon}}\text{FO} \subseteq \text{FO}(\text{mon-}Q^1_{\text{Mon}}) \subseteq \text{SOM}(\text{mon-}Q^1_{\text{Mon}})$$

sind trivial.

Um $\text{REG} \subseteq \text{mon-}Q^1_{\text{Mon}}\text{FO}$ zu zeigen benutzen wir, dass jede reguläre Sprache von einer SOM-Formel definiert werden kann, die aus einem einzigen Existenzquantor zweiter Stufe gefolgt von einer FO-Formel besteht. Dies ist ein Ergebnis aus [Tho82]. Außerdem wissen wir schon, dass Existenzquantoren auch Monoidquantoren sind. Sie entsprechen dem Quantor $Q_{L_\exists}$ mit $L_\exists = 0^*1(0+1)^*$.

Nun müssen wir nur noch $\text{SOM}(\text{mon-}Q^1_{\text{Mon}}) \subseteq \text{REG}$ zeigen. Wir folgen dabei dem Beweis von Satz III.1.1 in [Str94]. Dort wird $\text{SOM} \subseteq \text{REG}$ mithilfe einer Induktion über den Aufbau von SOM-Formeln gezeigt. Da $\text{SOM}(\text{mon-}Q^1_{\text{Mon}})$-Formeln auch nur SOM-Formeln mit nur einem neuen Konstruktionselement – den Lindströmquantoren – sind, müssen wir die Induktion nur um einen Fall erweitern:

Seien $\Sigma$ ein Alphabet und $\mathcal{V}_1$ und $\mathcal{V}_2$ Mengen von Variablen erster bzw. zweiter Ordnung. Eine Formel $\varphi \in \text{SOM}(\text{mon-}Q^1_{\text{Mon}})$ mit freien Variablen $\mathcal{V}_1 \cup \mathcal{V}_2$ akzeptiert dann eine Sprache $L_\varphi \in (\Sigma \times 2^{\mathcal{V}_1} \times 2^{\mathcal{V}_2})^*$. Wir müssen zeigen, dass $L_\varphi$ regulär ist. Wie wir schon erwähnt haben, benutzt der Beweis eine Induktion (siehe [Str94, pp. 21ff]). Der neue Fall im Induktionsschritt ist nun die Situation $\varphi \triangleq Q_L \overline{X}\big[\varphi_1(\overline{X}), \varphi_2(\overline{X}), \ldots, \varphi_{s-1}(\overline{X})\big]$. Nach Induktionsvoraussetzung existieren DEAs $M_1, M_2, \ldots, M_{s-1}$, die die $(\mathcal{V}_1, \mathcal{V}_2 \cup \{X_1, \ldots, X_k\})$-Strukturen, die Modelle für $\varphi_1, \varphi_2, \ldots, \varphi_{s-1}$ sind, akzeptieren. Lemma 4.3.1 sagt uns dann aber, dass es einen Blattautomaten $M_\varphi$ gibt, der $L_\varphi$ mit Blattsprache $L$ akzeptiert.

Da $L$ aber regulär ist, sagt uns Proposition 4.2.2 wiederum, dass $L_\varphi$ ebenfalls regulär ist. Damit ist die Induktion gezeigt und der Beweis abgeschlossen. □

Wir können nun eine *Normalform für die monadische Monoidlogik zweiter Stufe* angeben, wenn wir das obige Ergebnis mit dem im Beweis genannten Ergebnis von Thomas [Tho82] verbinden:

**Korollar 4.4.2.** *Jede* SOM(mon-$Q^1_{\mathrm{Mon}}$)*-Formel ist äquivalent zu einer Formel der Form* $\exists X \varphi(X)$*, wobei* $X$ *eine Mengenvariable und* $\varphi$ *eine Formel ohne Quantoren zweiter Stufe ist.*

## 4.5 Gruppoidquantoren

Zum Ende dieses Kapitels beschäftigen wir uns noch mit der Ausdrucksstärke von monadischen Gruppoidquantoren zweiter Stufe. Wir werden zeigen, dass sie ausdrucksstark genug sind, um jede Sprache in LOGCFL zu definieren. Dabei ist LOGCFL die Klasse aller Sprachen, die in logarithmischem Platz auf eine kontextfreie Sprache reduzierbar sind.

**Satz 4.5.1.** LOGCFL $\subseteq$ mon-$Q^1_{\mathrm{Grp}}$FO.

*Beweis.* In [LMSV01] zeigten Lauteman et al. LOGCFL $= Q_{\mathrm{Grp}}$FO. Das heißt aber, dass es für jede Formel $L \in$ LOGCFL eine Formel $\varphi$ mit $L = L_\varphi$ gibt, die aus einem verallgemeinerten Quantor für eine kontextfreie Sprache und einer ihm folgenden Formel der Logik erster Stufe besteht.

Solch ein CFL-Quantor kann aber durch einen monadischen Lindströmquantor zweiter Stufe über eine andere kontextfreie Sprache simuliert werden: Betrachten wir die Formel $\varphi \triangleq Q_L x_1 \cdots x_k \big[\psi(x_1 \cdots x_k)\big]$, wobei $\psi$ eine Formel erster Stufe und $L$ eine kontextfreie Sprache über $(0,1)$ ist. (Für größere Alphabete läuft der Beweis vollständig analog ab.) Sei $L'$ die Sprache, die man aus $L$ erhält, wenn man einen neutralen Buchstaben hinzunimmt. Sei z.B. $L'$ über dem Alphabet $(0,1,\epsilon)$ definiert. Dann setzen wir

$$\varphi' \triangleq Q_{L'} X_1 \cdots X_k \big[\psi_1(X_1 \cdots X_k), \psi_2(X_1 \cdots X_k)\big],$$

wobei

$$\psi_1(X_1 \cdots X_k) \triangleq \exists x_1 \cdots x_k \Big( \textstyle\bigwedge_{i=1}^k \big(x_i \in X_i \wedge \forall y (y \in X_i \to x_i = y)\big) \wedge \psi(x) \Big),$$
$$\psi_2(X_1 \cdots X_k) \triangleq \exists x_1 \cdots x_k \Big( \textstyle\bigwedge_{i=1}^k \big(x_i \in X_i \wedge \forall y (y \in X_i \to x_i = y)\big) \wedge \neg\psi(x) \Big).$$

Dann sind nur die Mengen, die aus einem einzigen Element bestehen, Zuweisungen der Mengenvariable $X_i$, die nicht zum neutralen Buchstaben $\epsilon$ in

$f_{\psi_1\psi_2}^{(X_1,\ldots,X_k)}$ führen. Und diese korrespondieren, bei gleicher relativer Anordnung, mit den Zuweisungen von $x_i$ in $\psi$. Deshalb gilt für jede Eingabe $w$, dass $f_{\psi_1\psi_2}^{(X_1,\ldots,X_k)}(w) \in L' \iff f_{\psi}^{(x_1,\ldots,x_k)}(w) \in L$, und somit gilt $L_\varphi = L_{\varphi'}$.

Damit haben wir gezeigt, dass LOGCFL $= Q_{\mathrm{Grp}}\mathrm{FO} \subseteq$ mon-$Q^1_{\mathrm{Grp}}\mathrm{FO}$ gilt. □

Nachdem sie in [PV01] endliche Blattautomaten eingeführt hatten, untersuchten Peichl und Vollmer die Mächtigkeit verschiedener Klassen von Blattsprachen. Ein spezieller Fall, der in diesem Artikel offen gelassen wurde, ist die Frage, wie mächtig Blattautomaten mit kontextfreien Blattsprachen sind; mit anderen Worten, wie sich die Klasse $\mathrm{Leaf}^{\mathrm{FA}}(\mathrm{CFL})$ zu anderen, bekannten Klassen verhält. Die einzigen bekannten oberen und unteren Schranken waren $\mathrm{CFL} \subsetneq \mathrm{Leaf}^{\mathrm{FA}}(\mathrm{CFL}) \subseteq \mathrm{DSPACE}(n^2) \cap \mathrm{DTIME}\big(2^{O(n)}\big)$. Wir können nun die untere Schranke verbessern:

**Korollar 4.5.2.** LOGCFL $\subseteq \mathrm{Leaf}^{FA}(\mathrm{CFL})$.

*Beweis.* Satz 4.5.1 liefert uns LOGCFL $\subseteq$ mon-$Q^1_{\mathrm{Grp}}\mathrm{FO} =$ mon-$Q^1_{\mathrm{CFL}}\mathrm{FO}$. Satz 4.3.2 sagt uns aber, dass die letztere Klasse mit $\mathrm{Leaf}^{\mathrm{FA}}(\mathrm{CFL})$ identisch ist. Beides zusammen ergibt LOGCFL $\subseteq \mathrm{Leaf}^{\mathrm{FA}}(\mathrm{CFL})$. □

## 4.6 Offene Fragen

Es ist uns noch immer nicht möglich, eine vollständige Einordnung der Klasse $\mathrm{Leaf}^{\mathrm{FA}}(\mathrm{CFL})$ vorzunehmen. Ist die Inklusion LOGCFL $= Q_{\mathrm{Grp}}\mathrm{FO} \subseteq$ mon-$Q^1_{\mathrm{Grp}}\mathrm{FO} = \mathrm{Leaf}^{\mathrm{FA}}(\mathrm{CFL})$ echt, sind also Lindströmquantoren zweiter Stufe in diesem Zusammenhang mächtiger als Quantoren erster Stufe? Anders betrachtet: Es ist relativ einfach zu sehen, dass jede Sprache der Klasse $\mathrm{Leaf}^{\mathrm{FA}}(\mathrm{CFL})$ durch Kellerautomaten mit linear großem Hilfsband und exponentieller Zeitbeschränkung akzeptiert werden kann. Das heißt

$$\begin{aligned} \mathrm{LOGCFL} &= \mathrm{NAuxPDA\text{-}SPACE\text{-}TIME}(\log n, n^{O(1)}) \\ &\subseteq \mathrm{Leaf}^{\mathrm{FA}}(\mathrm{CFL}) \\ &\subseteq \mathrm{NAuxPDA\text{-}SPACE\text{-}TIME}(n, 2^{O(n)}). \end{aligned}$$

Ist eine dieser Inklusionen vielleicht eine Gleichheit?

Wir haben gesehen, dass es sehr einfache reguläre Sprachen $L$ gibt, für die $Q^1_L\mathrm{FO} = \mathrm{REG}$. Da wären z.B. $L_\exists$ oder die „deterministische“ Sprache $L_{\mathrm{det}} = 1(0+1)^*$ (die als Blattsprache über TM die Klasse P beschreibt). Gibt es vielleicht noch restriktivere Quantoren, mit deren Hilfe man interessante Unterklassen von REG beschreiben kann?

# Kapitel 5

# Blattfunktionen

## 5.1 Einführung

Bisher haben wir uns ausschließlich mit Komplexitätsklassen von Mengen beschäftigt. Ein Gebiet der Komplexitätstheorie sind aber auch Komplexitätsklassen von Funktionen. Will man den Ressourcenverbrauch bei der Berechnung einer Funktion messen, muss man zuerst ein Berechnungsmodell festlegen. In unserem Fall sind dies TM, die ein Wort über einem Alphabet ausgeben, die *Transducer*.

Jede deterministische Komplexitätsklasse von Mengen besitzt ein Pendant auf der Funktionenseite. So ist z.B. FP, die Klasse der in Polynomialzeit von einem deterministischen Transducer berechenbaren Funktionen, die Entsprechung zur Klasse P der in Polynomialzeit entscheidbaren Mengen und FPSPACE, die Klasse der in polynomiellem Platz von einem deterministischen Transducer berechenbaren Funktionen die Entsprechung zur Klasse PSPACE der in polynomiellem Platz entscheidbaren Mengen.

Da ein nichtdeterministischer Transducer auf seinen verschiedenen Berechnungspfaden verschiedene Ausgaben berechnen kann, muss eine Methode gefunden werden, wie man daraus einen eindeutigen Funktionswert erzeugen soll. Es gibt mehrere Möglichkeiten, einen nichtdeterministischen Transducer einen Funktionswert berechnen zu lassen. Es ist z.B. möglich, die Anzahl der verschiedenen Ergebnisse zu zählen; so erhält man die Klasse span-P. Man kann auch nur die Pfade zählen, auf denen ein von 0 verschiedenes Ergebnis herauskommt; so erhält man die Klasse #P.

In [KSV00] wurde ein Methode eingeführt, um eine einheitliche Beschreibung aller Klassen, die mit nichtdeterministischen Transducern erzeugt werden können, zu ermöglichen. Die Methode der *Blattfunktionen* ist inspiriert von der in [BCS92] eingeführten Methode der Blattsprachen. Das Prinzip dieser Methode lautet: Ein Transducer generiert auf einer Eingabe einen Vektor

von Werten. Auf diesen Vektor wird nun eine Funktion – die Blattfunktion – angewandt und der Funktionswert ist das Ergebnis der Berechnung.

Ein Resultat aus [HLS+93] ist die Tatsache, dass die Klasse PSPACE durch eine reguläre Blattsprache beschrieben werden kann. Dies ist insofern bemerkenswert als PSPACE eine platzbeschränkte Klasse ist, während alle Blattsprachenklassen ja zeitbeschränkte Klassen sind. In diesem Kapitel soll für die Funktionenklasse FPSPACE ein ähnliches Ergebnis erzielt werden wie für PSPACE. Es wird gezeigt, dass es eine „einfache“ Blattfunktion gibt, die FPSPACE beschreibt, nämlich die Matrixmultiplikation. Wenn man einer Funktion $f$ die Blattsprachenklasse $f$-FP zuordnet (siehe 5.2.1), dann heißt die obige Aussage genauer:

**Satz 5.1.1.** *Sei $\mathfrak{F}$ eine Funktion, die als Eingabe eine Folge von $3 \times 3$-Matrizen mit Einträgen aus $\{-1, 0, 1\}$ erhält und den linken oberen Eintrag des Produktes dieser Matrizen ausgibt. Dann gilt $\mathfrak{F}$-FP = FPSPACE.*

Während wir diesen Satz beweisen, werden wir noch eine Vielzahl anderer Beschreibungen für FPSPACE finden, unter anderem durch Straight-Line-Programme und Matrixprogramme.

Es gibt noch andere Beschreibungen von FPSPACE durch einfache Blattfunktionen, wie die folgende von Wagner mit einem endlichen Automaten [Wag02].

**Satz 5.1.2.** *Es gibt einen endlichen Automaten $M$ mit Ausgabe, so dass für die von $M$ berechnete Funktion $f_M$ gilt: $f_M$-FP = FPSPACE.*

*Beweis.* Sei $f \in$ FPSPACE. Dann ist die Menge $\{ (x, i) \mid f(x)_i = 1 \}$ in PSPACE. Es existiert eine NPTM $M$, die mit einer bestimmten regulären Blattsprache $B$ die Menge akzeptiert. Nun können wir eine NPTM $M'$ definieren, die wie folgt arbeitet: Auf Eingabe $x$ verzweigt $M'$ sich zuerst in so viele Pfade, dass die Zahl der Pfade auf jeden Fall mindestens so hoch ist wie $|f(x)|$. Jeder Pfad entspricht einem $i$. Auf dem zu $i$ gehörenden Pfad wird dann $M$ auf Eingabe $(x, i)$ simuliert. Außerdem muss $M'$ noch sicherstellen, dass alle Blocks für verschiedene Werte von $i$ durch ein spezielles Symbol # auf dem Blattwort getrennt sind. Sei nun $M''$ ein endlicher Automat, der als Eingabe ein Blattwort von $M'$ erhält und wie folgt arbeitet: Während $M''$ die Symbole eines Blocks für ein $i$ liest, simuliert er einen weiteren endlichen Automaten $M'''$, der $B$ akzeptiert. Sobald $M''$ auf ein durch # markiertes Blockende trifft, gibt er 1 aus genau dann, wenn sich $M'''$ in einem Endzustand befindet und 0, wenn nicht. Beim nächsten Block beginnt $M''$ dann erneut die Simulation von $M'''$ vom Startzustand aus. Also gibt $M''$ nur an den Trennsymbolen # ein Zeichen aus und diese Zeichen ergeben zusammen genau $f(x)$. □

Leider hilft diese Beschreibung nicht dabei, die innere Struktur der Klasse FPSPACE deutlich zu machen.

## 5.2 Grundlagen

### 5.2.1 Komplexitätsklassen von Funktionen

Wir halten ein Alphabet $\Sigma = \{0, 1\}$ fest. Deterministische TM berechnen Funktionen $f\colon \Sigma^* \to \Sigma^*$. Wir benutzen eine der bekannten Bijektionen zwischen $\Sigma^*$ und den natürlichen Zahlen $\mathbb{N}$ bzw. den ganzen Zahlen $\mathbb{Z}$ um TM Funktionen $f\colon \Sigma^* \to \mathbb{N}$ bzw. $f\colon \Sigma^* \to \mathbb{Z}$ berechnen zu lassen. Die Klasse aller Funktionen $f\colon \Sigma^* \to \mathbb{Z}$, die von TM mit polynomiellem Platzbedarf berechnet werden können, nennen wir FPSPACE. In diesem Fall trägt die Anzahl der Zellen, die auf dem Ausgabeband verbraucht werden, nicht zum Platzbedarf der Maschine bei; deshalb darf der Ausgabewert einer FPSPACE-Funktion exponentiell lang im Vergleich zur Eingabe sein. Die Klasse aller Funktionen $f\colon \Sigma^* \to \mathbb{N}$, die von PSPACE-TM berechnet werden können, nennen wir $\text{FPSPACE}_+$.

Wir werden einige Zählfunktionen benutzen: Für eine TM $M$ und Eingabe $x$ definieren wir $acc(M, x)$ als die Anzahl akzeptierender und $rej(M, x)$ als die Anzahl ablehnender Berechnungspfade von $M$ auf $x$. Wir definieren #PSPACE als die Klasse von Funktionen $f\colon \Sigma^* \to \mathbb{N}$, für die eine NPSPACE-TM existiert, so dass für alle $x$ gilt $f(x) = acc(M, x)$. Außerdem definieren wir Gap-PSPACE als die Klasse aller Funktionen $f\colon \Sigma^* \to \mathbb{Z}$, für die eine NPSPACE-TM existiert, so dass für alle $x$ gilt $f(x) = acc(M, x) - rej(M, x)$.

Sei $M$ eine APTIME-TM. Ein *akzeptierender Berechnungsbaum* (oder *Beweisbaum*) [VT89] auf Eingabe $x$ ist ein Teilbaum $T'$ des Berechnungsbaumes $T_{M(x)}$ von $M$ auf $x$, dessen Wurzel mit der von $T_{M(x)}$ übereinstimmt, dessen Blätter akzeptierende Blätter von $T_{M(x)}$ sind und dessen Knoten folgende Bedingungen erfüllen:

- Jeder existenzielle Knoten in $T'$ hat genau einen Nachfolger in $T'$.
- Jeder universelle Knoten in $T'$ hat alle Knoten, die Nachfolger in $T_{M(x)}$ sind, auch als Nachfolger in $T'$.

Dann definieren wir $\#(M, x)$ als die Anzahl unterschiedlicher akzeptierender Berechnungsbäume von $M$ auf Eingabe $x$ und wir definieren #APTIME als die Klasse aller Funktionen $f\colon \Sigma^* \to \mathbb{N}$, für die es eine APTIME-TM $M$ gibt, so dass für alle $x$ gilt $f(x) = \#(M, x)$.

**Proposition 5.2.1 [Lad89].** $\mathrm{FPSPACE}_+ = \#\mathrm{PSPACE} = \#\mathrm{APTIME}$.

In [KSV00] wurde ein allgemeines Konstrukt zur Beschreibung von Funktionenklassen vorgestellt, welches starke Ähnlichkeit mit dem Prinzip der Blattsprachen besitzt. Wir werden eine leicht vereinfachte Version benutzen, die für unsere Zwecke ausreicht:

Sei $\Gamma$ ein Alphabet und sei $\Omega$ die Menge aller endlichen Vektoren (Folgen) von Elementen aus $\Gamma$. Eine *Blattfunktion* ist eine Funktion $F\colon \Omega \to \mathbb{Z}$, die diese Vektoren *auswertet*. (In [KSV00] wurde eine etwas allgemeinere Form von Blattfunktionen eingeführt, die dort *Generators* genannt wurde.) Jede Blattfunktion $F$ definiert eine Klasse $F$-FP. Dies ist die Klasse aller Funktionen $f\colon \Sigma^* \to \mathbb{Z}$, für die Polynomialzeit-berechenbare Funktionen $g\colon \Sigma^* \times \mathbb{N} \to \Gamma$ und $h\colon \Sigma^* \to \mathbb{N}$ existieren, so dass für alle $x \in \mathbb{N}$ die Folge $S_x = (g(x,0), g(x,1), \ldots, g(x,h(x)))$ zu $\Omega$ gehört und außerdem gilt $f(x) = F(S_x)$. Man denke bei $g(x,i)$ an die Ausgabe auf Pfad $i$ einer nichtdeterministischen TM auf Eingabe $x$ und bei $S_x$ an die Folge der Ausgaben der TM, also das Blattwort von $M(x)$, das von der Blattfunktion $F$ ausgewertet wird.

Seien $\mathcal{F}_1$ und $\mathcal{F}_2$ zwei Funktionenklassen. Wir definieren $\mathcal{F}_1 - \mathcal{F}_2$ als die Klasse der Differenzen von Funktionen aus $\mathcal{F}_1$ und $\mathcal{F}_2$ (nicht zu verwechseln mit $\mathcal{F}_1 \setminus \mathcal{F}_2$). D.h. also $\mathcal{F}_1 - \mathcal{F}_2 =_{\text{def}} \{ f \mid$ es gibt $f_1 \in \mathcal{F}_1$ und $f_2 \in \mathcal{F}_2$ mit $f(x) = f_1(x) - f_2(x)$ für alle $x \}$.

### 5.2.2 Nichtuniforme Berechnungsmodelle

#### Arithmetische Schaltkreise

Arithmetische Schaltkreise über den natürlichen Zahlen besitzen Eingabegatter mit Werten aus $\{0,1\}$ und konstante Gatter 0, 1 und $-1$. Ihre inneren Knoten sind Gatter, die Addition und Multiplikation berechnen. Ein arithmetischer Schaltkreis mit $n$ Eingabegattern berechnet eine Funktion $f\colon \{0,1\}^n \to \mathbb{Z}$ auf die offensichtliche Weise.

Sei $\overline{C} = (C_n)_{n\in\mathbb{N}}$ eine Familie arithmetischer Schaltkreise, wobei $C_n$ $n$ Eingabegatter besitzt und seien $s, d\colon \mathbb{N} \to \mathbb{N}$ Funktionen. Wir sagen, dass $\overline{C}$ die *Größe* $s$ (*Tiefe* $d$) besitzt, falls für jedes $n \in \mathbb{N}$ die Anzahl der Gatter in $C_n$ nicht größer ist als $s(n)$ (die Länge des längsten Pfades in $C_n$ nicht größer ist als $d(n)$).

Seien $F\colon \{0,1\}^* \to \mathbb{Z}$ und $s, d\colon \mathbb{N} \to \mathbb{N}$ Funktionen. Wir sagen $F \in$ AC-SIZE-DEPTH$(s,d)$, falls eine Familie $\overline{C}$ arithmetischer Schaltkreise der Größe $s$ und Tiefe $d$ existiert, so dass für alle $x \in \{0,1\}^*$ gilt $f(x) = C_{|x|}(x)$.

### Straight-Line-Programme

Ein $n$-Eingaben-*Straight-Line-Programm*, das $m$ Register benutzt, ist nach [BOC92] (wobei Ben-Or und Cleve diese spezielle Form von Programmen als *Linear-Bijection-Straight-Line-Programs* bezeichneten) eine Folge $SP = (s_t)_{1 \leq t \leq l}$ von Instruktionen, wobei jede Instruktion $s_t$ eine der folgenden Formen besitzt:

- $R_j \leftarrow R_j + c * R_i$,
- $R_j \leftarrow R_j - c * R_i$,
- $R_j \leftarrow R_j + x_k * R_i$, oder
- $R_j \leftarrow R_j - x_k * R_i$;

wobei $i, j \in \{1, \ldots, m\}$, $i \neq j$, $c \in \{0, 1\}$ und $k \in \{1, \ldots, n\}$. Wir nennen $l$ die *Länge* von $SP$.

Ein solches Programm berechnet eine Funktion $f_{SP} \colon \{0,1\}^n \to \mathbb{Z}$ wie folgt: Sei $x = x_1 \cdots x_n$. Zu Beginn der Berechnung hat das Register $R_1$ den Inhalt 1 und alle übrigen Register haben den Inhalt 0. Dann werden die Instruktionen $s_1; s_2; \ldots; s_l$ in dieser Reihenfolge ausgeführt; dabei ändern sie den Inhalt der Register in der natürlichen Weise. Schließlich ist der Inhalt des Registers $R_1$ das Ergebnis der Berechnung, also der Wert von $f_{SP}(x)$.

Sei $\overline{SP} = (SP_n)_{n \in \mathbb{N}}$ eine Familie von Straight-Line-Programmen und sei $s \colon \mathbb{N} \to \mathbb{N}$ eine Funktion. Wir sagen, dass $\overline{SP}$ die *Größe* $s$ besitzt, falls für jedes $n \in \mathbb{N}$ die Länge von $SP_n$ nicht größer als $s(n)$ ist. Seien $F \colon \{0,1\}^* \to \mathbb{Z}$ und $s, r \colon \mathbb{N} \to \mathbb{N}$ Funktionen. Wir sagen $F \in \text{SLP-SIZE-REG}(s, r)$, falls eine Familie $\overline{SP}$ von Straight-Line-Programmen der Größe $s$ existiert, so dass kein $SP_n$ mehr als $r(n)$ Register benutzt und für alle $x \in \mathbb{N}$ gilt $f(x) = SP_{|x|}(x)$.

### Matrixprogramme

Als $d$-dimensionales $n$-Eingaben-Matrixprogramm bezeichnen wir eine Folge $MP = (N_t)_{1 \leq t \leq l}$ von $d \times d$-Matrizen, deren Einträge Elemente der Menge $\{-1, 0, 1\} \cup \{x_1, \ldots, x_n, -x_1, \ldots, -x_n\}$ sind (siehe [CMTV98]). Wir nennen $l$ die *Länge* des Programmes.

Ein solches Programm berechnet eine Funktion $f_{MP} \colon \{0,1\}^n \to \mathbb{Z}$ wie folgt: Sei $x = x_1 \cdots x_n$. Das Ergebnis der Berechnung von $MP$ ist

$$f_{MP}(x) = (1\ 0\ \ldots\ 0) \cdot \left( \prod_{t=1}^{l} N_t \right) \cdot \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

in anderen Worten: $f_{MP}(x)$ ist der linke obere Eintrag der Matrix $\prod_{t=1}^{l} N_t$.

Sei $\overline{MP} = (MP_n)_{n\in\mathbb{N}}$ eine Familie von Matrixprogrammen und sei $s\colon \mathbb{N} \to \mathbb{N}$ eine Funktion. Wir sagen $\overline{MP}$ hat die *Größe* $s$, falls die Länge von $MP_n$ für kein $n \in \mathbb{N}$ größer als $s(n)$ ist. Seien $F\colon \{0,1\}^* \to \mathbb{Z}$ und $s, d\colon \mathbb{N} \to \mathbb{N}$ Funktionen. Wir sagen $F \in$ MP-SIZE-DIM$(s, d)$ falls eine Matrixprogrammfamilie $\overline{MP}$ der Größe $s$ existiert, so dass alle $MP_n$ die Dimension $d(n)$ besitzen und dass für alle $x \in \mathbb{N}$ gilt $f(x) = MP_{|x|}(x)$.

### Uniformität

Jedem Berechnungsbaum $T_{M(x)}$ einer alternierenden TM $M$ auf Eingabe $x$ kann ein arithmetischer Schaltkreis $C_{M(x)}$ auf folgende Weise zugeordnet werden:

1. Jeder existenzielle Knoten in $T_{M(x)}$ ist ein Additionsgatter in $C_{M(x)}$.

2. Jeder universelle Knoten in $T_{M(x)}$ ist ein Multiplikationsgatter in $C_{M(x)}$.

3. Zwei Knoten, die durch eine Kante in $T_{M(x)}$ verbunden sind, sind durch einen Draht in $C_{M(x)}$ verbunden.

4. Falls ein Pfad in $T_{M(x)}$ akzeptierend ist – was bedeutet, dass eine 1 auf dem Blatt ausgegeben wird –, so ist das korrespondierende Gatter in $C_{M(x)}$ die Konstante 1. Falls ein Pfad ablehnend ist, so ist das entsprechende Gatter die Konstante 0.

5. Die Wurzel von $T_{M(x)}$ ist das Ausgabegatter von $C_{M(x)}$.

Es ist nun einfach zu sehen, dass $\#(M, x) = C_{M(x)}$ gilt. Wir sagen dann, dass $M(x)$ den Schaltkreis $C_{M(x)}$ *beschreibt*. Man beachte, dass der Schaltkreis keine Eingabegatter besitzt.

Wir sagen, dass $M$ *schaltkreiserhaltend* ist, falls $M$ für jede Eingabe $x$ einen vollständigen binären Berechnungsbaum $T_{M(x)}$ erzeugt und falls sich die Berechnungsbäume für verschiedene Eingaben gleicher Länge höchstens an den Ausgabewerten der Blätter unterscheiden dürfen.

Um arithmetische Schaltkreise mit Eingabegattern beschreiben zu können, müssen wir alternierende Transducer betrachten. Ein alternierender Transducer $M$ auf Eingabe $x$ beschreibt einen arithmetischen Schaltkreis $C_{M(x)}$, falls $M$ nur Ausgaben aus der Menge $\{-1, 0, 1, 2, 3, 4, \ldots, |x| + 1\}$ auf seinen Blättern produziert. Der dadurch beschriebene Schaltkreis ist wie bei TM, mit dem folgenden Unterschied:

4. Falls auf einem Blatt von $T_{M(x)}$ die Ausgabe eine Zahl $2 \leq i \leq |x|+1$ ist, dann ist das entsprechende Gatter in $C_x$ ein Eingabegatter mit Beschriftung $x_{i-1}$. Falls die Ausgabe eine Zahl $-1 \leq i \leq 1$ ist, so ist das entsprechende Gatter eine Konstante $i$.

Der vom Schaltkreis auf Eingabe $x$ berechnete Wert ist $C_{M(x)}(x)$. Falls der Transducer $M$ für alle Eingaben der gleichen Länge identische Berechnungsbäume erzeugt (was eine stärkere Bedingung als „schaltkreiserhaltend“ ist), dann nennen wir die erzeugten arithmetischen Schaltkreise $C_{M(|x|)}$. Der Transducer beschreibt dann eine Schaltkreisfamilie $\overline{C_M}$.

Wir sagen nun, dass ein Schaltkreisfamilie $\overline{C}$ $\mathrm{U_{FBT}}$-*uniform* ist, falls sie in der oben beschriebenen Weise durch einen Polynomialzeit-Transducer beschrieben werden kann.

Eine Familie $\overline{SLP}$ von Straight-Line-Programmen ist $\mathrm{U_{FBT}}$-*uniform*, falls ein nichtdeterministischer Polynomialzeit-Transducer existiert, der nur vollständige binäre Berechnungsbäume erzeugt und der auf Eingabe $x$ als Blattwort das Straight-Line-Programm $SLP_{|x|}$ ausgibt, wobei die einzelnen Instruktionen auf den Blättern ausgegeben werden.

Die Definition für Matrixprogramme ist analog: Wir müssen einen nichtdeterministischen Polynomialzeit-Transducer finden, der auf Eingabe $x$ auf seinen Blättern die Matrizen von $MP_{|x|}$ ausgibt.

### Boolesche Schaltkreise

Für den Beweis von Lemma 5.3.1 benötigen wir den Begriff der Booleschen Schaltkreise. Eine ausführliche Abhandlung über dieses Thema findet sich z.B. in [Vol99a].

Boolesche Schaltkreise unterscheiden sich von den arithmetischen Schaltkreisen dadurch, dass sie statt der Additions- und Multiplikationsgatter $\wedge$- und $\vee$-Gatter besitzen. Wir betrachten nur Boolesche Schaltkreise mit einem einzigen Ausgabegatter. Ein Boolescher Schaltkreis $B_n$ mit $n$ Eingabegattern berechnet also eine Funktion $f_{B_n} : \{0,1\}^n \to \{0,1\}$. Eine Familie $\overline{B}$ Boolescher Schaltkreise berechnet also die charakteristische Funktion einer Menge, sie akzeptiert demnach diese Menge.

Wir sagen, dass eine Familie Boolescher Schaltkreise $\overline{B}$ von einem deterministischen Transducer $M$ *beschrieben* wird, falls dieser Transducer auf Eingabe $1^n$ eine Kodierung des Schaltkreises $B_n$ auf seinem Ausagabeband ausgibt.

Für eine Funktion $f$ bezeichnen wir mit $\mathrm{U_L}$-SIZE($f$) die Klasse aller Mengen $L$, für die eine Schaltkreisfamilie der Größe $f$ existiert, die $L$ akzeptiert und die von einem deterministischen Transducer $M$ mit Platzbedarf $\log(p)$ beschrieben wird.

## 5.3 Resultat

Wir benötigen zwei technische Lemmata für den Beweis unseres Hauptresultats:

**Lemma 5.3.1.** $\mathrm{P} \subseteq \mathrm{U}_{\mathrm{FBT}}\text{-AC-SIZE-DEPTH}(2^{n^{O(1)}}, n^{O(1)})$.

*Beweis.* Sei $L \in \mathrm{P}$. Wir wissen (siehe z.B. [Vol99a] Korollar 2.30.), dass $\mathrm{P} = \mathrm{U_L}\text{-SIZE}(n^{O(1)})$. D.h. also, dass es ein Polynom $p$ gibt und dass ein deterministischer Transducer $M$ mit Raumkomplexität $\log(p)$ existiert, der auf Eingabe $1^n$ auf seinem Ausgabeband einen binären Booleschen Schaltkreis $B_n$ der Größe (und damit natürlich auch Tiefe) $p(n)$ ausgibt. Dieser Schaltkreis akzeptiert genau die Wörter $x$ der Länge $n$, für die $x \in L$ gilt. Da $\log(p(n))$ die Raumkomplexität von $M$ ist, ist die Zeitkomplexität $2^{\log(p(n))} = p(n)$. Damit ist $M$ also ein Polynomialzeit-Transducer.

Wir wollen nun Booleschen Schaltkreise durch arithmetische Schaltkreise simulieren. Dabei hilft uns die Tatsache, dass für $x, y \in \{0, 1\}$ gilt: $\neg x = 1 + (-1) \cdot x$, $x \vee y = x \cdot (1 + (-1) \cdot y) + y$ und $x \wedge y = x \cdot y$. Also können wir jedes Boolesche Gatter durch einen binären arithmetischen Schaltkreis mit höchstens 4 Gattern ersetzen, der bei gleicher Eingabe das gleiche Ergebnis liefert. Bemnach können wir eine Familie $\overline{A}$ binärer arithmetischer Schaltkreise mit Größe und Tiefe $4 \cdot p(n)$ finden, die konstante Gatter 1 und $-1$ besitzen und für die gilt $A_{|x|}(x) = B_{|x|}(x)$ für alle $x \in \mathbb{N}$. Es gibt natürlich einen deterministischen Polynomialzeit-Transducer $M_A$, der $\overline{A}$ beschreibt. Wir müssen aber einen alternierenden Polynomialzeit-Transducer $M'$ finden, der eine Familie $\overline{C_{M'}}$ total binärer arithmetischer Schaltkreise produziert, für die $C_{M'(|x|)}(x) = A_{|x|}(x)$ gilt.

$M'(x)$ wird zuerst $M_A(1^{|x|})$ simulieren und $A_{|x|}$ in geeigneter Kodierung auf sein Arbeitsband schreiben. Dann wird $M'(x)$ einen Berechnungsbaum aufbauen, der $A_{|x|}$ ähnelt. Da aber $A_{|x|}$ als arithmetischer Schaltkreis kein Baum sein muss, $M'(x)$ aber nur einen Baum $C_{M'(x)}$ erzeugen kann, müssen einige Veränderungen vorgenommen werden:

Jeder Unterschaltkreis von $A_{|x|}$, dessen Resultat von mehr als einem Gatter benötigt wird, wird für jede Benutzung einmal repliziert. Diese Replikation startet an der Wurzel.

Der resultierende binäre arithmetische Schaltkreis $C_{M'(x)}$ kann durch die Replikationen von exponentieller Größe sein, aber seine Höhe ist identisch zu der von $A_{|x|}$; durch diesen Prozess ist aus einem azyklischen Graph ein Baum geworden.

Außerdem kann $C_{M'(x)}$ einfach aufgeblasen werden, um zu erreichen, dass er total balanciert ist.

Da die Höhe von $T_{M'}(|x|) = C_{M'(|x|)}$ den Wert $4 \cdot p(|x|)$ besitzt, ist seine Größe maximal $2^{4 \cdot p(|x|)}$. Da $C_{M'(|x|)}(x) = A_{|x|}(x) = B_{|x|}(x)$ gilt, wissen wir, dass $\overline{C_{M'}}$ die Sprache $L$ entscheidet. Also gilt $L \in \mathrm{U_{FBT}}$-AC-SIZE-DEPTH($2^{n^{O(1)}}, n^{O(1)}$). □

**Lemma 5.3.2.** *Sei $f \in$ #APTIME. Dann gibt es ein Polynom $r$, so dass für jedes Polynom $q$ mit $q \geq r$ eine schaltkreiserhaltende* APTIME*-TM $M_q$ mit Berechnungsbäumen der Tiefe genau $q(|x|)$ existiert, für die gilt: $f(x) = \#(M_q, x)$.*

*Beweis.* Sei $f \in$ #APTIME. Wir wissen, dass #PSPACE = #APTIME gilt. Also gibt es eine NPSPACE-TM $M_f$, die $f \in$ #PSPACE zeigt. Wir benutzen den Beweis von Proposition 5.2.1, um zu zeigen, dass eine von $M_f$ abhängige APTM $M'$ existiert mit $f(x) = \#(M', x)$ für alle $x \in \mathbb{N}$. Damit hätten wir nur #APTIME $\subseteq$ #PSPACE $\subseteq$ #APTIME gezeigt. Wir werden den Beweis jedoch ein wenig erweitern um gleichzeitig zu zeigen, dass man $M'$ auch so konstruieren kann, dass die TM total balancierte Berechnungsbäume erzeugt.

Sei $x$ eine Eingabe für $M_f$ die die Länge $n$ besitzt. Ohne die Anzahl akzeptierender Pfade von $M_f$ auf $x$ zu verändern können wir alle Berechnungen so aufblasen, dass sie – für ein Polynom $p$ – die Länge $2^{p(n)}$ besitzen. Außerdem können wir annehmen, dass es nur eine einzige akzeptierende Konfiguration gibt. Wir betrachten den folgenden alternierenden Algorithmus reach$(C, D, K)$, der genau dann akzeptiert, wenn Konfiguration $D$ aus Konfiguration $C$ in genau $2^k$ Schritten erreichbar ist.

```
function reach(C, D, k);
begin
  if k = 0
    then
      if D ist aus C in einem Schritt erreichbar
        then akzeptiere
        else lehne ab
    else
      ⋁ E[reach(C, E, k − 1) ∧ reach(E, D, k − 1)]
end
```

Mit der Bezeichnung $\bigvee E$ ist gemeint: „Wähle existenziell eine Konfiguration $E$“. Diese existenzielle Verzweigung kann mithilfe eines vollständigen Binärbaumes erzielt werden. Falls die Anzahl der Verzweigungen keine Potenz von 2 ist, können wir den Baum mithilfe von nicht-erreichbaren Konfigurationen aufblasen. Die Notation $\wedge$ ist ein binärer Operator, der bedeutet: „Wähle universell einen der beiden Nachfolger“. Also erzeugt ein Aufruf der

Funktion reach$(C, D, k)$ eine vollständig balancierten Berechnungsbaum. Der rekursive Aufruf „[reach$(C, E, k-1) \wedge$ reach$(E, D, k-1)]“ sorgt dafür, dass zwei vollständige Bäume derselben Größe an jedes Blatt des ersten Baumes gehängt werden. Dies setzt sich fort, bis $k = 0$ gilt und damit ist der gesamte Berechnungsbaum vollständig balanciert.

Da die Erzeugung des Berechnungsbaumes nur von der Eingabelänge – nicht von der Eingabe selbst – abhängt, sind alle Berechnungsbäume für Eingaben derselben Länge identisch.

Der alternierende Algorithmus, der $M_f$ auf Eingabe $x$ simuliert, ist der Aufruf reach$(\mathit{init}, \mathit{acc}, p(n))$, wobei *init* die Startkonfiguration von $M_f$ auf Eingabe $x$ und *acc* die eindeutige akzeptierende Konfiguration ist.

Da durch Induktion über $k$ gezeigt werden kann, dass die Anzahl der Berechnungspfade von $C$ nach $D$ der Länge $2^k$ identisch zur Anzahl der akzeptierenden Berechnungsbäume von reach$(C, D, k)$ ist, ist auch die Anzahl der akzeptierenden Berechnungen von $M_f$ auf Eingabe $x$ identisch zur Anzahl akzeptierender Berechnungsbäume von reach$(\mathit{init}, \mathit{acc}, p(n))$.

Also wird reach$(\mathit{init}, \mathit{acc}, p(n))$ von $M'(x)$ berechnet. Sei $h\colon \mathbb{N} \to \mathbb{N}$ die Funktion, die die Höhe des Berechnungsbaumes in Abhängigkeit der Eingabelänge berechnet. Wir können ein Polynom finden, das eine obere Schranke für $h$ bildet. Dieses Polynom wird das von uns gesuchte $r$. Für ein Polynom $q$ mit $q \geq r$ wird die TM $M_q$ wie folgt arbeiten: Auf Eingabe $x$ wird zuerst $q(|x|)$ brechnet. Dann wird $M'(x)$ simuliert, wobei die Höhe des Berechnungsbaumes mitgezählt wird. Hat $M'(x)$ auf Höhe $h(|x|)$ die Berechnung beendet, so wird der Berechnungsbaum um $q(|x|) - h(|x|)$ Höhenstufen aufgeblasen ohne die Anzahl der akzeptierenden Pfade zu verändern. □

Unser Hauptresultat liefert nun verschiedene Charakterisierungen von FPSPACE mithilfe von Zählklassen, arithmetischen Schaltkreisen, Straight-Line-Programmen, Matrixprogrammen und Blattsprachen. Die Aussage des Satzes 5.1.1 ist dabei die Gleichheit (FPSPACE = $\mathfrak{F}$-FP) des folgenden Satzes.

**Satz 5.3.3.** *Sei* $\Sigma$ *die Menge aller* $3 \times 3$ *Matrizen über* $\{-1, 0, 1\}$ *und* $\mathfrak{F}\colon \Sigma^* \to \mathbb{Z}$ *mit*

$$\mathfrak{F}(N_1, \ldots, N_n) =_{\text{def}} \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \cdot \left( \prod_{i=1}^{n} N_i \right) \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}.$$

*Dann sind die folgenden Komplexitätsklassen identisch:*

$K_1$ = FPSPACE

$K_2$ = Gap-PSPACE

$K_3$ = #PSPACE − #PSPACE

$K_4$ = #APTIME − #APTIME

$K_5$ = $\mathrm{U}_{\mathrm{FBT}}$-AC-SIZE-DEPTH$(2^{n^{O(1)}}, n^{O(1)})$

$K_6$ = $\mathrm{U}_{\mathrm{FBT}}$-SLP-SIZE-REG$(2^{n^{O(1)}}, 3)$

$K_7$ = $\mathrm{U}_{\mathrm{FBT}}$-MP-SIZE-DIM$(2^{n^{O(1)}}, 3)$

$K_8$ = $\mathfrak{F}$-FP

*Beweis.* Wir zeigen, dass der folgende Inklusionskreis gilt: $K_1 \subseteq K_3 \subseteq K_4 \subseteq K_5 \subseteq K_6 \subseteq K_7 \subseteq K_8 \subseteq K_1$. Außerdem zeigen wir noch, dass $K_2 = K_3$ gilt. Damit ist die Aussage bewiesen.

$K_1 \subseteq K_3$

Sei $f \in$ FPSPACE. Wir müssen nun zwei NPTM $M_1, M_2$ finden, so dass für alle $x \in \mathbb{N}$ gilt $f(x) = \#_p(M_1, x) - \#_p(M_2, x)$.

Sei $M$ ein PSPACE-Transducer, der $f$ berechnet. Daraus können wir auf einfache Weise zwei PSPACE-Transducer $M_f^+$ and $M_f^-$ konstruieren die $M_f$ simulieren und $\max\{f(x), 0\}$ bzw. $\max\{-f(x), 0\}$ berechnen. Beide Transducer berechnen Funktionen aus $\mathrm{FPSPACE}_+$ und es gilt $f(x) = M_f^+(x) - M_f^-(x)$.

Das bedeutet FPSPACE $\subseteq \mathrm{FPSPACE}_+ - \mathrm{FPSPACE}_+$. Wir wissen aber bereits, dass $\mathrm{FPSPACE}_+$ = #PSPACE. Und daraus folgt dann FPSPACE $\subseteq$ #PSPACE − #PSPACE.

$K_2 = K_3$

"$\supseteq$": Sei $f \in$ Gap-PSPACE und sei $M_f$ eine NPSPACE-TM, die zeigt, dass $f \in$ Gap-PSPACE. Dann gilt $f(x) = acc(M_f, x) - rej(M_f, x)$.

Sei $M_f^+$ eine NPSPACE-TM, die identisch zu $M_f$ ist. Dann gilt natürlich $acc(M_f^+, x) = acc(M_f, x)$. Wir konstruieren nun eine weitere NPSPACE-TM $M_f^-$ wie folgt: $M_f^-(x)$ simuliert $M_f(x)$ und invertiert auf jedem Pfad das Akzeptanzverhalten von $M_f(x)$. Dies bedeutet $acc(M_f^-, x) = rej(M_f, x)$.

Insgesamt gilt $f(x) = acc(M_f^+, x) - acc(M_f^-, x)$ für alle $x$, woraus $f \in$ #PSPACE − #PSPACE folgt.

"$\subseteq$": Sei $f \in$ #PSPACE − #PSPACE. Dann gibt es zwei NPSPACE-TM $M_1$ und $M_2$ mit $f = acc(M_1, x) - acc(M_2, x)$. Wir konstruieren eine

NPSPACE-TM $M_f$, die auf Eingabe $x$ folgendermaßen arbeitet: Sie beginnt mit einer binären Verzweigung. Auf dem ersten Pfad wird dann $M_1(x)$ simuliert. Falls $M_1$ auf einem Pfad akzeptiert, so akzeptiert auch $M_f$. Falls $M_1$ ablehnt, so zweigt sich $M$ in zwei Pfade auf. Auf dem ersten akzeptiert $M_f$ und auf dem zweiten wird abgelehnt.

Auf dem zweiten Pfad der Verweigung zu Beginn der Berechnung wird $M_2(x)$ simuliert. Falls $M_2$ akzeptiert, so verzweigt sich $M_f$ in zwei Pfade; auf dem ersten wird akzeptiert und auf dem zweiten wird abgelehnt. Falls $M_2$ ablehnt, so lehnt auch $M_f$ ab.

Offensichtlich gilt $acc(M_f, x) - rej(M_f, x) = acc(M_1, x) - acc(M_2, x)$ für alle $x$. Damit gilt aber $f \in$ Gap-PSPACE.

$K_3 = K_4$

Dies folgt direkt aus Proposition 5.2.1.

$K_4 \subseteq K_5$

Sei $f \in$ #APTIME $-$ #APTIME. Also existieren zwei APTIME-TM $M_1$ und $M_2$ mit $f(x) = \#(M_1, x) - \#(M_2, x)$ für alle $x \in \Sigma^*$. Wir müssen nun einen alternierenden Polynomialzeit-Transducer $M$ finden, der auf jede Eingabe $x$ einen vollständigen binären Berechnungsbaum produziert, der einen arithmetischen Schaltkreis $C_{|x|}$ (nur abhängig von der Eingabelänge) mit $C_{|x|}(x) = \#(M_1, x) - \#(M_2, x)$ beschreibt.

Aufgrund von Lemma 5.3.2 wissen wir, dass zwei schaltkreiserhaltende APTIME-TM $M_1'$ und $M_2'$ existieren, die beide Berechnungsbäume der Höhe $q(|x|)$ für ein Polynom $q$ erzeugen und für die gilt $\#(M_1, x) = \#(M_1', x)$ und $\#(M_2, x) = \#(M_2', x)$. D.h. also, dass für unseren gesuchten Transducer $M$ ausreichend ist, wenn $C_{|x|}(x) = \#(M_1', x) - \#(M_2', x)$ gilt.

Die Berechnungsbäume $T_{M_1'(x)}$ und $T_{M_2'(x)}$ können auch als arithmetische Schaltkreise mit Ausgabe $\#(M_1', x)$ bzw. $\#(M_2', x)$ betrachtet werden. Wir konstruieren nun einen neuen schaltkreiserhaltenden Transducer $M'$, der auf Eingabe $x$ immer konstante arithmetische Schaltkreise der Höhe $q(|x|) + 2$ mit $C_{M'(x)} = \#(M_1', x) - \#(M_2', x)$ erzeugt:

$M'(x)$ verzweigt sich zuerst existenziell. Auf beiden Pfaden verzweigt $M'(x)$ sich dann universell. Nun gibt es vier Pfade. Auf Pfad 1 wird $M_1'(x)$ simuliert; auf Pfad 2 wird ein komplett balancierter Berechnungsbaum $S_1$ der Höhe $q(|x|)$ erzeugt, der, als arithmetischer Schaltkreis $C_{S_1}$ ohne Eingabe interpretiert, den Wert 1 ausgibt. Auf Pfad 3 wird ein komplett balancierter Berechnungsbaum $S_2$ der Höhe $q(|x|)$ erzeugt, der, als arithmetischer Schaltkreis $C_{S_2}$ ohne Eingabe interpretiert, den Wert $-1$ ausgibt. Auf Pfad 4 wird schließ-

lich $M'_2(x)$ simuliert. Offensichtlich gilt $C_{M'(x)} = (C_{M'_1(x)} * 1) + (-1 * C_{M'_2(x)}) = \#(M'_1, x) - \#(M'_2, x)$.

Nun können wir unseren gesuchten Transducer $M$ konstruieren: $M'$ ist schaltkreiserhaltend. Das heißt, dass seine Berechnungsbäume – mit der Ausnahme der Blätter selbst, die von der Eingabe abhängen – nur von der Eingabelänge abhängen. Die einzigen Blätter, die aber tatsächlich von der Eingabe abhängen sind die der Teilbäume $T_{M'_1(x)}$ und $T_{M'_2(x)}$. Jedes dieser Blätter kann in deterministischer Polynomialzeit berechnet werden. Demnach können wir einen Polynomialzeit-Transducer $M'_p$ finden, der auf Eingabe $(x, n)$ mit $1 \leq n \leq 2^{q(|x|)+2}$ die Ausgabe des Pfades $n$ von $M'(x)$ berechnet. Dann können wir aber aufgrund von Lemma 5.3.1 einen Polynomialzeit-Transducer $M'_c$ finden, der auf Eingabe $(x, n)$ einen arithmetischen Schaltkreis erzeugt, der $M'_p(x, n)$ berechnet.

Der Transducer $M$ simuliert nun auf Eingabe $x$ den Transducer $M'(x)$ und simuliert dann, nachdem er Pfad $n$ dessen Berechnungsbaumes beendet hat, $M'_c(x, n)$. Durch diese Simulation wird das Blatt $n$ von $C_{M'(x)}$ durch einen weiteren arithmetischen Schaltkreis ersetzt. Dieser Schaltkreis hat allerdings die virtuelle Eingabe $(x, n)$. Falls eines seiner Eingabeblätter sich auf ein Bit von $n$ bezieht, dann muss $M(x)$ dieses Eingabegatter durch ein konstantes Gatter ersetzten, das den gleichen Wert besitzt wie dieses Bit von $n$. Danach müssen noch alle Bäume der $M'_p(x, n)$ zu gleicher Größe „aufgebläht“ werden, ohne die Berechnung zu verändern.

Der resultierende Berechnungsbaum $T_{M(x)}$ hängt nur von der Eingabelänge ab und ist ein vollständiger Binärbaum. Die Ausgaben an seinen Blättern sind alle aus der Menge $\{-1, 0, 1, 2, 3, \ldots, |x|+1\}$ und der arithmetische Schaltkreis, der von ihm beschrieben wird, hat die Eigenschaft $C_{|x|}(x) = \#(M'_1, x) - \#(M'_2, x)$.

Damit haben wir $f \in \mathrm{U_{FBT}}\text{-AC-SIZE-DEPTH}(2^{n^{O(1)}}, n^{O(1)})$ gezeigt.

### $K_5 \subseteq K_6$

Sei $f \in \mathrm{U_{FBT}}\text{-AC-SIZE-DEPTH}(2^{n^{O(1)}}, n^{O(1)})$. Dann existiert ein alternierender Polynomialzeit-Transducer $M$, der nur vollständige Berechnungsbäume produziert für die gilt $C_{M(|x|)}(x) = f(x)$.

Es ist seit [BOC92] bekannt, dass die Berechnung eines arithmetischen Schaltkreises über den ganzen Zahlen durch ein Straight-Line-Programm mit nur drei Registern simuliert werden kann (siehe dazu auch [Vol99a], Theorem 5.15). Sei $P_{|x|}$ das Straight-Line-Programm, das den Schaltkreis $C_{M(|x|)}$ simuliert. Wir werden nun zeigen, dass es einen Polynomialzeit-Transducer $M_S$ gibt, der auf Eingabe $x$ auf dem Blatt jedes seiner Berechnungspfade eine Instruktion von $P_{|x|}$ ausgibt, so dass das gesamte Blattwort das gesamte

Programm in korrekter Reihenfolge darstellt. Wir sagen dann, dass $M_S(x)$ das Straight-Line-Programm $P_{|x|}$ *beschreibt*.

Dabei bedienen wir uns des Beweises von Theorem 5.15 in [Vol99a]. Dort wird, nach Ben-Or and Cleve [BOC92], der Begriff des *Offsets* eingeführt: Ein Straight-Line-Programm mit Registern $R_1, \ldots, R_k$ erzeugt einen Offset $o$ in Register $R_i$, $1 \leq i \leq k$, falls bei beliebigem Registerinhalt in allen Registern vor Start des Programms, wobei $R_i$ den Inhalt $x$ besitzt, sich der Inhalt aller Register mit Ausnahme von $R_i$ nach Ausführung des Programms nicht geändert hat und $R_i$ den Inhalt $x + o$ hat.

Sei $C$ ein Gatter eines arithmetischen Schaltkreises und seien $i, j, k \in \{1, 2, 3\}$ mit $\{i, j, k\} = \{1, 2, 3\}$ die Register. Wir wollen ein Straight-Line-Programm finden, das in Register $R_j$ einen Offset von $R_i * f_C$ erzeugt und ein Programm, das in Register $R_j$ einen Offset von $-R_i * f_C$ erzeugt. Der oben genannte Beweis zeigt, das dies möglich ist. Die Programme haben folgende Form:

Falls $C$ ein Eingabegatter ist, so lauten die Programme $R_j \leftarrow R_j + f_C * R_i$ bzw. $R_j \leftarrow R_j - f_C * R_i$. Falls $C$ die direkten Vorgänger $D$ und $E$ besitzt und falls gilt $f_C = f_D + f_E$ bzw. $f_C = f_D * f_E$, dann können wir die Programme für $f_C$ unter der Annahme, dass schon Straight-Line-Programme für $f_D$ und $f_E$ existieren, konstruieren:

**Existentielles Gatter**

Offset von $R_i * f_C$ in $R_j$:

1. Offset von $R_i * f_D$ in $R_j$;
2. Offset von $R_i * f_E$ in $R_j$;
3. Offset von $R_i * f_E$ in $R_j$;
4. Offset von $-R_i * f_E$ in $R_j$.

Offset von $-R_i * f_C$ in $R_j$:

1. Offset von $-R_i * f_D$ in $R_j$;
2. Offset von $-R_i * f_E$ in $R_j$;
3. Offset von $R_i * f_E$ in $R_j$;
4. Offset von $-R_i * f_E$ in $R_j$.

In beiden Programmen neutralisiert das vierte Unterprogramm den Effekt des dritten Unterprogramms. Die Programme würden also dasselbe leisten, wenn die beiden letzten Unterprogramme fehlen würden. Diese sind nur Padding, um die Gesamtlänge der Programme auf vier zu erhöhen.

**Universelles Gatter**

Offset von $R_i * f_C$ in $R_j$:

1. Offset von $-R_k * f_E$ in $R_j$;
2. Offset von $R_i * f_D$ in $R_k$;
3. Offset von $R_k * f_E$ in $R_j$;
4. Offset von $-R_i * f_D$ in $R_k$.

Offset von $-R_i * f_C$ in $R_j$:

1. Offset von $R_k * f_E$ in $R_j$;
2. Offset von $R_i * f_D$ in $R_k$;
3. Offset von $-R_k * f_E$ in $R_j$;
4. Offset von $-R_i * f_D$ in $R_k$.

Offensichtlich benutzt jedes Programm genau 4 Unterprogramme und in jedem Unterprogramm wird nur das Resultat eines einzigen arithmetischen Gatters benutzt. Für einen vollständig balancierten arithmetischen Schaltkreis der Tiefe $d$ wird das resultierende Straight-Line-Programm aus genau $4^d$ Instruktionen bestehen. Da $C_{|x|}$ vollständig balanciert ist, gilt die Eigenschaft auch für diesen Schaltkreis.

Wir verwenden im Weiteren den Begriff der *Konfigurationen*, der benutzt wird, um die Situation eines Transducers zu einem bestimmten Zeitpunkt seiner Berechnung zu beschreiben. Die Informationen, die benötigt werden, um eine Konfiguration eines Transducers zu beschreiben, sind die Inhalte der Bänder, die Positionen der Köpfe und der Zustand des Transducers. Für einen Transducer $N$ und Konfiguration $c$ nennen wir die Berechnung von $N$, die bei Konfiguration $c$ startet, $N(c)$. Sie erzeugt den Berechnungsbaum $T_{N(c)}$.

Sei $c$ eine Konfiguration, die von $M$ auf Eingabe $x$ erreicht wird und sei der Zustand von $c$ universell, existentiell oder ein Endzustand. Dann erzeugt $M(c)$ den Berechnungsbaum $T_{M(c)}$, der auf schon bekannte Weise als arithmetischer Schaltkreis aufgefasst werden kann. Sei $G(c)$ das Gatter an der Wurzel von $T_{M(c)}$. Wir werden nun durch Induktion über die Höhe $h$ von $T_{M(c)}$ zeigen, dass ein nichtdeterministischer Polynomialzeit-Transducer $M'_s$ existiert, der für $i \neq j$ sowohl das Straight-Line-Programm, das den Offset $R_j * f_{G(x,c)}$ in Register $R_i$ erzeugt, als auch das Straight-Line-Programm, das $-R_j * f_{G(x,c)}$ in Register $R_i$ erzeugt, beschreiben kann.

$h = 0$: Die Wurzel von $G(c)$ ist das einzige Gatter von $T_{M(c)}$. Dies bedeutet, dass $M$ bei Konfiguration $c$ einen Endzustand erreicht hat. Also gibt $M(c)$ direkt einen Wert $f_{G(c)} \in \{-1, 0, 1, 2, \ldots, |x| + 1\}$ aus. Falls das zu beschreibende Straight-Line-Programm „Erzeuge Offset $R_j * f_{G(x,c)}$ in Register $R_i$“ lautet, dann gibt $M'_S$ den Straight-Line-Befehl „$R_j \leftarrow R_j + f_{G(x,c)} * R_i$“ aus. Falls „Erzeuge Offset $-R_j * f_{G(c)}$ in Register $R_i$“ zu beschreiben war, so wird „$R_j \leftarrow R_j - f_{G(c)} * R_i$“ ausgegeben.

$h \to h + 1$: $G(c)$ hat zwei Vorgängergatter $D(c_1)$ and $E(c_2)$, wobei $c_1$ die Konfiguration ist, die auf der linken Verzweigung des Berechnungsbaums $T_{M(c)}$ die erste Konfiguration mit universellem, existentiellem oder Endzustandund ist und $c_2$ die entsprechende Konfiguration auf der rechten Seite. Sei $G(c)$ ein universelles Gatter und das zu beschreibende Straight-Line-Programm laute „Erzeuge Offset $R_j * f_{G(x,c)}$ in Register $R_i$“.

$M'_S(c)$ teilt sich in 4 Äste auf. (Da wir einen Binärbaum benötigen, geschieht dies in zwei Schritten.) Auf den Ästen werden dann folgende Straight-Line-Programme beschrieben:

1. Offset von $-R_k * f_E(c_2)$ in $R_j$;
2. Offset von $R_i * f_D(c_1)$ in $R_k$;
3. Offset von $R_k * f_E(c_2)$ in $R_j$;
4. Offset von $-R_i * f_D(c_1)$ in $R_k$.

Dies geschieht, indem auf den Ästen 2. und 4. der linke Pfad von $M(c)$ simuliert wird und auf den Ästen 1. und 3. der rechte Pfad simuliert wird, bis die Gatter $D(c_1)$ bzw. $E(c_2)$ erreicht werden. Aufgrund der Induktionshypothese wissen wir, dass die Straight-Line-Programme in $1. - 4.$ schon durch $M'_S$ beschrieben werden können.

Nun können wir $M_S$ konstruieren. $M_S(x)$ verzweigt sich in zwei Pfade. Auf dem ersten Pfad wird ein vollständiger binärer Berechnungsbaum der Höhe $d$ konstruiert, auf dessen Blätter, mit Ausnahme der letzten beiden Blätter, Straight-Line-Befehle ohne Effekt (also z.B. $R_1 \leftarrow R_1 + 0 * R_1$) geschrieben werden. Auf seine letzten beiden Blätter werden die folgenden Befehle geschrieben:

1. $R_2 \leftarrow R_2 + 1 * R_1$
2. $R_1 \leftarrow R_1 - 1 * R_2$

Auf dem zweiten Pfad der ersten Verzweigung simuliert $M_S(x)$ die Maschine $M_A(x)$ bis der erste universelle oder existentielle Zustand erreicht wird. Wir nennen die Konfiguration von $M_A$ zu diesem Zeitpunkt $c_A$. Nun wird $M'_S$ mit der Aufgabe simuliert, das Straight-Line-Programm „Erzeuge Offset $R_2 * f_{G(c_A)}$ in $R_1$“ zu beschreiben.

Es ist nun einfach zu sehen, dass $M_S(x)$ das Straight-Line-Programm $P_{|x|}$ beschreibt. Und da $\overline{P}$ die Funktion $f$ berechnet, haben wir bewiesen, dass $f \in \mathrm{U_{FBT}}\text{-SLP-SIZE-REG}(2^{n^{O(1)}}, 3)$ gilt.

### $K_6 \subseteq K_7$

In Theorem 5.35 von [Vol99a] wurde gezeigt, dass eine enge Beziehung zwischen Straight-Line-Programmen und Matrixprogrammen existiert: Aus einem beliebigen Straight-Line-Programm der Länge $l$ mit $k$ Registern kann man ein äquivalentes Matrixprogramm der gleichen Länge mit Dimension $k$ erzeugen, indem man jeden Befehl des Straight-Line-Programm nach einfachen Regeln in eine Matrix umwandelt.

### $K_7 \subseteq K_8$

Sei $f \in \mathrm{U_{FBT}}\text{-MP-SIZE-DIM}(2^{n^{O(1)}}, 3)$. Dann existiert ein nichtdeterministischer Polynomialzeit-Transducer $M$, der eine Familie von Matrixprogrammen beschreibt, die $f$ berechnen. Sei $h$ die Funktion, die auf Eingabe $x$ die Anzahl der Blätter von $T_{M(x)}$ berechnet und sei $g$ die Funktion, die auf Eingabe $(x, n)$ die Matrix berechnet, die auf Pfad $n$ von $T_{M(x)}$ ausgegeben wird, wobei die Variablen $x_i$ durch das $i$-te Bit von $x$ ersetzt werden. Beide Funktionen sind in FP und für alle $x$ gilt $f(x) = \mathfrak{F}(g(x,1), \ldots, g(x, h(x)))$. Deshalb gilt $f \in \mathfrak{F}\text{-FP}$.

### $K_8 \subseteq K_1$

Sei $f \in \mathfrak{F}\text{-FP}$. Dann gibt es zwei Polynomialzeit-berechenbare Funktionen $g\colon \Sigma^* \times \mathbb{N} \to \Sigma$ und $h\colon \Sigma^* \to \mathbb{N}$, so dass für alle $x$ gilt $f(x) = \mathfrak{F}(g(x,1), \ldots, g(x,h(x)))$.

Die Matrizen $g(x,i)$ können in polynomiellem Platz berechnet werden. Wir müssen aber zeigen, dass auch das Ergebnis

$$f(x) = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \cdot \left( \prod_{i=1}^{h(x)} g(x,i) \right) \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

in polynomiellem Platz berechnet werden kann. Dies wird durch die Tatsache erschwert, dass $h(x)$ exponentielle Größe in $|x|$ erreichen kann. Dies führt

nicht nur dazu, dass es exponentiell viele Matrizen $g(x,i)$ gibt, die nicht alle in polynomiellem Platz gespeichert werden können, sondern auch noch dazu, dass die Einträge der Matrix $\prod_{i=1}^{h(x)} g(x,i)$ exponentiell in $h(x)$ sein können, was ihnen exponentielle Länge in $|x|$ gibt.

Aber diese Probleme können durch ein Verfahren überwunden werden, dass z.B. auch im Beweis des Satzes #APTIME $\subseteq$ FPSPACE in [Lad89] benutzt wurde:

Sei $prod(x,a,l) = \prod_{i=a}^{\min\{a+2^l-1,h(x)\}} g(x,i)$. Dann gilt

$$f(x) = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \cdot prod(x,1,|h(x)|) \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}.$$

Außerdem können wir die Gleichung $prod(x,a,l+1) = prod(x,a,l) \cdot prod(x, a+2^l, l)$ verwenden, um eine rekursive Prozedur zur Berechnung von $f(x)$ zu erhalten.

Es ist uns zwar aufgrund ihrer Größe immer noch nicht möglich, die Teilprodukte $prod(x,a,l)$ auf das Arbeitsband zu schreiben, aber wir beschreiben nun eine rekursive Prozedur $bit(i,j,k,x,a,l)$, die das $k$-te Bit des Eintrags $(i,j)$ der Matrix $prod(x,a,l)$ berechnet:

```
function bit(i, j, k, x, a, l);
begin
  if l = 0
    then
      Gib das k-te Bit des Eintrags (i, j) der Matrix g(x, a) aus.
    else
      Gib das k-te Bit des Eintrags (i, j) der Matrix
      prod(x, a, l − 1) · prod(x, a + 2^(l−1), l − 1) aus.
end
```

Dieses Verfahren ist deshalb rekursiv, weil zur Berechnung von $prod(x,a,l-1) \cdot prod(x, a+2^{l-1}, l-1)$ wieder auf die Prozedur *bit* zurückgegriffen wird.

Da die Addition und die Multiplikation in logarithmischem Platz berechnet werden können, ist es uns möglich, das Matrixprodukt zweier Matrizen mit exponentiell großen Einträgen in $|x|$ in polynomiellem Platz in $|x|$ zu berechnen. Wir müssen nur die obige Prozedur für jedes Bit jedes Eintrags laufen lassen. Falls die Prozedur ein Bit ihrer Eingabe verlangt, wird ihr dies durch einen rekursiven Aufruf von *bit* gegeben.

Die Tiefe der Rekursion ist $|h(x)|$. Dies ist polynomiell in $|x|$. Während jeden Schrittes der Rekursion benötigen wir Platz, der logarithmisch in der Größe der Matrizen $prod(x,b,m)$ mit $0 \leq m \leq |h(x)|$ ist. Dieser Platzbedarf

ist polynomiell in $|x|$. Demnach benötigt die Berechnung von $bit(i, j, k, x, a, l)$ polynomiellen Platz. Da $f(x)$ der Eintrag $(1, 1)$ der Matrix $prod(x, 1, |h(x)|)$ ist, können wir diesen Wert durch Aufrufe von $bit(1, 1, k, x, 1, |h(x)|)$ für $1 \leq k \leq |f(x)|$ erzeugen. □

### 5.3.1 Kompakte Versionen von Komplexitätsklassen

Sei $C$ ein Boolescher Schaltkreis mit $n$ Eingabegattern. Dann beschreibt $C$ ein Wort aus $\{0, 1\}^{2^n}$ dessen $i$-tes Bit durch die Ausgabe von $C$ auf Eingabe $i - 1$ gegeben ist. (Auf diese Weise können durch Schaltkreise natürlich nur Wörter beschrieben werden, deren Länge eine Potenz von 2 ist; allerdings gibt es Möglichkeiten, willkürliche Wortlängen zu erreichen. Siehe dazu auch [Vei98].) Wir nennen für eine Sprache $L$ die Menge aller Booleschen Schaltkreise, die Wörter aus $L$ beschreiben, die *kompakte Version von* $L$ („Succinct Version of $L$“).

Helmut Veith hat gezeigt (siehe [Vei98]), dass die kompakte Version einer Sprache $L$ vollständig für eine Komplexitätsklasse $\mathcal{C}$ unter Projektionen erster Ordnung ist, wenn $L$ eine Blattsprache für $\mathcal{C}$ ist. Dabei sind Projektionen erster Ordnung („First-Order Projections“), wie sie in [Imm99] beschrieben sind, eine uniforme Version der Projektionsreduktionen von Valiant (siehe [Val82]).

**Korollar 5.3.4.** *Gegeben sei das Problem, für eine Folge von* $3 \times 3$*-Matrizen über* $\{-1, 0, 1\}$ *herauszufinden, ob der Eintrag in der oberen linken Ecke des Produkts der Matrizen ungleich* 0 *ist. Dann ist die kompakte Darstellung dieses Problems vollständig für* PSPACE *unter Projektionen erster Ordnung.*

*Beweis.* Aus Satz 5.3.3 folgt direkt, dass die Funktion $\mathfrak{F}$ in ihrer Entscheidungsversion („Ist der Eintrag in der oberen linken Ecke des Produkts der Eingabematrizen ungleich 0?“) eine Blattsprache für PSPACE ist. Mithilfe des Resultats von Veith folgt daraus das Korollar. □

# Index

# Literaturverzeichnis


[BCS92] D. P. Bovet, P. Crescenzi, and R. Silvestri. A uniform approach to define complexity classes. *Theoretical Computer Science*, 104:263–283, 1992.

[BE58] J. R. Büchi and C. C. Elgot. Decision problems of weak second order arithmetics and finite automata, Part I. *Notices of the American Mathematical Society*, 5:834, 1958.

[BG81] C. Bennett and J. Gill. Relative to a random oracle $P^A \neq NP^A \neq coNP^A$ with probability 1. *SIAM Journal on Computing*, 10:96–113, 1981.

[BGS75] T. Baker, J. Gill, and R. Solovay. Relativizations of the P=NP problem. *SIAM Journal on Computing*, 4:431–442, 1975.

[BI87] M. Blum and R. Impagliazzo. Generic oracles and oracle classes. In *Proceedings 28th IEEE Symposium on Foundations of Computer Science*, pages 118–126. IEEE Computer Society Press, 1987.

[BIS90] D. A. Mix Barrington, N. Immerman, and H. Straubing. On uniformity within $NC^1$. *Journal of Computer and System Sciences*, 41:274–306, 1990.

[BLM93] F. Bédard, F. Lemieux, and P. McKenzie. Extensions to Barrington's M-program model. *Theoretical Computer Science*, 107:31–61, 1993.

[BOC92] M. Ben-Or and R. Cleve. Computing algebraic formulas using a constant number of registers. *SIAM Journal on Computing*, 21:54–58, 1992.

[Büc62] J. R. Büchi. On a decision method in restricted second-order arithmetic. In *Proceedings Logic, Methodology and Philosophy of Sciences 1960*, Stanford, CA, 1962. Stanford University Press.

[BV98] H.-J. Burtschick and H. Vollmer. Lindström quantifiers and leaf language definability. *International Journal of Foundations of Computer Science*, 9:277–294, 1998.

[CCG+94] R. Chang, B. Chor, O. Goldreich, J. Hartmanis, J. Hastad, D. Ranjan, and P. Rohatgi. The random oracle hypothesis is false. *Journal of Computer and System Sciences*, 49:24–39, 1994.

[CIY97] S. A. Cook, R. Impagliazzo, and T. Yamakami. A tight relationship between generic oracles and type-2 complexity theory. *Information and Computation*, 137(2):159–170, 1997.

[CMTV98] H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer. Nondeterministic $NC^1$ computation. *Journal of Computer and System Sciences*, 57:200–212, 1998.

[EF95] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Perspectives in Mathematical Logic. Springer Verlag, Berlin Heidelberg, 1995.

[Fos93] J. A. Foster. The generic oracle hypothesis is false. *Information Processing Letters*, 45:59–62, 1993.

[GKV03] M. Galota, S. Kosub, and H. Vollmer. Generic separations and leaf languages. *Mathematical Logic Quarterly*, 49(4):353–362, 2003.

[GV01] M. Galota and H. Vollmer. A generalization of the Büchi-Elgot-Trakhtenbrot-Theorem. In *Computer Science Logic*, Lecture Notes in Computer Science, Berlin Heidelberg, 2001. Springer Verlag.

[GV05] M. Galota and H. Vollmer. Functions computable in polynomial space. *Information and Computation*, 198(1):56–70, 2005.

[HLS+93] U. Hertrampf, C. Lautemann, T. Schwentick, H. Vollmer, and K. W. Wagner. On the power of polynomial time bit-reductions. In *Proceedings 8th Structure in Complexity Theory*, pages 200–207, 1993.

[Imm99] N. Immerman. *Descriptive Complexity*. Graduate Texts in Computer Science. Springer Verlag, New York, 1999.

[KSV00] S. Kosub, H. Schmitz, and H. Vollmer. Uniform characterizations of complexity classes of functions. *International Journal of Foundations of Computer Science*, 11(4):525–551, 2000.

[Lad89] R. Ladner. Polynomial space counting problems. *SIAM Journal on Computing*, 18(6):1087–1097, 1989.

[Lin66] P. Lindström. First order predicate logic with generalized quantifiers. *Theoria*, 32:186–195, 1966.

[LMSV01] C. Lautemann, P. McKenzie, T. Schwentick, and H. Vollmer. The descriptive complexity approach to LOGCFL. *Journal of Computer and Systems Sciences*, 62(4):629–652, 2001.

[MP71] R. McNaughton and S. Papert. *Counter-Free Automata*. MIT Press, 1971.

[PV01] T. Peichl and H. Vollmer. Finite automata with generalized acceptance criteria. *Discrete Mathematics and Theoretical Computer Science*, 4(2):179–192, 2001.

[Soa87] R. I. Soare. *Recursively Enumerable Sets and Degrees – A Study of Computable Functions and Computably Generated Sets*. Perspectives in Mathematical Logic. Springer Verlag, Berlin Heidelberg, 1987.

[Str94] H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, Boston, 1994.

[Tho82] W. Thomas. Classifying regular events in symbolic logic. *Journal of Computer and Systems Sciences*, 25:360–376, 1982.

[Tra61] B. A. Trakhtenbrot. Finite automata and logic of monadic predicates. *Doklady Akademii Nauk SSSR*, 140:326–329, 1961. In Russian.

[Val82] L. Valiant. Reducibility by algebraic projections. *L'enseignement mathématique*, 28:253–268, 1982.

[Vei98] H. Veith. Succinct representation, leaf languages, and projection reductions. *Information & Computation*, 142:207–236, 1998.

[Ver93] N. K. Vereshchagin. Relativizable and non-relativizable theorems in the polynomial theory of algorithms. *Izvestija Rossijskoj Akademii Nauk*, 57:51–90, 1993. In Russian.

[Vol99a] H. Vollmer. *Introduction to Circuit Complexity – A Uniform Approach*. Texts in Theoretical Computer Science. Springer Verlag, Berlin Heidelberg, 1999.

[Vol99b] H. Vollmer. Uniform characterizations of complexity classes. *Complexity Theory Column 23, ACM-SIGACT News*, 30(1):17–27, 1999.

[VT89] H. Venkateswaran and M. Tompa. A new pebble game that characterizes parallel complexity classes. *SIAM J. on Computing*, 18:533–549, 1989.

[Wag02] K. W. Wagner. A characterisation of FPSPACE. Personal Correspondence, 2002.

# Lebenslauf

## Persönliche Daten

| | |
|---|---|
| Name | Matthias Franz Joachim Galota |
| geboren am | 10. Juni 1974 in Würzburg |

## Schulbildung

| | |
|---|---|
| 1984–1993 | Friedrich-Koenig-Gymnasium in Würzburg |
| Juli 1993 | Allgemeine Hochschulreife |

## Berufsausbildung

| | |
|---|---|
| 1993–1999 | Informatikstudium an der Universität Würzburg |
| November 1999 | Diplom |

## Berufstätigkeit

| | |
|---|---|
| 1999–2002 | wissenschaftlicher Mitarbeiter am Lehrstuhl für Theoretische Informatik der Universität Würzburg |
| 2002–2004 | wissenschaftlicher Mitarbeiter im Fachgebiet Theoretische Informatik der Universität Hannover |
| seit Mai 2004 | Software-Entwickler bei der 3SOFT GmbH, Erlangen |

## Wissenschaftliche Veröffentlichungen

1. M. Galota, C. Glaßer, S. Reith, H. Vollmer. *A Polynomial-time Approximation Scheme for Base Station Positioning in UMTS Networks*, Proc. 5th Discrete Algorithms and Methods for Mobile Computing and Communications, 2001.

2. M. Galota, H. Vollmer. *A Generalization of the Büchi-Elgot-Trakhtenbrot Theorem*, Computer Science Logic 2001, Springer Lecture Notes in Computer Science Vol. 2142, 2001.

3. M. Galota, S. Kosub, H. Vollmer. *Generic Separations and Leaf Languages*, Mathematical Logic Quarterly, 49(4), 2003.

4. M. Galota, H. Vollmer. *Functions Computable in Polynomial Space*, Information and Computation, 198(1), 2005.