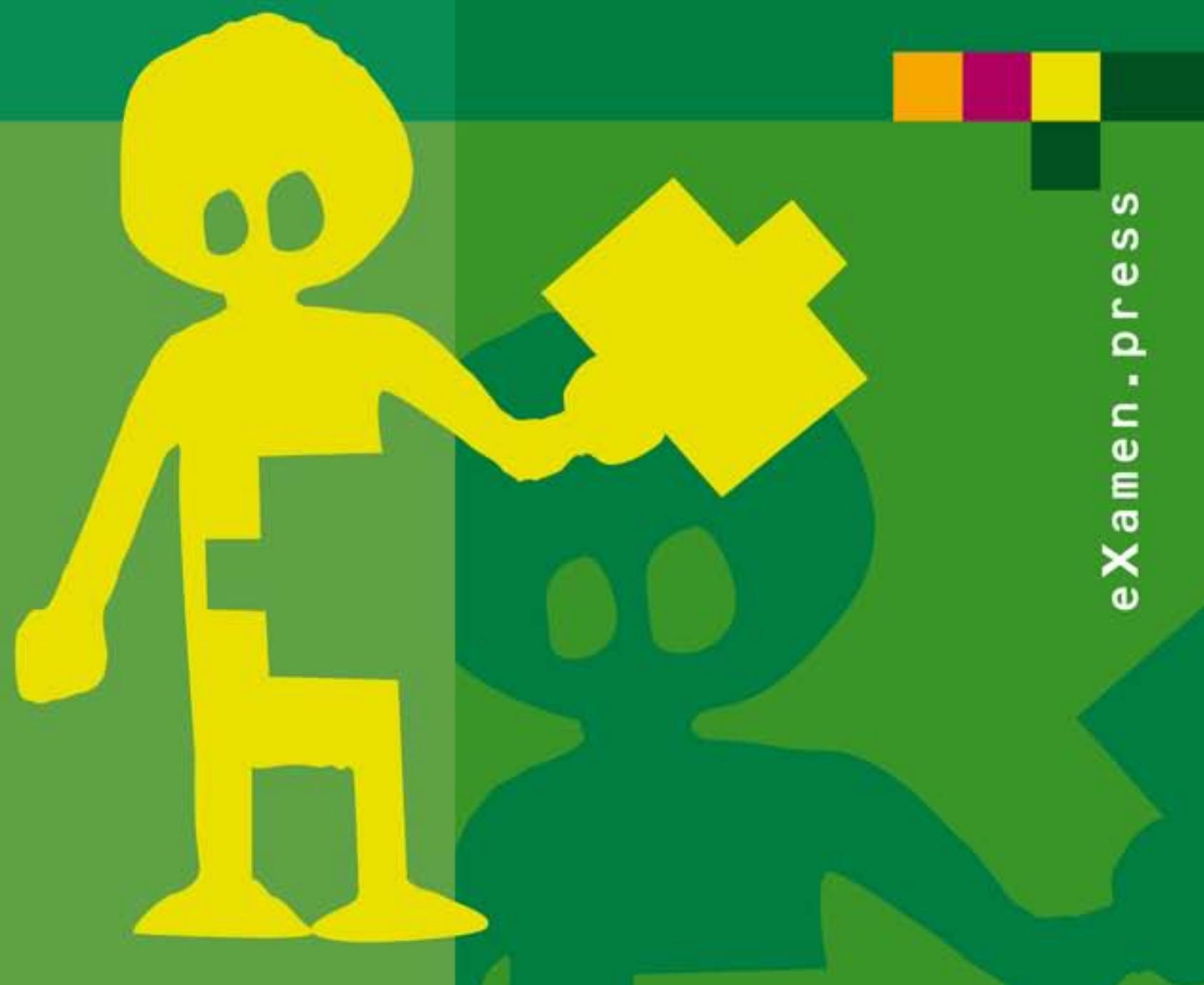


JÖRG ROTHE

Komplexitätstheorie und Kryptologie



eXamen.press



Springer

eXamen.press

eXamen.press ist eine Reihe, die Theorie und Praxis aus allen Bereichen der Informatik für die Hochschulausbildung vermittelt.

Jörg Rothe

Komplexitätstheorie und Kryptologie

Eine Einführung in Kryptokomplexität

Prof. Dr. Jörg Rothe
Institut für Informatik
Heinrich-Heine-Universität Düsseldorf
Universitätsstr. 1
40225 Düsseldorf
rothe@cs.uni-duesseldorf.de

ISBN 978-3-540-79744-9

e-ISBN 978-3-540-79745-6

DOI 10.1007/978-3-540-79745-6

eXamen.press ISSN 1614-5216

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

© 2008 Springer-Verlag Berlin Heidelberg

Dieses Werk ist urheberrechtlich geschützt. Die dadurch begründeten Rechte, insbesondere die der Übersetzung, des Nachdrucks, des Vortrags, der Entnahme von Abbildungen und Tabellen, der Funksendung, der Mikroverfilmung oder der Vervielfältigung auf anderen Wegen und der Speicherung in Datenverarbeitungsanlagen, bleiben, auch bei nur auszugsweiser Verwertung, vorbehalten. Eine Vervielfältigung dieses Werkes oder von Teilen dieses Werkes ist auch im Einzelfall nur in den Grenzen der gesetzlichen Bestimmungen des Urheberrechtsgesetzes der Bundesrepublik Deutschland vom 9. September 1965 in der jeweils geltenden Fassung zulässig. Sie ist grundsätzlich vergütungspflichtig. Zu widerhandlungen unterliegen den Strafbestimmungen des Urheberrechtsgesetzes.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Einbandgestaltung: KünkelLopka, Heidelberg

Gedruckt auf säurefreiem Papier

9 8 7 6 5 4 3 2 1

springer.de

Für Irene, Paula und Ella

Vorwort

Dieses Buch führt in die Komplexitätstheorie und Kryptologie ein, zwei eng miteinander verflochtene Gebiete der Theoretischen Informatik. Basierend auf Vorlesungen, die ich seit dem Jahr 2000 an der Heinrich-Heine-Universität Düsseldorf und davor, von 1996 bis 2000, an der Friedrich-Schiller-Universität Jena hielt, ist dieses Lehrbuch in erster Linie für Master- und Diplomstudierende sowie für Doktoranden und Doktorandinnen der Informatik, Mathematik und Ingenieurwissenschaften geschrieben. Forscher, Dozenten und Praktiker, die in diesen Gebieten arbeiten, werden es ebenfalls als eine umfassende, aktuelle, forschungsbezogene Quelle hinsichtlich zentraler Themen in der Kryptokomplexität schätzen. Kapitel 1 gibt einen detaillierteren Überblick über die Themen dieses Buches, macht konkrete Vorschläge für seinen Einsatz in der Lehre und umreißt kurz die Inhalte der einzelnen Kapitel.

Danksagungen

Zu tiefem Dank verpflichtet bin ich Gerd Wechsung, der mich ermutigte, dieses Buch zu schreiben. Großen Dank schulde ich auch Dorothea Baumeister, Gábor Erdélyi, Tobias Riege, Holger Spakowski und Gerd Wechsung, die Teile dieses Buches gründlich lasen; ihre Korrekturen und Vorschläge waren mir eine unschätzbarre Hilfe. Ebenso gilt mein besonderer Dank allen, die mich auf Fehler in der englischen Ausgabe dieses Buches aufmerksam machten, so dass ich sie in der hier vorliegenden deutschen Übersetzung beheben konnte. Außer den gerade genannten sind dies insbesondere Daniel Baselt, Stefan Bunger, Christian Glaßer, Frank Gurski, Yves Jerschow, Peter Lieven, Markus Nowak und Holger Wichert. Für die verbleibenden Fehler übernehme ich die Verantwortung selbst, denn ich konnte niemanden sonst dazu überreden, sich verantwortlich zu fühlen.

Für ihre großzügige Hilfe und Unterstützung und ihren nützlichen Rat über nun anderthalb Jahrzehnte sowie für zahlreiche wundervolle Forschungsaufenthalte an ihren Institutionen, der University of Rochester und dem Rochester Institute of Technology, schulde ich Lane A. Hemaspaandra und Edith Hemaspaandra tiefen Dank. Auch danke ich all meinen Mitforschern, Koautoren und Kollegen: Lane und Edith Hemaspaandra, Dorothea Baumeister, Alina Beygelzimer, Bernd Borchert, Dagmar

Bruß, Gábor Erdélyi, Piotr Faliszewski, Judy Goldsmith, André Große, Christopher M. Homan, Zhigen Jiang, Tim Meyer, Mitsunori Ogihara, Kari Pasanen, Rajesh P. N. Rao, Tobias Riege, Amitabh Saxena, Holger Spakowski, Jörg Vogel, Osamu Watanabe, Gerd Wechsung und Masaki Yamamoto. Einige der in diesem Buch beschriebenen Forschungsresultate wurden gemeinsam mit ihnen erzielt.

Danken möchte ich auch meinen Freunden in Rochester, NY, und Atlanta, GA, die einige der Stories in diesem Buch lesen mussten: Kathleen und Charles Landers-Appell, Mette Stromnes, Dave Lutz, Narin Hassan, Mark Leibert und Jodi Beckwith und Stefan Cohen.

Für inspirierende Diskussionen danke ich Klaus Ambos-Spies, Sigurd Assing, Harald Hempel, Uwe Schöning, Andreas Stelzer, Dietrich Stoyan, Klaus W. Wagner und Gerd Wechsung. Für die stets angenehme Arbeitsatmosphäre danke ich meinen Kollegen an den Instituten für Informatik in Düsseldorf und Jena: Volker Au-rich, Dorothea Baumeister, Stefan Conrad, Gábor Erdélyi, André Große, Arndt von Haeseler, Harald Hempel, Maren Hinrichs, Dieter Kratsch, Martin Lercher, Michael Leuschel, Gerhard Lischke, Martin Mauve, Haiko Müller, Tobias Riege, Michael Schöttner, Holger Spakowski, Jörg Vogel, Egon Wanke und Gerd Wechsung. Für ihre Unterstützung danke ich insbesondere unseren technischen Mitarbeitern und Sekretärinnen in Düsseldorf: Claudia Forstinger, Claudia Kiometzis, Guido Königstein, Berthold Nöckel, Marga Pothoff, Janus Tomaschewski und Lutz Voigt.

Claude Crépeau danke ich für seine freundliche Erlaubnis, die von ihm entworfenen Zeichnungen von Alice und Bob in Abbildung 1.1 verwenden zu dürfen.

Für die stets professionelle und freundliche Unterstützung danke ich den Mitarbeitern des Springer-Verlags.

Dieses Buch enthält einige Resultate, die aus der Forschung meiner Gruppe und aus Gemeinschaftsprojekten mit Kollegen in Deutschland, den USA und Japan hervorgingen. Mein besonderer Dank gilt den Institutionen, die diese Forschung förderten. Für die ursprüngliche englische Fassung des Buchs betrifft dies das DFG-Projekt RO 1202/9-1 und das durch den DAAD und die NSF gemeinsam unterstützte internationale Projekt NSF-INT-9815095/DAAD-315-PPP-gü-ab. Für die deutsche Übersetzung dieses Buchs betrifft dies die DFG-Projekte RO 1202/9-3, RO 1202/11-1 und RO 1202/12-1 (innerhalb des Projekts „Computational Foundations of Social Choice“, das im EUROCORES-Programm LogICCC der European Science Foundation gefördert wird) sowie internationale Gemeinschaftsprojekte, die von der Alexander von Humboldt-Stiftung im TransCoop-Programm (gemeinsam mit der amerikanischen NSF) und im CONNECT-Programm (gemeinsam mit der japanischen JSPS) gefördert wurden.

An allererster Stelle danke ich meiner Frau Irene und meinen Töchtern Paula und Ella für ihre Liebe, Ermutigung und Unterstützung und ihren Rat. Insbesondere danke ich Irene für ihren kleinen Drachen, Ring und Gral, und ich danke Paula und Ella für ihre Auftritte in einigen Stories, die in diesem Buch erzählt werden.

Inhaltsverzeichnis

Vorwort	VII
1 Einladung zur Kryptokomplexität	1
2 Grundlagen der Informatik und Mathematik	11
2.1 Algorithmen: Der Euklidische Algorithmus	11
2.2 Formale Sprachen und Berechenbarkeitstheorie	19
2.3 Logik	33
2.3.1 Aussagenlogik	33
2.3.2 Prädikatenlogik	39
2.4 Algebra, Zahlentheorie und Graphentheorie	43
2.4.1 Algebra und Zahlentheorie	43
2.4.2 Permutationsgruppen	48
2.4.3 Graphentheorie	49
2.5 Wahrscheinlichkeitstheorie	52
2.6 Übungen und Probleme	54
2.7 Zusammenfassung und bibliographische Notizen	58
3 Grundlagen der Komplexitätstheorie	61
3.1 Aufgaben und Ziele der Komplexitätstheorie	61
3.2 Komplexitätsmaße und -klassen	64
3.3 Beschleunigungs-, Kompressions- und Hierarchiesätze	72
3.4 Zwischen Logarithmischem und Polynomialem Raum	82
3.5 Reduzierbarkeiten und Vollständigkeit	88
3.5.1 Many-One-Reduzierbarkeiten, Härte und Vollständigkeit	88
3.5.2 NL-Vollständigkeit	93
3.5.3 NP-Vollständigkeit	100
3.6 Innerhalb von NP	120
3.6.1 P versus NP und das Graphisomorphie-Problem	120
3.6.2 Die Berman–Hartmanis–Isomorphievermutung und Einwegfunktionen	122

3.7	Übungen und Probleme	129
3.8	Zusammenfassung und bibliographische Notizen	136
4	Grundlagen der Kryptologie	145
4.1	Aufgaben und Ziele der Kryptologie	145
4.2	Einige klassische Kryptosysteme und ihre Kryptoanalyse	148
4.2.1	Substitutions- und Permutationschiffren	149
4.2.2	Affin-lineare Blockchiffren	155
4.2.3	Block- und Stromchiffren	165
4.3	Perfekte Geheimhaltung	172
4.3.1	Satz von Shannon und Vernams One-Time Pad	172
4.3.2	Entropie und Schlüsselmehrdeutigkeit	177
4.4	Übungen und Probleme	183
4.5	Zusammenfassung und bibliographische Notizen	190
5	Hierarchien über NP	195
5.1	Die boolesche Hierarchie über NP	196
5.2	Die Polynomialzeit-Hierarchie	215
5.3	Paralleler Zugriff auf NP	227
5.3.1	Eine kurze Abschweifung in die Theorie der Wahlsysteme	233
5.3.2	Gewinnerproblem für Young-Wahlen	235
5.4	Frage-Hierarchien über NP	239
5.5	Die boolesche Hierarchie kollabiert die Polynomialzeit-Hierarchie	245
5.6	Alternierende Turingmaschinen	249
5.7	Die Low- und die High-Hierarchie in NP	261
5.8	Übungen und Probleme	271
5.9	Zusammenfassung und bibliographische Notizen	278
6	Randomisierte Algorithmen und Komplexitätsklassen	293
6.1	Das Erfüllbarkeitsproblem der Aussagenlogik	294
6.1.1	Deterministische Zeitkomplexität	296
6.1.2	Probabilistische Zeitkomplexität	297
6.2	Probabilistische Polynomialzeit-Klassen	302
6.2.1	PP, RP und ZPP: Monte-Carlo- und Las-Vegas-Algorithmen	302
6.2.2	BPP: Probabilistische Polynomialzeit mit beschränktem Fehler	310
6.3	Quantoren und Arthur-Merlin-Spiele	314
6.3.1	Quantoren und BPP	314
6.3.2	Die Arthur-Merlin-Hierarchie	321
6.4	Zählklassen	326
6.5	Graphisomorphie und Lowness	330
6.5.1	Graphisomorphie ist in der Low-Hierarchie	330
6.5.2	Graphisomorphie ist in SPP	334
6.6	Übungen und Probleme	338
6.7	Zusammenfassung und bibliographische Notizen	343

7 RSA-Kryptosystem, Primzahltests und das Faktorisierungsproblem	349
7.1 RSA	350
7.1.1 Das RSA Public-Key-Kryptosystem	350
7.1.2 Digitale Signaturen mit RSA	355
7.2 Primzahltests	355
7.2.1 Fermat-Test	358
7.2.2 Miller–Rabin-Test	362
7.2.3 Solovay–Strassen-Test	368
7.2.4 Das Primzahl-Problem ist in P	374
7.3 Das Faktorisierungsproblem	375
7.3.1 Probdivision	376
7.3.2 Pollards Algorithmus	377
7.3.3 Das quadratische Sieb	378
7.3.4 Andere Faktorisierungsmethoden	383
7.4 Sicherheit von RSA: Angriffe und Gegenmaßnahmen	386
7.5 Übungen und Probleme	395
7.6 Zusammenfassung und bibliographische Notizen	399
8 Weitere Public-Key-Kryptosysteme und Protokolle	403
8.1 Diffie–Hellman und das Problem des diskreten Logarithmus	404
8.1.1 Das Schlüsseltausch-Protokoll von Diffie und Hellman	405
8.1.2 Diskrete Logarithmen und das Diffie–Hellman-Problem	408
8.2 Die Protokolle von ElGamal	412
8.2.1 ElGamals Public-Key-Kryptosystem	412
8.2.2 Digitale Signaturen mit ElGamal	414
8.2.3 Sicherheit der Protokolle von ElGamal	416
8.3 Rabins Public-Key-Kryptosystem	424
8.3.1 Rabins Kryptosystem	425
8.3.2 Sicherheit des Systems von Rabin	427
8.4 Arthur-Merlin-Spiele und Zero-Knowledge	430
8.5 Das Public-Key-Kryptosystem von Merkle und Hellman	439
8.6 Die Protokolle von Rabi, Rivest und Sherman	442
8.7 Übungen und Probleme	450
8.8 Zusammenfassung und bibliographische Notizen	456
Tabellenverzeichnis	463
Abbildungsverzeichnis	465
Literaturverzeichnis	467
Sach- und Autorenverzeichnis	497

1

Einladung zur Kryptokomplexität

Über dieses Buch

Dieses Buch führt in zwei Gebiete ein, *Komplexitätstheorie* und *Kryptologie*, die eng miteinander verwandt sind, sich aber recht unabhängig voneinander entwickelt haben. Neben anderen Gebieten wie etwa der Zahlentheorie verwendet die moderne Kryptologie die mathematisch strengen Konzepte und Methoden der Komplexitätstheorie. Umgekehrt ist die aktuelle Forschung in der Komplexitätstheorie oft durch Fragen und Probleme motiviert, die in der Kryptologie auftreten. Das vorliegende Buch trägt diesem Trend Rechnung, und sein Gegenstand könnte daher treffend als „*Kryptokomplexität*“ bezeichnet werden, eine Art Symbiose dieser beiden Gebiete.

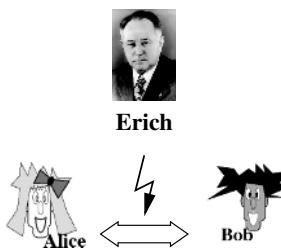


Abb. 1.1. Ein typisches kryptographisches Szenario

Abbildung 1.1 zeigt ein typisches Szenario der Kryptographie. Alice und Bob (deren Design von Crépeau entworfen wurde) möchten Nachrichten über einen unsicheren Kanal austauschen – etwa über eine öffentliche Telefonleitung oder über das Internet –, der von Erich abgehört wird. Deshalb verschlüsselt Alice ihre Nachricht an Bob so, dass Bob sie leicht entschlüsseln kann, Erich jedoch nicht. Kryptographie

ist die Kunst und Wissenschaft vom Entwurf sicherer Kryptosysteme. Alice und Bob verwenden Kryptosysteme und kryptographische Techniken, um ihre privaten Daten zu schützen und geheim zu halten, um ihre Nachrichten digital zu signieren, so dass ihre Unterschriften nicht gefälscht werden können, zum Zwecke der Authentikation, für den Schutz von Urheberrechten, zur sicheren Nutzung von Computer-Netzwerken und um in sicherer Weise über das Internet Informationen zu tauschen und Geschäfte zu machen.

Ihr Gegenspieler Erich ist ärgerlich, weil er zwar ihre Nachrichten empfangen und abhören, aber keinen Vorteil daraus schlagen kann. Sein Ziel ist die nicht authrorisierte Entschlüsselung ihrer Schlüsseltexte, er möchte sich ihre Schlüssel aneignen, um ihr Kryptosystem zu knacken. Kryptoanalyse ist die Kunst und Wissenschaft des Brechens von Kryptosystemen. Kryptologie umfasst beide Gebiete, die Kryptographie und die Kryptoanalyse.

Kryptographie und Kryptoanalyse führen seit Urzeiten einen immerwährenden Krieg gegeneinander. Als unsere Vorfahren zu denken und zu sprechen und zu schreiben lernten, wollten sie ihre Gedanken und Nachrichten nicht nur übermitteln, sondern auch gegen den Zugriff unauthorisierter Empfänger schützen, d.h., sie wollten sie geheim halten. So ist verbürgt, dass bereits Julius Cäsar, Alleinherrscher von Rom und Diktator auf Lebenszeit, von einem einfachen (und leicht zu brechenden) Kryptosystem Gebrauch machte.

Schlacht um Schlacht wird seither zwischen diesen beiden widerstreitenden Welten geschlagen: Sobald die Kryptographen ein neues Kryptosystem entworfen haben, geben die Kryptoanalytiker keine Ruhe, bevor sie es nicht gebrochen haben, woraufhin bessere Kryptosysteme entwickelt werden, und so fort. Die Frontlinie verläuft dabei nicht zwischen Ländern, sie teilt nicht nur Familien, sondern sie spaltet Personen: Oft sind Kryptographen gleichzeitig Kryptoanalytiker und umgekehrt.

Die Ausdrücke „Krieg“ und „Schlacht“ sind hier durchaus wörtlich zu nehmen. Während des Zweiten Weltkriegs war der Kampf der alliierten Codebrecher gegen die berüchtigte Verschlüsselungsmaschine *Enigma* der Deutschen Wehrmacht eine Sache von Leben und Tod. Die *Enigma*, einst für unbedingt sicher gehalten, wurde schließlich von den britischen Codebrechern in Bletchley Park gebrochen, unter anderem unterstützt durch die vorherige Arbeit polnischer Mathematiker und durch die Mithilfe eines deutschen Doppelspions. Die Leistung dieser Kryptoanalytiker war entscheidend – wenn nicht für den Krieg, so doch für eine Reihe von Schlachten, besonders für die großen Seeschlachten und die Zerstörung der deutschen U-Boot-Flotte. Ausführlich erzählen Singh [Sin99] und Bauer [Bau00a, Bau00b] die aufregende Geschichte dieses Kampfes zwischen deutschen Kryptographen und alliierten Kryptoanalytikern. Der Erfolg, die *Enigma* gebrochen zu haben, wird unter anderen Alan Turing zugeschrieben. Seine Brillanz als Kryptoanalytiker wird womöglich nur durch seine genialen, fundamentalen Leistungen in der Theoretischen Informatik noch übertroffen. Indem er die nach ihm benannte Turingmaschine erfand, legte er das Fundament der Berechenbarkeitstheorie, die als die Mutter der Komplexitätstheorie gilt.

Dass effiziente Algorithmen nützliche Anwendungen in der Praxis haben, ist offensichtlich. Im Gegensatz dazu versucht man in der Komplexitätstheorie zu zeigen,

dass bestimmte Probleme nicht effizient lösbar sind. Sie stellt die Mittel und Methoden zur Verfügung, mit denen Probleme hinsichtlich der ihnen innewohnenden Komplexität klassifiziert werden können. Ebenso liefert sie nützliche Werkzeuge und Techniken, um die relative Komplexität zweier gegebener Probleme mittels Reduktionen zu vergleichen.

In kryptographischen Anwendungen jedoch bedeutet Ineffizienz Sicherheit: Die Sicherheit der heute verwendeten Kryptosysteme beruht auf der Annahme, dass bestimmte Probleme nicht effizient gelöst werden können. Das Problem, ein Kryptosystem zu brechen, kann mittels Reduktionen auf geeignete Probleme zurückgeführt werden, die allgemein als „störrisch“ gelten, also widerspenstig in dem Sinn, dass sie sich einer effizienten Lösbarkeit widersetzen. Die Kryptographie erfordert und benutzt demnach die Berechnungswiderspenstigkeit von Problemen. Kurz gesagt benötigt die Kryptographie komplexitätstheoretische Begriffe, Modelle, Methoden und Ergebnisse, und dadurch motiviert sie diese. Insbesondere sind die Begriffe der Einwegfunktion, des interaktiven Beweissystems und des Zero-Knowledge-Protokolls sowohl in der Kryptologie als auch in der Komplexitätstheorie zentral, was die gegenseitige Durchdringung dieser beiden Gebiete demonstriert. Dieses Buch behandelt die Kryptologie und die Komplexitätstheorie gleichermaßen, mit einem besonderen Augenmerk auf ihre gegenseitige Beziehung.

Wie benutzt man dieses Buch?

Dieses Lehrbuch beruht auf den Vorlesungen des Autors, die an der Heinrich-Heine-Universität Düsseldorf und an der Friedrich-Schiller-Universität Jena seit 1996 gehalten wurden. Vorwiegend für Diplom- und Masterstudierende in Informatik, Mathematik und den Ingenieurwissenschaften geschrieben, ist es ebenso eine nützliche Quelle für Forscher, Dozenten und Praktiker, die in diesen Gebieten arbeiten.

Dieses Buch kann in mehr als einer Weise in der Lehre eingesetzt werden. Einerseits ist es für Einführungsveranstaltungen in Kryptologie aus komplexitätstheoretischer Sicht geeignet. Andererseits kann es für Einführungsveranstaltungen in Komplexitätstheorie verwendet werden, die besonderen Wert auf mögliche Anwendungen in der Kryptologie legen. In beiderlei Hinsicht gibt es einen umfassenden, aktuellen, forschungsorientierten Überblick über den aktuellen Stand in diesen beiden Gebieten, wobei ihr Zusammenhang betont und ein einheitlicher Zugang gewählt wird.

Am besten jedoch ist dieses Buch für eine Folge von zusammenhängenden Kursen geeignet, die in diese beiden Gebiete gemeinsam einführen und etwa in einem Modul oder mehreren Modulen zusammengefasst sind. Beispielsweise wird das in diesem Buch enthaltene Material seit einigen Jahren vom Autor in Düsseldorf in einer Reihe von sechs einsemestrigen Vorlesungen (zu je zwei Semesterwochenstunden, ergänzt um Übungen und Seminare) präsentiert, die insgesamt in einem Zeitraum von zwei Jahren gehalten werden und wobei je zwei Vorlesungen zu einem Modul kombiniert werden können. Je eines der Kapitel 3 bis 8 macht den Inhalt einer solchen Vorlesung aus, wobei die jeweils nötigen Grundlagen aus Kapitel 2 nach Bedarf ergänzt werden. Als vorteilhaft hat es sich dabei erwiesen, dass die

einzelnen Kapitel relativ unabhängig voneinander sind, so dass die Studierenden wirklich flexibel je zwei Vorlesungen zu einem Modul ihrer Wahl zusammenstellen können. Wenn nötig, können sie sich noch fehlenden Stoff aus anderen Kapiteln bzw. Vorlesungen mit Hilfe des Buches im Selbststudium leicht aneignen. Die starke Nachfrage nach diesen Veranstaltungen und der positive Zuspruch seitens der Studierenden, sowohl im persönlichen Gespräch als auch in anonymen Befragungen, lassen den Schluss zu, dass dieser Zugang, der sich auf die gegenseitige Verflechtung von Komplexitätstheorie und Kryptologie konzentriert, für die Studierenden nutzbringender ist, als wenn beide Gebiete separat und unabhängig unterrichtet würden. Natürlich können und sollten die in diesem Buch behandelten Themen um aktuelle Forschungsergebnisse der Originalliteratur und anderes interessantes Material erweitert werden. Detaillierte Beschreibungen der Veranstaltungen in Düsseldorf findet man unter <http://ccc.cs.uni-duesseldorf.de/~rothe/vorlesungen>.

Viel Sorgfalt wurde der Motivation und Erklärung der präsentierten Begriffe und Resultate gewidmet. Zahlreiche Beispiele, Abbildungen und Tabellen sollen den Text zugänglich, verständlich, leicht lesbar und hoffentlich hin und wieder auch unterhaltsam machen. Gelegentlich wird daher ein Begriff oder Satz, bevor er abstrakt, formal und mathematisch präsentiert wird, zunächst mittels einer kleinen Geschichte (einer „Story“) eingeführt und anhand von mehr oder weniger alltäglichen Beispielen erläutert. Dieses Buch zu lesen, ist jedoch kein reines Vergnügen. Es ist auch harte Arbeit: Jedes Kapitel bietet dem Leser eine Reihe von Übungen und Problemen an, einige mit Hinweisen für mögliche Lösungen oder mit Verweisen auf die Originalliteratur. Der Schwierigkeitsgrad der Übungen bewegt sich in einem recht großen Bereich; es gibt ziemlich leichte Übungen und es gibt schwere. Viele der Probleme sind überaus große Herausforderungen. Einige von ihnen beschreiben erst kürzlich gelöste Forschungsfragen, die manchmal tiefe Einsichten oder clevere Ideen erfordern. Selbst wenn sich diese als zu schwierig erweisen sollten, lohnt es sich, eine eigene Lösung zu versuchen.

Dank der umfassenden Bibliographie (mit 550 Einträgen) und dem Sach- und Autorenverzeichnis (mit 1569 Hauptstichwörtern und 995 Nebeneinträgen und Querverweisen) ist dieses Buch auch für Wissenschaftler von Wert, die in der Komplexitätstheorie oder Kryptologie arbeiten. Es bahnt sich seinen Weg von den grundlegenden Anfängen bis hin zu den Fronten der aktuellen Forschung in ausgewählten Themen dieser beiden Gebiete, wobei Wert auf einen einheitlichen Zugang gelegt wird. Jedes Kapitel schließt mit einer Zusammenfassung, die die historische Entwicklung der zuvor dargelegten Begriffe und Resultate beschreibt, verwandte Begriffe und Ideen erklärt und umfassende, detaillierte bibliographische Anmerkungen macht.

Das Sach- und Autorenverzeichnis enthält eine Fülle von Stichwörtern und Querverweisen, denn ein Lehrbuch ist nur so nützlich wie sein Index.¹ Jedes Stichwort kann mehrere Einträge haben, einen fettgedruckten Haupteintrag, der auf die Defi-

¹ Angenommen, man sucht nach jedem Vorkommen des Ausdrucks *Baby-Klonen* in diesem Buch. Oder man interessiert sich für ein spezielles Werkzeug, etwa eine *Kettensäge* oder eine *Turingmaschine*. Oder man möchte alles wissen, was dieses Buch über *Polygamie*, den

nition des entsprechenden Begriffs verweist, und eine Anzahl weiterer Einträge, die auf Sätze verweisen, in denen dieser Begriff vorkommt. Ein Lehrbuch ohne Index, oder mit einem dürftigen oder schlampig erstellten Index, ist dem Leser kaum eine größere Hilfe als eine Bibliothek ohne Katalog, in der alle Bücher unsortiert auf einen Haufen geworfen wurden. Man steht dann vielleicht vor diesem gewaltigen Bücherberg und weiß, dass man alles Wissen und alle Weisheit des Universums darin finden könnte, und doch findet man die ganz spezielle Information nicht, die man so verzweifelt sucht. Diese Erkenntnis wurde eloquent von Borges [Bor89] in seiner Kurzgeschichte „Die Bibliothek von Babel“ geschildert. Übrigens findet man wirklich jedes in Fußnote 1 erwähnte Stichwort im Sach- und Autorenverzeichnis. Sehen Sie mal nach.

Zugegeben, dieses Buch fokussiert scharf auf Theorie. Praktische Aspekte des *Security Engineering*, wozu etwa der Entwurf sicherer Public-Key-Infrastrukturen gehört, findet man hier nicht. Empfehlenswerte Referenzen für dieses Thema sind z.B. Buchmann [Buc01b] und Schneier [Sch96].

Während der Arbeit an der englischen Originalversion dieses Buches, 2003 und 2004, entwickelte eine Gruppe Düsseldorfer Studierender ein System, das eine Reihe von Kryptosystemen implementiert, welche auch hier behandelt werden. Danke möchte ich an dieser Stelle besonders Tobias Riege, der dieses Praktikum leitete, und auch Yves Jerschow, Claudia Lindner, Tim Schlüter, David Schneider, Andreas Stelzer, Philipp Stöcker, Alexander Tchernin, Pavel Tenenbaum, Oleg Uman斯基, Oliver Wollermann und Isabel Wolters. Den Quellcode in Java erhält man von <http://ccc.cs.uni-duesseldorf.de/~riege/praktikum>.

Überblick über die Buchkapitel

Kapitel 2 führt knapp in die Gebiete der Informatik und Mathematik ein, die für die in diesem Buch behandelten Themen der Komplexitätstheorie und Kryptologie relevant sind. Die verwendeten Begriffe werden so einfach wie möglich erklärt, aber auch mit der nötigen mathematischen Strenge. Insbesondere werden elementare Grundlagen der Algorithmik, der Theorie der formalen Sprachen, der Berechenbarkeitstheorie, der Logik, Algebra, Zahlentheorie, Graphentheorie und Wahrscheinlichkeitstheorie bereitgestellt. Obwohl jedes dieser Gebiete von Grund auf eingeführt und nicht viel an mathematischen Vorkenntnissen vom Leser verlangt wird, kann es hilfreich sein, wenn man mit den Grundlagen der Mathematik und Informatik bereits ein wenig vertraut ist.

In den Kapiteln 3 und 4 werden die Grundlagen der Komplexitätstheorie und Kryptologie gelegt, und ihre historische Entwicklung wird kurz skizziert. In Kapitel 3 werden die Komplexitätsmaße Zeit und Raum und die entsprechenden Komplexitätsklassen im traditionellen Worst-Case-Modell definiert. (Das Average-Case-Komplexitätsmodell wird hier nicht behandelt; für dieses Modell enthalten z.B. die

Zauberer Merlin, den *Einen Ring*, den *Heiligen Gral* oder *DNA-Tests* zu sagen hat. Oder man möchte alles über seine *Dogmen* erfahren.

exzellenten Arbeiten von Goldreich [Gol97a] und Wang [Wan97] viele nützliche Referenzen.) Grundlegende Eigenschaften der Worst-Case-Komplexität werden studiert, einschließlich der linearen Raumkompression und Beschleunigung sowie der Hierarchiesätze für Zeit und Raum. Die Beziehungen der wichtigsten Komplexitätsklassen zwischen logarithmischem und polynomialem Raum werden erkundet. Am namhaftesten unter diesen sind die Klassen P und NP, deterministische und nicht-deterministische polynomiale Zeit.

P kann man sich als eine Komplexitätsklasse vorstellen, die den intuitiven Begriff der effizienten Berechnung einfängt, wohingegen die härtesten Probleme in NP, die NP-vollständigen Probleme nämlich, als widerspenstig gelten, falls $P \neq NP$. Die P-versus-NP-Frage besteht darin, ob diese beiden Klassen verschieden sind oder nicht. Sie ist eine der wichtigsten offenen Fragen in der theoretischen Informatik, und sie hat die Komplexitätstheoretiker seit inzwischen mehr als dreißig Jahren geärgert. Falls $P \neq NP$ gilt, dann kann kein NP-vollständiges Problem effiziente (d.h. polynomialzeit-berechenbare) Algorithmen haben. Gilt jedoch $P = NP$, dann sind alle NP-Probleme in Polynomialzeit lösbar, und insbesondere können dann die meisten der derzeit verwendeten Kryptosysteme gebrochen werden.

Besondere Aufmerksamkeit wird in Kapitel 3 den komplexitätsbeschränkten Reduzierbarkeiten gewidmet, wie etwa der polynomialzeit-beschränkten Many-one-Reduzierbarkeit, und den darauf beruhenden Begriffen der Härte und Vollständigkeit. Reduzierbarkeiten sind mächtige, nützliche Werkzeuge für den Vergleich der Komplexität zweier gegebener Probleme, und Vollständigkeit erfasst die härtesten Probleme einer gegebenen Komplexitätsklasse bezüglich einer gegebenen Reduzierbarkeit. Insbesondere werden die vollständigen Probleme in den Klassen NL (nichtdeterministischer logarithmischer Raum) und NP intensiv untersucht, und sehr viele spezifische Beispiele natürlicher vollständiger Probleme in diesen Klassen werden angegeben. Dazu gehören auch verschiedene Varianten des Erfüllbarkeitsproblems, welches fragt, ob eine gegebene boolesche Formel erfüllbar ist. Die Liste der Probleme, deren NP-Vollständigkeit in diesem Kapitel gezeigt wird, enthält mehrere Graphenprobleme, wie etwa das Dreifärbbarkeitsproblem für Graphen, und auch bestimmte Varianten des Rucksack-Problems. In Kapitel 8 wird später ein Kryptosystem vorgestellt, das auf solch einem Rucksack-Problem beruht.

Es gibt Probleme in NP, die vermutlich weder NP-vollständig sind noch effiziente Algorithmen haben. Ein solches Beispiel ist das Graphisomorphie-Problem, das in Kapitel 2 eingeführt und in den Kapiteln 3, 6 und 8 tiefgründiger untersucht wird. Ein weiteres Beispiel eines Problems, das in nichtdeterministischer Polynomialzeit gelöst werden kann, von dem man aber nicht weiß, ob es in deterministischer Polynomialzeit lösbar ist, ist das Faktorisierungsproblem, das in Kapitel 7 gründlich untersucht wird. Die Sicherheit vieler Kryptosysteme, einschließlich des berühmten RSA-Systems, beruht auf der vermuteten Härte des Faktorisierungsproblems.

Kapitel 3 führt auch eine interessante Komplexitätsklasse ein, der vollständige Probleme vermutlich fehlen: UP, „Unambiguous Polynomial Time“, enthält genau die NP-Probleme, deren Instanzen nie mehr als eine Lösung haben. Die Komplexitätsklasse UP ist unter anderem nützlich für die Charakterisierung der Existenz bestimmter Typen von Einwegfunktionen im Worst-Case-Modell. Eine Funktion ist

eine Einwegfunktion, falls sie „leicht“ zu berechnen, aber „schwer“ zu invertieren ist. In der Komplexitätstheorie stehen solche Funktionen in einem engen Zusammenhang zur Isomorphie-Vermutung von Berman und Hartmanis. Einwegfunktionen (in einem adäquaten Komplexitätsmodell) sind auch in der Kryptographie wichtig; solche Funktionen werden in Kapitel 8 diskutiert.

Kapitel 4 führt die Grundbegriffe der Kryptologie ein, wie etwa symmetrische (alias *private-key*) und asymmetrische (alias *public-key*) Kryptosysteme. Dieses Kapitel stellt einige klassische symmetrische Kryptosysteme vor, einschließlich der Substitutions-, der affinen und der Permutationschiffre, der affin linearen Blockchiffren, der Stromchiffren, der Vigenère- und der Hill-Chiffre. Kryptoanalytische Angriffe auf diese Kryptosysteme werden anhand von Beispielen präsentiert. Außerdem wird der Begriff der perfekten Geheimhaltung für Kryptosysteme eingeführt, der auf dem Entropiebegriff im Sinne der Informations- und Codierungstheorie von Shannon [Sha49] beruht. Schließlich wird Shannons Resultat vorgestellt, das notwendige und hinreichende Bedingungen dafür angibt, dass ein Kryptosystem perfekte Geheimhaltung erreicht.

Kapitel 5 wendet sich wieder der Komplexitätstheorie zu und führt Hierarchien ein, die auf NP beruhen, einschließlich der booleschen Hierarchie über NP und der Polynomialzeit-Hierarchie. Im Zusammenhang damit werden verschiedene polynomialzeit-beschränkte Turing-Reduzierbarkeiten definiert. Diese Hierarchien enthalten beide NP als ihre erste Stufe und sind sehr nützlich für die Klassifizierung von Problemen, die vermutlich härter als NP-vollständige Probleme sind. Zu den Beispielen für Probleme, die in den höheren Stufen der booleschen Hierarchie vollständig sind, zählen „exakte“ Varianten von NP-vollständigen Optimierungsproblemen, Facettenprobleme und kritische Graphenprobleme. Zu den Beispielen für Probleme, die in den höheren Stufen der Polynomialzeit-Hierarchie vollständig sind, gehören bestimmte Varianten von NP-vollständigen Problemen, die durch eine beschränkte Anzahl von alternierenden längenbeschränkten Quantoren dargestellt werden können. Beispielsweise gehören dazu solche Probleme, die das Erfüllbarkeitsproblem verallgemeinern, indem sie nach dem Wahrheitsgehalt von quantifizierten booleschen Formeln mit einer beschränkten Anzahl von alternierenden existenziellen und universellen Quantoren fragen.

Im Zusammenhang damit wird in Kapitel 5 der Begriff der alternierenden Turingmaschine eingeführt, und die Klassen P und PSPACE werden bezüglich solcher Maschinen charakterisiert: Deterministische Polynomialzeit ist gleich alternierendem logarithmischem Raum, und deterministischer polynomialer Raum ist gleich alternierender Polynomialzeit. Das erstgenannte Resultat zeigt, dass alternierende Turingmaschinen ein vernünftiges Modell der Parallelberechnung sind, denn sie erfüllen das Kriterium von Cook, nach welchem parallele Zeit (in etwa) dasselbe ist wie sequenzieller (d.h. deterministischer) Raum. Aus dem letztgenannten Resultat folgt insbesondere, dass das Problem der quantifizierten booleschen Formeln mit unbeschränkter Anzahl von alternierenden längenbeschränkten Quantoren vollständig für PSPACE ist.

Es gibt einen bemerkenswerten Zusammenhang zwischen der Polynomialzeit-Hierarchie und der booleschen Hierarchie über NP: Wenn die boolesche Hierarchie

auf eine endliche Stufe kollabiert, so kollabiert auch die Polynomialzeit-Hierarchie. Weiter führt Kapitel 5 die Frage-Hierarchien über NP mit einer beschränkten Anzahl von Fragen sowie die Low- und die High-Hierarchie in NP ein. Die Low-Hierarchie kann als ein Maßstab verwendet werden, um die Komplexität solcher NP-Probleme zu messen, die vermutlich weder in P liegen noch NP-vollständig sind.

Kapitel 6 befasst sich mit randomisierten Algorithmen und probabilistischen Komplexitätsklassen. Insbesondere wird ein randomisierter Algorithmus für das NP-vollständige Erfüllbarkeitsproblem vorgestellt und analysiert, der zwar immer noch nur in Exponentialzeit läuft, aber schneller als der naive deterministische Algorithmus für dieses Problem ist. Außerdem werden Monte-Carlo- und Las-Vegas-Algorithmen und die probabilistischen Komplexitätsklassen PP („Probabilistic Polynomial Time“), RP („Random Polynomial Time“), ZPP („Zero-error Probabilistic Polynomial Time“) und BPP („Bounded-error Probabilistic Polynomial Time“) in Kapitel 6 eingeführt und gründlich untersucht. Indem man den Fehler eines solchen randomisierten Algorithmus „von ein halb weg beschränkt“, ist es möglich, eine sehr nützliche Technik zur Wahrscheinlichkeitsverstärkung anzuwenden, mittels derer man eine Fehlerwahrscheinlichkeit der Berechnung erreichen kann, die exponentiell klein in der Eingabegröße ist. Eine so kleine Fehlerwahrscheinlichkeit kann in den meisten praktischen Anwendungen (nahezu) unbeschadet vernachlässigt werden, besonders wenn man bedenkt, dass schon die Wahrscheinlichkeit eines Hardware-Fehlers größer sein kann. Wieder haben einige probabilistische Komplexitätsklassen (z.B. PP) vollständige Probleme, wohingegen andere (z.B. BPP) höchstwahrscheinlich keine vollständigen Probleme besitzen.

In Kapitel 6 werden außerdem die von Babai und Moran eingeführten Arthur-Merlin-Spiele studiert, welche man als „interaktive Beweissysteme mit öffentlichen Münzwürfen“ auffassen kann. Auch sie definieren eine Hierarchie von Komplexitätsklassen mittels alternierender längenbeschränkter Quantoren. Die Hauptergebnisse über die Arthur-Merlin-Hierarchie in Kapitel 6 sind erstens, dass diese Hierarchie auf eine endliche (nämlich die zweite) Stufe kollabiert, und zweitens, dass das Graphisomorphie-Problem in der zweiten Stufe dieser Hierarchie enthalten ist. Daraus folgt, dass das Graphisomorphie-Problem auch in der zweiten Stufe der Low-Hierarchie liegt und somit vermutlich nicht NP-vollständig ist.

Kapitel 7 stellt das RSA-Kryptosystem vor, das erste im öffentlichen Bereich entwickelte Public-Key-Kryptosystem, das auch heute noch in der Praxis weitverbreitet und in Gebrauch ist. Das RSA-Schema für digitale Signaturen, welches auf dem RSA-Kryptosystem beruht, wird ebenfalls präsentiert. Ein Protokoll für digitale Signaturen ermöglicht es Alice, ihre Nachrichten an Bob so zu unterschreiben, dass Bob verifizieren kann, dass tatsächlich sie die Senderin war, und ohne dass Erich Alice' Signatur fälschen kann. Des Weiteren werden zahlreiche kryptoanalytische Angriffe auf das RSA-System inspiziert und gründlich diskutiert, und für jeden solchen Angriff auf RSA werden mögliche und zweckmäßige Gegenmaßnahmen vorgeschlagen.

Im Zusammenhang mit dem RSA-System werden in Kapitel 7 das Faktorisierungsproblem und das Primzahl-Problem intensiv untersucht. Einerseits hängt die Sicherheit von RSA wesentlich von der vermuteten Härte des Faktorisierens großer

ganzer Zahlen ab. Andererseits erfordern sowohl das RSA-Kryptosystem als auch das RSA-Schema für digitale Signaturen, wie auch viele andere Kryptosysteme und Protokolle, die effiziente Erzeugung großer Primzahlen. Die Komplexität der prominentesten Faktorisierungsmethoden, zu denen z.B. das quadratische Sieb gehört, wird in Kapitel 7 diskutiert. Anzumerken ist, dass es für das Faktorisierungsproblem derzeit weder einen effizienten Algorithmus noch einen mathematisch strengen Beweis seiner Härte gibt.

Außerdem werden in Kapitel 7 eine Reihe von effizienten, in der Praxis verwendeten Primzahltests vorgestellt, einschließlich des Fermat-Tests, des Miller–Rabin-Tests und des Solovay–Strassen-Tests. Bei diesen handelt es sich um randomisierte Algorithmen, und einige von ihnen sind vom Monte-Carlo-Typ. Ein kürzlich erzieltes Resultat, nach dem das Primzahlproblem sogar in deterministischer Polynomialzeit gelöst werden kann, wird ebenfalls diskutiert.

Kapitel 8 inspiriert weitere wichtige Public-Key-Kryptosysteme und kryptographische Protokolle, einschließlich des Diffie–Hellman-Protokolls für den Tausch geheimer Schlüssel und des ElGamal-Protokolls für digitale Signaturen. Das letztere Protokoll ist, mit geeigneten Modifizierungen, als der Digital Signature Standard der Vereinigten Staaten übernommen worden. Im Zusammenhang mit diesen Protokollen wird auch das Problem des diskreten Logarithmus in diesem Kapitel sorgfältig studiert. Die Sicherheit vieler wichtiger Protokolle, wie z.B. der beiden eben erwähnten, beruht auf der vermuteten Härte dieses Problems.

Indem wir uns dann dem in den vorherigen Kapiteln studierten Graphisomorphie-Problem und dem Begriff der Arthur-Merlin-Spiele wieder zuwenden, wird in Kapitel 8 der Begriff des Zero-Knowledge-Protokolls eingeführt, der für die kryptographische Aufgabe der Authentifikation wichtig ist.

Es hat in der Vergangenheit Versuche gegeben, Kryptosysteme zu entwerfen, deren Sicherheit auf NP-harten Problemen beruht, insbesondere auf Varianten des Rucksack-Problems. Einige dieser Kryptosysteme wurden gebrochen, wohingegen andere noch immer intakt sind (also nach wie vor als sicher gelten). Ein solches Kryptosystem wird in Kapitel 8 vorgestellt und kritisch diskutiert. Im Zusammenhang damit wird der Begriff der Falltür-Einwegfunktion (*trapdoor one-way function*) diskutiert, der in der Public-Key-Kryptographie sehr wichtig ist. Schließlich werden in diesem Kapitel Protokolle für den Tausch geheimer Schlüssel und für digitale Signaturen vorgestellt, die auf assoziativen, stark nichtinvertierbaren Einwegfunktionen (im Worst-Case-Modell) beruhen.

Selbstverständlich gibt es noch viele weitere interessante Themen und Resultate in der Komplexitätstheorie und Kryptologie, auf die in diesem Buch zwar nicht eingegangen werden kann, für die aber ein paar empfehlenswerte Referenzen gegeben werden. Beispielsweise wird das Thema der Approximation und der Nichtapproximierbarkeit, welches sowohl von theoretischer als auch von praktischer Bedeutung ist, hier nicht angesprochen; siehe z.B. Ausiello et al. [ACG⁺03], Vazirani [Vaz03] und das umfassende, stets aktuell gehaltene Kompendium von NP-Optimierungsproblemen, herausgegeben von Crescenzi, Kann, Halldórsson, Karpiniski und Woeginger: <http://www.nada.kth.se/~viggo/problemList>.

Für eine Vielzahl weiterer wichtiger Themen der Komplexitätstheorie sei auf die folgenden Bücher verwiesen: Balcázar, Díaz und Gabarró [BDG95, BDG90], Bovet und Crescenzi [BC93], Du und Ko [DK00], Garey und Johnson [GJ79], L. Hemaspaandra und Ogiwara [HO02], Papadimitriou [Pap95], Reischuk [Rei90], Vollmer [Vol99], Wagner und Wechsung [WW86, Wec00] und Wegener [Weg87, Weg03], sowie auf die Sammelwerke, die herausgegeben wurden von Selman und L. Hemaspaandra [Sel90, HS97] und von Ambos-Spies, Homer und Schöning [AHS93].

Das Rechnen mit so genannten Quantencomputern wird hier nicht behandelt; Berthiaume [Ber97] führt in dieses faszinierende neue Gebiet ein, das über die Grenzen der klassischen Berechenbarkeits- und Komplexitätstheorie hinausgeht und auf den quantenmechanischen Prinzipien der Physik gründet. Einen verständlichen Einstieg in das verwandte Gebiet der Quantenkryptographie, welches sich gleichermaßen von der hier präsentierten klassischen Kryptographie beträchtlich unterscheidet, geben z.B. Bruß, Erdélyi, T. Meyer, Riege und Rothe [BEM⁺07] sowie Gisin, Ribordy, Tittel und Zbinden [GRTZ02]. Für andere Themen der Kryptologie, die hier nicht behandelt werden, sei beispielsweise verwiesen auf Goldreich [Gol99, Gol01], Luby [Lub96], Micciancio und Goldwasser [MG02], Salomaa [Sal96], Schneier [Sch96], Stinson [Sti05] und Welsh [Wel98].

Grundlagen der Informatik und Mathematik

Eine Sprache der Niederlande ist Holländisch. Eine Sprache des Lebens ist der genetische Code. Und eine Sprache der Natur und der Wissenschaften ist die Mathematik.¹ Die Begriffe, Resultate und Methoden der Komplexitätstheorie und der Kryptologie – wie auch vieler anderer (natur-)wissenschaftlicher Gebiete – können am präzisesten und am elegantesten in der Sprache der Mathematik formuliert werden.

Dieses Kapitel stellt kurz jene Gebiete der theoretischen Informatik und Mathematik vor, die für die in diesem Buch behandelten Komplexitätstheoretischen und kryptologischen Themen relevant sind. Die erforderlichen Konzepte werden mit mathematischer Strenge eingeführt und so einfach wie möglich erklärt, dabei aber so detailliert, wie es für das Verständnis nötig ist. Insbesondere werden die wichtigsten Grundlagen der Algorithmik, der formalen Sprachen und der Berechenbarkeitstheorie, der Logik, Algebra, Zahlentheorie, Graphentheorie und der Wahrscheinlichkeitstheorie zur Verfügung gestellt. Dieses Kapitel kann ebenso gut vorerst übersprungen und später erneut aufgesucht werden, wann immer das notwendig erscheint.

2.1 Algorithmen: Der Euklidische Algorithmus

Was ist ein Algorithmus? Wir werden uns bei dieser Frage, die durchaus auch einen philosophischen Aspekt hat, auf einen ganz pragmatischen Standpunkt stellen und sie hier erst einmal nur informell behandeln. Sicherlich hat jeder eine intuitive Vorstellung davon, was ein Algorithmus ist. Ein mathematisch präzises, formales Modell für den Begriff des Algorithmus, nämlich die Turingmaschine, wird später in Abschnitt 2.2 vorgestellt, siehe die Definitionen 2.15 und 2.16. Mit Turingmaschinen und anderen, äquivalenten Algorithmenmodellen kann man den Begriff der Berechenbarkeit von Funktionen und der Entscheidbarkeit von Problemen formal fassen.

Der Ausdruck „Algorithmus“ ist durch Sprachtransformation aus dem Namen des persisch-arabischen Wissenschaftlers Muhammed Ibn Musa Abu Djáfar al Cho-

¹ In Abwandlung des bekannten Slogans: „Springer—the language of science.“

resmi (773 bis 850) entstanden,² dem Hofmathematiker des Kalifen in Bagdad. Im Jahre 820 verfasste er das höchst einflussreiche Werk „On the Indian Numbers“ (von Adam Ries übersetzt als „Rechnung auf der Linie“), in dem das Dezimalsystem (einschließlich der Zahl Null) eingeführt wird.

Intuitiv ist ein Algorithmus eine endliche Folge von (durch einen endlichen Text beschreibbaren) Regeln oder Prozeduren, die zur Lösung eines Problems befolgt oder abgearbeitet werden müssen. Der Prozess der Anwendung dieser Regeln auf eine gegebene Eingabe kann nach einer endlichen Anzahl von Schritten erfolgreich terminieren, wobei die Eingabe akzeptiert oder in eine Ausgabe überführt wird, die die gegebene Probleminstanz löst. Oder dieser Prozess terminiert nach einer endlichen Schrittzahl ohne Erfolg, wobei die Eingabe verworfen oder abgelehnt wird. Es kann auch passieren, dass dieser Prozess nie terminiert und ein Algorithmus bei einer Eingabe endlos läuft. Diesen dritten Fall von den beiden erstgenannten Fällen (in denen der Algorithmus terminiert) zu unterscheiden, ist schwieriger, als man auf den ersten Blick vermuten würde. Die Aufgabe, diese Unterscheidung für einen gegebenen Algorithmus und eine gegebene Eingabe zu treffen, kann selbst als ein Problem formuliert werden, nämlich das so genannte Halteproblem: Gegeben ein Algorithmus (codiert als ein endlicher Text) und eine Eingabe, hält der Algorithmus bei dieser Eingabe jemals an? Ein fundamentales Resultat der Berechenbarkeitstheorie sagt, dass das Halteproblem „algorithmisch nicht lösbar ist“ [Tur36], was bedeutet, dass es beweisbar keinen Algorithmus gibt, der das Halteproblem löst. Dieses Ergebnis ist das erste in einer langen Reihe von so genannten Unentscheidbarkeitsresultaten, welche die Grenzen der algorithmischen Lösbarkeit und der Berechnungskraft von Computern aufzeigen.

```
EUKLID( $n, m$ ) {
    if ( $m = 0$ ) return  $n$ ;
    else return EUKLID( $m, n \bmod m$ );
}
```

Abb. 2.1. Euklidischer Algorithmus

Sehen wir uns ein Beispiel an. Einer der einfachsten und doch grundlegendsten Algorithmen ist seit der Antike bekannt und wird bereits im Buch der „Elemente“ von Euklid von Alexandria (etwa 325 bis 265 v. Chr.) erwähnt. Trotz seines Alters ist der Euklidische Algorithmus noch immer sehr wichtig, beispielsweise im Kapitel 7, in dem das populäre Kryptosystem RSA vorgestellt wird.

² Andere Schreibweisen seines Namens sind ebenfalls bekannt, etwa Abu Ja'far Mohammed ibn Musa Al-Khowarizmi (siehe [Sch02a]). Die Transformation seines Namens in „Algorithmus“ erfolgte mittels der lateinischen Redewendung „*dixit algorizmi*“, die man mit „Und so sprach al Choresmi“ übersetzen kann und die die Korrektheit einer Berechnung mit einer Art Gütesiegel versah.

$\mathbb{N} = \{0, 1, 2, \dots\}$ ist die Menge der natürlichen Zahlen und $\mathbb{Z} = \{0, \pm 1, \pm 2, \dots\}$ ist die Menge der ganzen Zahlen. Der Euklidische Algorithmus bestimmt den größten gemeinsamen Teiler von zwei gegebenen ganzen Zahlen m und n mit $m \leq n$, d.h., die größte Zahl $k \in \mathbb{N}$, für die es Zahlen $a, b \in \mathbb{Z}$ mit $m = a \cdot k$ und $n = b \cdot k$ gibt. Dieses k wird als $\text{ggT}(n, m)$ bezeichnet und vom Euklidischen Algorithmus ausgegeben, der als Pseudocode in Abbildung 2.1 dargestellt ist.

Der Euklidische Algorithmus modifiziert die gegebenen Zahlen, n und m , sukzessive, indem er sich selbst rekursiv mit den neuen Zahlen m (statt n) und $n \bmod m$ (statt m) aufruft; die Notation „ $n \bmod m$ “ und die Arithmetik modulo m werden in Problem 2.1 am Ende dieses Kapitels erklärt. EUKLID fährt mit dieser Rekursion fort, bis die Abbruchbedingung ($m = 0$) erreicht ist. Dann ist der aktuelle Wert der Variablen n der größte gemeinsame Teiler $\text{ggT}(n, m)$ der ursprünglich gegebenen Zahlen n und m , siehe (2.1).

Diese rekursive Darstellung des Euklidischen Algorithmus ist zweifellos elegant. Der Algorithmus kann jedoch auch iterativ implementiert werden. Das heißt, dass es keine rekursiven Aufrufe gibt, sondern die Zwischenwerte der Berechnung explizit mittels einer geeigneten Datenstruktur, etwa einem *Stack*, gespeichert werden.

Tabelle 2.1. Testlauf des Euklidischen Algorithmus

n	m	$n \bmod m$
170	102	68
102	68	34
68	34	0
34	0	

Tabelle 2.1 zeigt einen Testlauf des Euklidischen Algorithmus bei Eingabe von $n = 170$ und $m = 102$. In diesem Fall berechnet der Algorithmus tatsächlich die korrekte Lösung, da $\text{ggT}(170, 102) = 34$. Doch schon Edsger Dijkstra (1930 bis 2002) wusste, dass solche Tests für spezielle Eingaben lediglich die Gegenwart von Fehlern, nicht jedoch ihre Abwesenheit nachweisen können. Um die Korrektheit des Euklidischen Algorithmus zu beweisen, genügt es, die folgende Gleichheit zu zeigen:

$$\text{ggT}(n, m) = \text{ggT}(m, n \bmod m). \quad (2.1)$$

Für $r = n \bmod m$ und eine geeignete Zahl $s \in \mathbb{Z}$ gilt $n = s \cdot m + r$, wobei $0 \leq r < m$. Um (2.1) zu zeigen, genügt es festzustellen, dass jeder gemeinsame Teiler von n und m ebenso ein gemeinsamer Teiler von m und $r = n \bmod m$ ist und umgekehrt. Sei also k ein beliebiger gemeinsamer Teiler von m und $r = n \bmod m$ ist und umgekehrt. Sei also k ein beliebiger gemeinsamer Teiler von m und n . Dann gibt es Zahlen $a, b \in \mathbb{Z}$ mit $m = a \cdot k$ und $n = b \cdot k$. Setzt man diese Werte ein, so ergibt sich $b \cdot k = s \cdot m + r = s \cdot a \cdot k + r$, woraus $r = (b - s \cdot a)k$ folgt. Somit ist k auch ein Teiler von $r = n \bmod m$. Umgekehrt sei k nun ein beliebiger gemeinsamer Teiler von m und $r = n \bmod m$. Dann gibt es Zahlen $c, d \in \mathbb{Z}$ mit $m = c \cdot k$ und $r = d \cdot k$. Das Einsetzen dieser Werte

ergibt nun $n = s \cdot m + r = s \cdot c \cdot k + d \cdot k = (s \cdot c + d)k$. Folglich ist k auch ein Teiler von n , und (2.1) ist bewiesen. Der Euklidische Algorithmus ist somit korrekt.

Wie oben erwähnt, folgt der Euklidische Algorithmus einer rekursiven Strategie. Diese geht nach dem Prinzip Teile und Herrsche vor, denn die zu betrachtenden Zahlen werden mit jedem rekursiven Aufruf echt kleiner. Das allgemeine Schema von Teile und Herrsche kann wie folgt beschrieben werden:

1. **Teile** das Problem in paarweise disjunkte kleinere Teilprobleme desselben Typs.
2. **Herrsche** über das Problem durch rekursives Lösen dieser kleineren Teilprobleme.
3. **Vereinige** die Lösungen der Teilprobleme zu einer Lösung des ursprünglichen Problems.

Anders als die meisten anderen Teile-und-Herrsche-Algorithmen benötigt der Euklidische Algorithmus keinen „Vereinige“-Schritt, d.h., es ist hier nicht nötig, die Lösungen der kleineren Teilprobleme zu einer Gesamtlösung des gegebenen Problems zu vereinigen.

```
ERWEITERTER-EUKLID( $n, m$ ) {
    if ( $m = 0$ ) return ( $n, 1, 0$ );
    else {
        ( $g, x', y'$ ) := ERWEITERTER-EUKLID( $m, n \bmod m$ );
         $x := y'$ ;
         $y := x' - y' * \lfloor \frac{n}{m} \rfloor$ ;
        return ( $g, x, y$ );
    }
}
```

Abb. 2.2. Erweiterter Euklidischer Algorithmus

In der erweiterten Version des Euklidischen Algorithmus jedoch, die in Abbildung 2.2 dargestellt ist, wird der „Vereinige“-Schritt nicht weggelassen. Der erweiterte Euklidische Algorithmus berechnet eine Linearkombination der beiden gegebenen Zahlen, m und n , bestimmt also Zahlen x und y mit $\text{ggT}(n, m) = x \cdot n + y \cdot m$. Dieser Algorithmus ist ebenfalls sehr nützlich in einer Reihe von Anwendungen wie z.B. dem RSA-Kryptosystem in Kapitel 7. Der „Vereinige“-Schritt besteht hier darin, die Werte x und y aus den rekursiv berechneten Werten x' und y' zu bestimmen. Die Notation „ $\lfloor \frac{n}{m} \rfloor$ “ in Abbildung 2.2 bezeichnet die größte ganze Zahl, die n/m nicht überschreitet. Ebenso bezeichnet „ $\lceil \frac{n}{m} \rceil$ “ die kleinste ganze Zahl, die nicht kleiner als n/m ist.

Tabelle 2.2 zeigt einen Testlauf des erweiterten Euklidischen Algorithmus bei Eingabe von $n = 170$ und $m = 102$. Die beiden linken Spalten der Tabelle werden von oben nach unten wie beim (einfachen) Euklidischen Algorithmus gefüllt, wobei der Algorithmus ERWEITERTER-EUKLID in jedem Durchlauf rekursiv mit neuen,

kleineren Werten von n und m aufgerufen wird. In der letzten Zeile der Tabelle wird die Abbruchbedingung mit $n = 34$ und $m = 0$ erreicht. Nun erfolgt kein weiterer rekursiver Aufruf mehr, sondern der Algorithmus setzt $(g, x, y) := (34, 1, 0)$, und es werden die drei rechten Spalten der Tabelle von unten nach oben gefüllt, während ERWEITERTER-EUKLID aus den verschachtelten rekursiven Aufrufen nacheinander wieder auftaucht. Schließlich gibt ERWEITERTER-EUKLID(170, 102) das Tripel $(34, -1, 2)$ in der ersten Zeile der Tabelle aus. Dieses Ergebnis ist tatsächlich korrekt, denn

$$(-1) \cdot 170 + 2 \cdot 102 = 34 = \text{ggT}(170, 102).$$

Übung 2.1 verlangt einen Korrektheitsbeweis für ERWEITERTER-EUKLID.

Tabelle 2.2. Testlauf des erweiterten Euklidischen Algorithmus

n	m	g	x	y
170	102	34	-1	2
102	68	34	1	-1
68	34	34	0	1
34	0	34	1	0

Neben der Korrektheit ist die Laufzeit ein weiteres wichtiges Merkmal von Algorithmen. Ist der Algorithmus effizient? Oder hat er für bestimmte „harte“ Probleminstanzen oder sogar im Mittel eine übermäßig lange Laufzeit, bis er ein Ergebnis liefert? Die Laufzeiten des Euklidischen Algorithmus und seiner erweiterten Version sind im Wesentlichen gleich; daher beschränken wir uns darauf, die erstere zu analysieren. Offenbar genügt es, die Zahl der rekursiven Aufrufe des Euklidischen Algorithmus abzuschätzen, um seine Laufzeit zu analysieren. Zu diesem Zweck werden nun einige Vorbereitungen getroffen.

Definition 2.1 (Fibonacci-Zahlen). Die Folge $\mathfrak{F} = (f_n)_{n \geq 0}$ der Fibonacci-Zahlen ist induktiv definiert durch:

$$\begin{aligned} f_0 &= 0, \\ f_1 &= 1, \\ f_n &= f_{n-1} + f_{n-2} \quad \text{für } n \geq 2. \end{aligned}$$

Mathematisch ausgedrückt sind die Fibonacci-Zahlen durch eine homogene lineare Rekurrenzgleichung zweiten Grades bestimmt. Das heißt, die Elemente der Folge \mathfrak{F} haben die folgende Form:

$$\begin{aligned} T(0) &= r, \\ T(1) &= s, \\ T(n) &= p \cdot T(n-1) + q \cdot T(n-2) \quad \text{für } n \geq 2, \end{aligned} \tag{2.2}$$

wobei p, q, r und s reelle Konstanten mit $p \neq 0$ und $q \neq 0$ sind.

Satz 2.2. Die Lösung der Rekurrenzgleichung (2.2) hat die Form:

$$T(n) = \begin{cases} A \cdot \alpha^n - B \cdot \beta^n & \text{falls } \alpha \neq \beta \\ (A \cdot n + B) \alpha^n & \text{falls } \alpha = \beta, \end{cases}$$

wobei α und β die beiden reellen Lösungen der quadratischen Gleichung

$$a^2 - p \cdot a - q = 0$$

und wobei die Zahlen A und B wie folgt definiert sind:

$$A = \begin{cases} \frac{s-r\beta}{\alpha-\beta} & \text{falls } \alpha \neq \beta \\ \frac{s-r\alpha}{\alpha} & \text{falls } \alpha = \beta \end{cases} \quad \text{und} \quad B = \begin{cases} \frac{s-r\alpha}{\alpha-\beta} & \text{falls } \alpha \neq \beta \\ r & \text{falls } \alpha = \beta. \end{cases}$$

Der Beweis von Satz 2.2 wird dem Leser als Übung 2.2 überlassen. Im Fall der Fibonacci-Zahlen erfüllen die Konstanten $p = q = s = 1$ und $r = 0$. Die ersten zwanzig Werte der Folge \mathfrak{F} sind in Tabelle 2.3 aufgeführt.

Tabelle 2.3. Die ersten zwanzig Fibonacci-Zahlen

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
f_n	0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181

Das rapide Wachstum der ersten Fibonacci-Zahlen in Tabelle 2.3 legt nahe, dass die Folge $\mathfrak{F} = \{f_n\}_{n \geq 0}$ exponentiell wächst. Bevor wir diese Vermutung beweisen, sehen wir uns ein illustratives Beispiel an. Denkt man an exponentielles Wachstum, so kommt einem vielleicht das Vermehrungsverhalten von Kaninchen in den Sinn. Und das war tatsächlich die ursprüngliche Motivation der Untersuchungen von Leonardo Pisano (1170 bis 1250), dessen Spitzname Fibonacci war. Die heute nach ihm benannte Folge sollte eine mathematische Beschreibung der Reproduktion von Kaninchen liefern, allerdings unter gewissen vereinfachenden Annahmen. In Fibonaccis Modell gebiert jedes Kaninchen einen Nachkommen pro Monat, außer im ersten und zweiten Monat seiner Lebenszeit. Weiter wird in Abstraktion von der Wirklichkeit angenommen, dass alle Kaninchen dasselbe Geschlecht haben und unsterblich sind, auch haben sie keine natürlichen Feinde. Beginnt man eine Kaninchenzucht mit nur einem Exemplar, so ist man stolzer Besitzer einer Population von genau f_n Kaninchen nach n Monaten. Tabelle 2.4 zeigt diesen Vermehrungsprozess für ein Anfangsstück der Folge \mathfrak{F} .

Nun beweisen wir unsere Vermutung, dass die Folge $\mathfrak{F} = \{f_n\}_{n \geq 0}$ exponentiell in n wächst. Genauer zeigen wir durch Induktion über n , dass für geeignete Konstanten a und c und für alle hinreichend großen n gilt:

$$f_n \geq c \cdot a^n. \tag{2.3}$$

Die Induktionsbasis ist trivial. Setzen wir nun die Induktionsannahme in die Rekurrenzgleichung für f_n ein, so erhalten wir:

Tabelle 2.4. Die Fibonacci-Zahlen vermehren sich wie die Kaninchen und umgekehrt

0. Monat		$f_0 = 0$
1. Monat		$f_1 = 1$
2. Monat		$f_2 = 1$
3. Monat		$f_3 = 2$
4. Monat		$f_4 = 3$
5. Monat		$f_5 = 5$
6. Monat		$f_6 = 8$
7. Monat		$f_7 = 13$
8. Monat		$f_8 = 21$
9. Monat		$f_9 = 34$
10. Monat		$f_{10} = 55$

$$f_n = f_{n-1} + f_{n-2} \geq c \cdot a^{n-1} + c \cdot a^{n-2} = c \cdot a^n \cdot \frac{a+1}{a^2} \geq c \cdot a^n.$$

Der Induktionsschritt ist vollendet, wenn wir die letzte Ungleichung oben zeigen können, welche äquivalent ist zu $a^2 - a - 1 \leq 0$. Eine quadratische Gleichung der Form $a^2 + p \cdot a + q = 0$ hat die beiden reellen Lösungen $-p/2 \pm \sqrt{p^2/4 - q}$. In unserem Fall ($a^2 - a - 1 = 0$) ergibt sich also $p = -1 = q$, und wir erhalten die folgenden Lösungen:

$$\alpha = \frac{1}{2} + \sqrt{\frac{1}{4} + 1} = \frac{1 + \sqrt{1+4}}{2} = \frac{1 + \sqrt{5}}{2} \approx 1.618,$$

$$\beta = \frac{1}{2} - \sqrt{\frac{1}{4} + 1} = \frac{1 - \sqrt{1+4}}{2} = \frac{1 - \sqrt{5}}{2} \approx -0.618.$$

Somit ist der Induktionsschritt für alle a mit $\beta \leq a \leq \alpha$ gezeigt.

Die Zahl $\alpha = (1 + \sqrt{5}) / 2$ tritt in verschiedenen Zusammenhängen in der Mathematik und Informatik auf. In der Geometrie beispielsweise ergibt sich diese Zahl, wenn man ein gegebenes Rechteck in ein Quadrat und ein kleineres Rechteck derart zerlegt, dass die Seitenlängen der beiden Rechtecke dasselbe Verhältnis haben. Diese Zerlegung wird als der „goldene Schnitt“ bezeichnet und detailliert in Übung 2.3 beschrieben.

Nach Satz 2.2 liefern die Zahlen α und β die Werte $A = 1/\sqrt{5}$ und $B = 1/\sqrt{5}$, woraus für die n -te Fibonacci-Zahl

$$f_n = \frac{1}{\sqrt{5}} \left(\frac{1 + \sqrt{5}}{2} \right)^n - \frac{1}{\sqrt{5}} \left(\frac{1 - \sqrt{5}}{2} \right)^n \quad (2.4)$$

folgt. Der zweite Term in (2.4), $-1/\sqrt{5} \left((1-\sqrt{5})/2 \right)^n$, kann vernachlässigt werden, da $\beta = (1-\sqrt{5})/2$ einen Absolutbetrag kleiner als 1 hat und der Term $1/\sqrt{5} \left((1-\sqrt{5})/2 \right)^n$ daher für wachsendes n gegen null geht. Analog kann man zeigen, dass für geeignete Konstanten d und y und für alle hinreichend großen n gilt:

$$f_n \leq d \cdot y^n.$$

Als nächstes zeigen wir, dass die Folge $\mathfrak{F} = \{f_n\}_{n \geq 0}$ der Fibonacci-Zahlen eng mit der Anzahl der rekursiven Aufrufe des Euklidischen Algorithmus zusammenhängt. Satz 2.3 bestimmt Zahlen, aus denen sich die Worst-Case-Laufzeit des Euklidischen Algorithmus ergibt.

Satz 2.3. Für jedes $k \geq 1$ gelten die folgenden beiden Aussagen.

1. EUKLID(f_{k+3}, f_{k+2}) erfordert genau $k + 1$ rekursive Aufrufe.
2. Wenn EUKLID(n, m) wenigstens $k + 1$ rekursive Aufrufe macht, dann gilt $|n| \geq f_{k+3}$ und $|m| \geq f_{k+2}$.

Beweis. Ohne Beschränkung der Allgemeinheit dürfen wir annehmen, dass die Eingabezahlen n und m in der zweiten Aussage des Satzes aus \mathbb{N} sind, d.h., wir müssen keine negativen Werte betrachten und können auf Absolutbeträge verzichten. Beide Aussagen des Satzes werden durch Induktion über k gezeigt.

Induktionsbasis: $k = 1$. Es gilt $f_{k+3} = f_4 = 3$ und $f_{k+2} = f_3 = 2$. Da EUKLID(3, 2) rekursiv EUKLID(2, 1) aufruft, dieses rekursiv EUKLID(1, 0) aufruft, welches dann ohne weiteren rekursiven Aufruf terminiert, gilt die erste Aussage für $k = 1$. Die zweite Aussage ist ebenfalls wahr für $k = 1$, weil EUKLID(n, m) für alle $n < 3$ und $m < 2$ höchstens einen rekursiven Aufruf macht.

Induktionsschritt: $(k - 1) \mapsto k$. Sei $k \geq 2$. Nach Definition der f_n gilt $f_n \geq 1$ für alle $n \geq 1$, und somit gilt $f_n > f_{n-1}$ für alle $n \geq 3$. Also gilt für $n \geq 1$:

$$f_{n+2} < f_{n+3} = f_{n+2} + f_{n+1} < 2f_{n+2},$$

woraus für $k \geq 2$ folgt, dass f_{k+3} nicht durch f_{k+2} teilbar ist. Da $f_{k+3} \bmod f_{k+2} = f_{k+1}$ gilt, ruft EUKLID(f_{k+3}, f_{k+2}) rekursiv EUKLID(f_{k+2}, f_{k+1}) auf. Nach Induktionsannahme benötigt die Berechnung von EUKLID(f_{k+2}, f_{k+1}) genau k weitere rekursive Aufrufe. Insgesamt braucht EUKLID(f_{k+3}, f_{k+2}) also genau $k + 1$ rekursive Aufrufe, womit die erste Aussage bewiesen ist.

Um die zweite Aussage zu beweisen, nehmen wir an, dass die Berechnung von EUKLID(n, m) mindestens $k + 1$ rekursive Aufrufe macht, wobei $k \geq 2$. Der erste Aufruf ist EUKLID($m, n \bmod m$). Nach Induktionsannahme gilt $m \geq f_{k+2}$ und $(n \bmod m) \geq f_{k+1}$. Es bleibt $n \geq f_{k+3}$ zu zeigen. Da $n \geq m$ und da $m \neq 0$ ist (denn andernfalls wäre EUKLID(n, m) nicht aufgerufen worden), folgt

$$n \geq m + (n \bmod m) \geq f_{k+2} + f_{k+1} = f_{k+3},$$

was den Satz beweist. □

Ist $g : \mathbb{N} \rightarrow \mathbb{N}$ eine gegebene Funktion, so definiere die Funktionenklasse

$$\mathcal{O}(g) = \{f : \mathbb{N} \rightarrow \mathbb{N} \mid (\exists c > 0) (\exists n_0 \in \mathbb{N}) (\forall n \geq n_0) [f(n) \leq c \cdot g(n)]\}.$$

(Bezeichnungen wie der Existenzquantor \exists und der Allquantor \forall werden in Abschnitt 2.3 erklärt.)

Eine Funktion $f \in \mathcal{O}(g)$ wächst asymptotisch nicht schneller als g . Die \mathcal{O} -Notation wird üblicherweise bei der Angabe und Analyse der Laufzeiten von Algorithmen benutzt. Sie vernachlässigt additive Konstanten, konstante Faktoren und endlich viele Ausnahmen. Insbesondere ist die Basis des Logarithmus in der \mathcal{O} -Notation irrelevant, siehe Übung 2.4. Deshalb legen wir per Konvention fest, dass der Logarithmus \log stets zur Basis 2 sein möge, wobei wir die Basis 2 nicht extra angeben. Weitere Bezeichnungen für asymptotische Wachstumsraten werden in Abschnitt 3.2 zur Verfügung gestellt, siehe Definition 3.5.

Kehren wir zur Analyse von EUKLID zurück. Satz 2.3 sagt, dass die Zahl der rekursiven Aufrufe von $\text{EUKLID}(n, m)$ durch $1 + \max\{k \mid f_{k+3} \leq n\}$ beschränkt ist. Wir wissen aus (2.3), dass $f_{k+3} \geq c \cdot \alpha^{k+3}$ für $\alpha = \frac{1+\sqrt{5}}{2} \approx 1.618$ und eine geeignete Konstante c gilt. Logarithmieren wir die linke und die rechte Seite der Ungleichung $c \cdot \alpha^{k+3} \leq f_{k+3} \leq n$ zur Basis α , so erhalten wir:

$$\log_\alpha c + \log_\alpha \alpha^{k+3} = \log_\alpha c + k + 3 \leq \log_\alpha n.$$

Es folgt aus den obigen Betrachtungen, dass die Zahl k der rekursiven Aufrufe von $\text{EUKLID}(n, m)$ in $\mathcal{O}(\log n)$ ist. Die Eingabezahlen n und m in Binärdarstellung haben jeweils $b = \lfloor \log n \rfloor + 1$ Bits; siehe den Absatz gleich nach Definition 2.4. Eine Iteration (ohne rekursiven Aufruf) von EUKLID hat die Bitkomplexität $\mathcal{O}(b^2)$, die im Wesentlichen von der Division bei der Berechnung von $\text{ggT}(n, m)$ herrührt; also ergibt sich eine Bitkomplexität von $\mathcal{O}(\log n) \cdot \mathcal{O}(b^2) = \mathcal{O}(b^3)$ für die gesamte Berechnung. (Tatsächlich kann man sogar zeigen, dass die Bitkomplexität von EUKLID in $\mathcal{O}(b^2)$ liegt, siehe [CLRS01, Sch01].) Somit ist EUKLID ein effizienter Algorithmus.

2.2 Formale Sprachen und Berechenbarkeitstheorie

In diesem Abschnitt werden die Grundlagen der Theorie der formalen Sprachen und Automaten und der Berechenbarkeitstheorie gelegt. Auf Beweise verzichten wir in diesem Abschnitt.

Definition 2.4 (Alphabet, Wort und Sprache). Ein Alphabet ist eine endliche, nichtleere Menge Σ , deren Elemente Buchstaben oder Symbole heißen. Ein Wort über Σ ist eine endliche Folge von Buchstaben aus Σ . Die Menge aller Wörter über Σ wird mit Σ^* bezeichnet.

Die Länge eines Wortes $w \in \Sigma^*$, bezeichnet durch $|w|$, ist die Anzahl der Symbole, die in w vorkommen. Das leere Wort, bezeichnet durch ε , ist das eindeutig bestimmte Wort der Länge null. Jede Teilmenge von Σ^* ist eine (formale) Sprache (über Σ).

Die Kardinalität einer Sprache $L \subseteq \Sigma^*$ ist die Anzahl ihrer Wörter und wird mit $\|L\|$ bezeichnet.

Probleme werden als Sprachen über einem fixierten Alphabet codiert; typischerweise arbeiten wir mit dem binären Alphabet $\Sigma = \{0, 1\}$. Zahlen werden als Wörter über einem fixierten Alphabet codiert. Insbesondere bezeichnet $\text{bin}(n)$ die Binärdarstellung einer Zahl $n \in \mathbb{N}$ ohne führende Nullen. Beispielsweise ist $\text{bin}(19) = 10011$.

Zusätzlich zu den üblichen mengentheoretischen Operationen wie Vereinigung, Durchschnitt und Komplement, die natürlich auch auf Sprachen angewandt werden können, werden nun einige weitere Grundoperationen auf Wörtern und Sprachen definiert.

Definition 2.5 (Operationen auf Wörtern und Sprachen).

- Seien A und B Sprachen über einem Alphabet Σ . Definiere den Durchschnitt von A und B als $A \cap B = \{x \in \Sigma^* \mid x \in A \text{ und } x \in B\}$, die Vereinigung von A und B als $A \cup B = \{x \in \Sigma^* \mid x \in A \text{ oder } x \in B\}$ und das Komplement von A (bezüglich Σ^*) als $\overline{A} = \{x \in \Sigma^* \mid x \notin A\}$.
- Seien $u = u_1 u_2 \cdots u_m$ und $v = v_1 v_2 \cdots v_n$ zwei Wörter über einem Alphabet Σ . Die Konkatenation von u und v ist definiert als $uv = u_1 u_2 \cdots u_m v_1 v_2 \cdots v_n$. Beachte, dass mit $m = 0$ bzw. $n = 0$ die Wörter u bzw. v auch leer sein können.
- Die Konkatenation zweier Sprachen $A \subseteq \Sigma^*$ und $B \subseteq \Sigma^*$ ist definiert als

$$AB = \{ab \mid a \in A \text{ und } b \in B\}.$$

- Die Iteration einer Sprache $A \subseteq \Sigma^*$ (auch als die Kleene-Hülle von A bezeichnet) ist die Sprache A^* , die induktiv definiert ist durch

$$A^0 = \{\varepsilon\}, \quad A^n = AA^{n-1} \quad \text{und} \quad A^* = \bigcup_{n \geq 0} A^n.$$

Definiere die ε -freie Iteration von A als $A^+ = \bigcup_{n \geq 1} A^n$. Es gilt $A^+ = A^* - \{\varepsilon\}$, falls $\varepsilon \notin A$.

Wenn man eine neue Sprache erlernen will, etwa Holländisch, so muss man zunächst das Vokabular und die Grammatik der Sprache lernen. Das Vokabular ist einfach die Menge aller Wörter in dieser Sprache. Die Grammatik ist eine Liste von Regeln, die spezifizieren, wie man Wörter und Wortgruppen kombinieren kann und geeignete Satzzeichen so setzt, dass syntaktisch korrekte Sätze entstehen. Später lernt man, wie man semantisch korrekte (oder sogar sinnvolle) Sätze bilden kann. Doch hier beschäftigen wir uns nur mit der Syntax von Sprachen. Betrachte beispielsweise die Sprache $L = \{a^n b^n \mid n \in \mathbb{N}\}$, die aus den Wörtern besteht, die mit einem Block von n aufeinander folgenden a -Symbolen beginnen, gefolgt von einem Block der Länge n von b -Symbolen, für irgendein $n \geq 0$. Beachte, dass wegen $n = 0$ auch das leere Wort ε zu L gehört.

Eine formale Sprache wie L hat eines mit einer natürlichen Sprache wie Holländisch gemein: Beide brauchen einen Grammatik, die ihre Syntax spezifiziert.

Definition 2.6 (Grammatik). Eine Grammatik ist ein Quadrupel $G = (\Sigma, \Gamma, S, R)$, wobei Σ und Γ disjunkte Alphabete sind (d.h., $\Sigma \cap \Gamma = \emptyset$), $S \in \Gamma$ ist das Startsymbol und $R \subseteq (\Sigma \cup \Gamma)^+ \times (\Sigma \cup \Gamma)^*$ ist die endliche Menge von Regeln (oder Produktionen). Die Symbole in Σ heißen Terminalzeichen (oder kurz Terminal); für sie werden üblicherweise Kleinbuchstaben verwendet. Die Symbole in Γ heißen Nichtterminal (oder Variablen); sie werden durch Großbuchstaben bezeichnet. Regeln (p, q) in R werden auch so geschrieben: $p \rightarrow q$.

Nun wird erklärt, wie man Wörter ableiten kann, indem man die Regeln der Grammatik anwendet, und es wird die Sprache einer Grammatik definiert. Die Wörter einer solchen Sprache enthalten nur Symbole aus dem Terminalalphabet.

Definition 2.7. Sei $G = (\Sigma, \Gamma, S, R)$ eine Grammatik und seien $u, v \in (\Sigma \cup \Gamma)^*$.

- Definiere die (unmittelbare) Ableitungsrelation bezüglich G , bezeichnet mit \vdash_G , durch

$$u \vdash_G v \iff u = xpy \text{ und } v = xqy \text{ für } x, y \in (\Sigma \cup \Gamma)^* \text{ und für eine Regel } p \rightarrow q \text{ in } R.$$

Wir schreiben $u \vdash_G^n v$, falls $u = x_0 \vdash_G x_1 \vdash_G \dots \vdash_G x_n = v$ für $n \geq 0$ und $x_0, x_1, \dots, x_n \in (\Sigma \cup \Gamma)^*$. Insbesondere ist $u \vdash_G^0 u$.

- Definiere $\vdash_G^* = \bigcup_{n \geq 0} \vdash_G^n$ als die reflexive und transitive Hülle von \vdash_G , d.h., \vdash_G^* ist die kleinste reflexive, transitive Binärrelation auf $(\Sigma \cup \Gamma)^*$, die \vdash_G umfasst.
- Ein Wort, das nur aus Terminalzeichen besteht, heißt Terminalwort; oft wird es der Einfachheit halber als Wort bezeichnet, und zur Unterscheidung nennt man ein Wort aus $(\Sigma \cup \Gamma)^*$ (das also sowohl Terminal als auch Nichtterminal enthalten kann) eine Satzform. Die Sprache von G besteht aus allen aus dem Startsymbol ableitbaren Wörtern und ist definiert als

$$L(G) = \{w \in \Sigma^* \mid S \vdash_G^* w\}.$$

Beispiel 2.8 (Grammatik). Betrachte die folgenden einfachen Grammatiken.

- $G_1 = (\Sigma_1, \Gamma_1, S_1, R_1)$ sei die folgende Grammatik: das Terminalalphabet ist $\Sigma_1 = \{a, b\}$, das Nichtterminalalphabet ist $\Gamma_1 = \{S_1\}$, und die Regelmenge ist gegeben durch $R_1 = \{S_1 \rightarrow aS_1b, S_1 \rightarrow \epsilon\}$. Es ist nicht schwer zu sehen, dass G_1 die Sprache $L = \{a^n b^n \mid n \in \mathbb{N}\}$ erzeugt, d.h., $L(G_1) = L$; siehe Übung 2.5.
- $G_2 = (\Sigma_2, \Gamma_2, S_2, R_2)$ sei die folgende Grammatik:
 - das Terminalalphabet ist $\Sigma_2 = \{\text{by, eaten, fox, rabbit, the, was}\}$,
 - das Nichtterminalalphabet ist $\Gamma_2 = \{S_2, S, P, O, A, N, H, V, P'\}$, und
 - die Regelmenge ist gegeben durch

$$R_2 = \left\{ \begin{array}{l} S_2 \rightarrow SPO, \quad S \rightarrow AN, \quad P \rightarrow HV, \quad O \rightarrow P'AN, \quad H \rightarrow \text{was}, \\ V \rightarrow \text{eaten}, \quad N \rightarrow \text{fox}, \quad N \rightarrow \text{rabbit}, \quad P' \rightarrow \text{by}, \quad A \rightarrow \text{the} \end{array} \right\}.$$

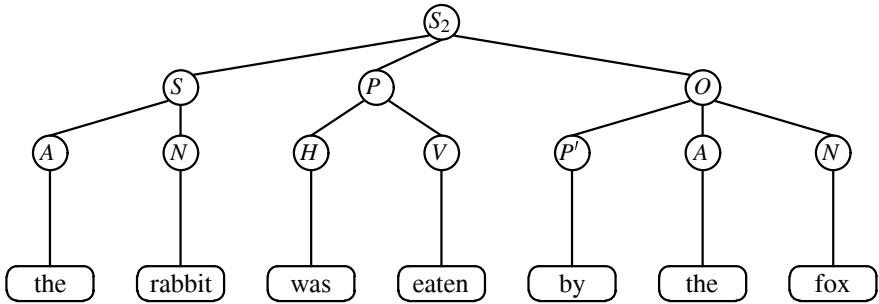


Abb. 2.3. Syntaxbaum für eine Ableitung in Beispiel 2.8

Englische Vokabeln wie etwa „fox“, die im Terminalalphabet vorkommen, werden hier als nur *ein* Terminalsymbol aufgefasst. Die Nichtterminale haben die folgende intuitive Bedeutung: Aus dem Startsymbol S_2 soll ein (englischer) „Satz“ abgeleitet werden, S steht für „Subjekt“, P für „Prädikat“, O für „Objekt“, A für „Artikel“, N für „Nomen“, H für „Hilfsverb“, V für „Verb“ und P' für „Präposition“.

Offenbar kann man durch Anwenden geeigneter Regeln der Grammatik aus dem Startsymbol S_2 den folgenden englischen „Satz“ ableiten:

the rabbit was eaten by the fox,

der eine Folge von sieben Terminalen darstellt und somit ein Wort in $L(G_2)$ ist, der Sprache von G_2 . Abbildung 2.3 zeigt den Syntaxbaum für diese Ableitung. Natürlich könnte ebenso

the fox was eaten by the rabbit

aus S_2 abgeleitet werden und wäre daher ebenfalls ein Wort in $L(G_2)$; siehe Übung 2.5. Ob dieses Wort, das einen englischen Satz darstellt, eine wahre oder plausible Aussage ausdrückt oder nicht, ist hier vollkommen unwichtig; wir interessieren uns ausschließlich für die Syntax von Grammatiken bzw. Sprachen.

Der Prozess des Ableitens von Wörtern ist inhärent nichtdeterministisch, da in jedem Ableitungsschritt womöglich mehr als eine Regel anwendbar ist. Mit Syntaxbäumen kann man eine konkrete Ableitung visualisieren, siehe Abbildung 2.3. Verschiedene Grammatiken können dieselbe Sprache erzeugen. Es gibt sogar für jede Sprache L , die überhaupt von einer Grammatik erzeugbar ist (d.h., für jede Sprache L in \mathcal{L}_0 , siehe Definition 2.9), unendlich viele verschiedene Grammatiken, die L erzeugen. Zwei Grammatiken heißen *äquivalent*, falls sie dieselbe Sprache erzeugen. Das heißt, eine Grammatik ist ein syntaktisches Objekt, das mittels der in Definition 2.7 eingeführten Begriffe ein semantisches Objekt spezifiziert, nämlich die von ihr erzeugte Sprache.

Grammatiken können klassifiziert werden, indem man ihre Regeln bestimmten Einschränkungen unterwirft. So erhält man z.B. die unten definierte Hierarchie von Sprachklassen, die so genannte Chomsky-Hierarchie.

Definition 2.9 (Chomsky-Hierarchie). Sei $G = (\Sigma, \Gamma, S, R)$ eine Grammatik.

- G ist eine Grammatik vom Typ 0, falls den Regeln in R keinerlei Einschränkung auferlegt wird.
- G ist eine Grammatik vom Typ 1 (oder auch eine kontextsensitive Grammatik), falls sämtliche Regeln $p \rightarrow q$ in R nichtverkürzend sind, d.h., es gilt $|p| \leq |q|$.
- G ist eine Grammatik vom Typ 2 (oder auch eine kontextfreie Grammatik), falls für sämtliche Regeln $p \rightarrow q$ in R gilt: $p \in \Gamma$.
- G ist eine Grammatik vom Typ 3 (oder auch eine rechtslineare oder reguläre Grammatik), falls für sämtliche Regeln $p \rightarrow q$ in R gilt: $p \in \Gamma$ und $q \in \Sigma \cup \Sigma\Gamma$.
- Eine Sprache $L \subseteq \Sigma^*$ ist vom Typ $i \in \{0, 1, 2, 3\}$, falls es eine Grammatik G vom Typ i mit $L(G) = L$ gibt.
- Definiere für $i \in \{0, 1, 2, 3\}$ die Sprachklasse

$$\mathcal{L}_i = \{L(G) \mid G \text{ ist eine Grammatik vom Typ } i\}.$$

Die Chomsky-Hierarchie besteht aus diesen vier Sprachklassen, die man üblicherweise wie folgt bezeichnet:

- \mathcal{L}_0 ist die Klasse der Sprachen, die von einer beliebigen Grammatik erzeugt werden können;
- $\mathcal{L}_1 = \text{CS}$ ist die Klasse der kontextsensitiven Sprachen;
- $\mathcal{L}_2 = \text{CF}$ ist die Klasse der kontextfreien Sprachen;
- $\mathcal{L}_3 = \text{REG}$ ist die Klasse der regulären Sprachen.

Offenbar sind beide Grammatiken aus Beispiel 2.8 kontextfrei. Der Ausdruck „kontextfrei“ für Typ-2-Grammatiken kommt daher, dass bei Anwendung von Regeln der Form $A \rightarrow q$, wobei A ein Nichtterminal ist, A durch q ersetzt wird, unabhängig vom Kontext von A in der bisher abgeleiteten Satzform. Ähnlich steht der Ausdruck „kontextsensitiv“ bei Typ-1-Grammatiken für den Fakt, dass eine jede Grammatik, deren Regeln nichtverkürzend sind (also die Form $p \rightarrow q$ mit $|p| \leq |q|$ haben), in eine äquivalente Grammatik $G = (\Sigma, \Gamma, S, R)$ umgeformt werden kann, deren Regeln die Form $uAv \rightarrow uwv$ mit $A \in \Gamma$, $u, v, w \in (\Sigma \cup \Gamma)^*$ und $w \neq \epsilon$ haben. Das heißt, das Nichtterminal A kann nur im Kontext von u und v durch w ersetzt werden, wenn eine solche Regel aus G angewandt wird.

Es ist nicht schwer zu sehen, dass die Chomsky-Hierarchie die in Fakt 2.10 angegebene Inklusionsstruktur besitzt. Alle diese Inklusionen sind echt, siehe Satz 2.21. So ist z.B. die in Beispiel 2.8 definierte kontextfreie Sprache $L = \{a^n b^n \mid n \in \mathbb{N}\}$ nicht regulär, woraus $\text{REG} \neq \text{CF}$ folgt; siehe Problem 2.2.

Fakt 2.10 $\text{REG} \subseteq \text{CF} \subseteq \text{CS} \subseteq \mathcal{L}_0$.

Unter den Klassen der Chomsky-Hierarchie ist die Klasse der kontextfreien Sprachen in der Informatik besonders wichtig, beispielsweise im Compilerbau für Programmiersprachen. Dieses Thema soll hier jedoch nicht weiter verfolgt werden. Stattdessen wenden wir uns nun der Theorie der Automaten (die ebenfalls für den Compilerbau wichtig ist) und der Berechenbarkeitstheorie zu; beide sind eng verwandt mit der Theorie der formalen Sprachen. So kann etwa jede Klasse der

Chomsky-Hierarchie durch einen geeigneten Automatentyp charakterisiert werden. Für die kleinste Klasse der Hierarchie, $\mathcal{L}_3 = \text{REG}$, weiß man zum Beispiel, dass jede reguläre Sprache von einem endlichen Automaten erkannt werden kann, und jede von einem endlichen Automaten erkennbare Sprache ist regulär. Wir beginnen mit der Einführung des Begriffs eines deterministischen endlichen Automaten.

Definition 2.11 (Deterministischer endlicher Automat). Ein deterministischer endlicher Automat (kurz ein DEA) ist ein Quintupel $M = (\Sigma, Z, \delta, z_0, F)$, wobei Σ ein Alphabet ist, Z eine endliche, nichtleere Menge von Zuständen mit $\Sigma \cap Z = \emptyset$, $\delta : Z \times \Sigma \rightarrow Z$ die Überführungsfunktion, $z_0 \in Z$ der Startzustand und $F \subseteq Z$ die Menge der Endzustände.

Die erweiterte Überführungsfunktion $\hat{\delta} : Z \times \Sigma^* \rightarrow Z$ von M ist induktiv definiert durch

$$\begin{aligned}\hat{\delta}(z, \varepsilon) &= z \quad \text{für jedes } z \in Z; \\ \hat{\delta}(z, ax) &= \hat{\delta}(\delta(z, a), x) \quad \text{für jedes } z \in Z, a \in \Sigma \text{ und } x \in \Sigma^*.\end{aligned}$$

Die von M akzeptierte Sprache ist definiert als $L(M) = \{x \in \Sigma^* \mid \hat{\delta}(z_0, x) \in F\}$.

Ein endlicher Automat kann durch einen Graphen repräsentiert werden, dessen Knoten die Zustände von M sind und dessen Kanten Zustandsübergänge gemäß der Überführungsfunktion δ von M darstellen. Ist $\delta(z, a) = z'$ für ein Symbol $a \in \Sigma$ und für zwei Zustände $z, z' \in Z$, so wird die gerichtete Kante von z nach z' mit a markiert. Der ausgezeichnete Startzustand z_0 wird durch einen auf ihn gerichteten Pfeil gekennzeichnet, und die Endzustände werden durch einen Doppelkreis markiert, siehe Abbildung 2.4.

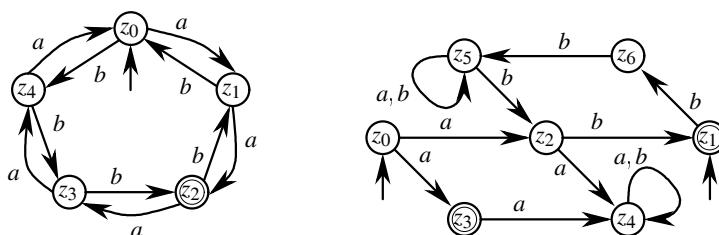


Abb. 2.4. Ein deterministischer und ein nichtdeterministischer endlicher Automat

Wie in Definition 2.11 bemerkt, können DEAs Sprachen erkennen. Für ein Eingabewort $x \in \Sigma^*$ führt der DEA M genau $|x|$ Schritte aus, nach welchen x entweder akzeptiert oder verworfen wird. Mit dem Startzustand z_0 beginnend, liest M dabei seine Eingabe x Symbol für Symbol, eines in jedem Schritt, und bewegt sich gemäß seiner Überführungsfunktion δ von Zustand zu Zustand: Ist M gerade im Zustand z und liest das Symbol $a \in \Sigma$ und ist $\delta(z, a) = z'$, so ändert sich der Zustand von

M zu z' . Die Berechnung hält, sobald das letzte Symbol von x gelesen wurde. Ist M dann in einem Endzustand, so wird x akzeptiert; andernfalls wird x verworfen. Die erweiterte Überführungsfunktion drückt den soeben beschriebenen Prozess formal aus: M akzeptiert x genau dann, wenn $\widehat{\delta}(z_0, x) \in F$.

Nun wird der Begriff des Nichtdeterminismus eingeführt, ein sehr mächtiges Konzept, das in vielen Gebieten der Informatik eine zentrale Rolle spielt, besonders auch in den späteren Kapiteln dieses Buches. Der links in Abbildung 2.4 bzw. Tabelle 2.5 dargestellte DEA ist deterministisch, weil jede Bewegung, die er machen kann, eindeutig durch seine Überführungsfunktion festgelegt ist. Die rechten Seiten in Abbildung 2.4 und in Tabelle 2.5 zeigen einen nichtdeterministischen endlichen Automaten (NEA), dessen Überführungsfunktion Paare (z, a) von Zuständen und Symbolen auf eine Teilmenge der Zustandsmenge abbildet. Daher erlaubt die Berechnung dieses Automaten nichtdeterministische Verzweigungen. Eine Eingabe wird von einem solchen nichtdeterministischen endlichen Automaten genau dann akzeptiert, wenn es wenigstens einen Berechnungspfad von einem der Startzustände zu einem Endzustand gibt. Die Frage, welche Sprachen der DEA und der NEA aus Abbildung 2.4 akzeptieren, wird in Übung 2.6 behandelt.

Tabelle 2.5. Überführungsfunktionen des DEA und des NEA aus Abbildung 2.4

δ	z_0	z_1	z_2	z_3	z_4
a	z_1	z_2	z_3	z_4	z_0
b	z_4	z_0	z_1	z_2	z_3

δ	z_0	z_1	z_2	z_3	z_4	z_5	z_6
a	$\{z_2, z_3\}$	\emptyset	$\{z_4\}$	$\{z_4\}$	$\{z_4\}$	$\{z_5\}$	\emptyset
b	\emptyset	$\{z_6\}$	$\{z_1\}$	\emptyset	$\{z_4\}$	$\{z_2, z_5\}$	$\{z_5\}$

Definition 2.12 (Nichtdeterministischer endlicher Automat). Ein nichtdeterministischer endlicher Automat (kurz ein NEA) ist ein Quintupel $M = (\Sigma, Z, \delta, S, F)$, wobei Σ ein Alphabet ist, Z eine endliche, nichtleere Menge von Zuständen mit $\Sigma \cap Z = \emptyset$, $\delta : Z \times \Sigma \rightarrow \mathcal{P}(Z)$ die Überführungsfunktion, $S \subseteq Z$ ist die Menge der Startzustände und $F \subseteq Z$ die Menge der Endzustände. Hier bezeichnet $\mathcal{P}(Z)$ die Potenzmenge von Z , d.h. die Menge aller Teilmengen von Z .

Die erweiterte Überführungsfunktion $\widehat{\delta} : \mathcal{P}(Z) \times \Sigma^* \rightarrow \mathcal{P}(Z)$ von M ist induktiv definiert durch

$$\begin{aligned}\widehat{\delta}(Z', \varepsilon) &= Z' \quad \text{für jede Teilmenge } Z' \subseteq Z; \\ \widehat{\delta}(Z', ax) &= \bigcup_{z \in Z'} \widehat{\delta}(\delta(z, a), x) \quad \text{für jedes } Z' \subseteq Z, a \in \Sigma \text{ und } x \in \Sigma^*.\end{aligned}$$

Die von M akzeptierte Sprache ist definiert als $L(M) = \{x \in \Sigma^* \mid \widehat{\delta}(S, x) \cap F \neq \emptyset\}$.

Eine Sprache kann genau dann von einem NEA akzeptiert werden, wenn sie durch eine reguläre Grammatik erzeugt wird, siehe Übung 2.7. Nach Definition ist jeder DEA ein spezieller NEA, dessen Überführungsfunktion δ jedes Paar (z, a) auf eine Menge abbildet, die genau ein Element enthält, und dessen Startzustandsmenge ebenfalls eine Einermenge ist. Umgekehrt gibt es für jeden NEA einen äquivalenten

DEA; der Beweis von Satz 2.13 wird in Übung 2.8 verlangt. Somit sind reguläre Grammatiken und deterministische und nichtdeterministische endliche Automaten paarweise äquivalente Begriffe; sie alle charakterisieren die Klasse der regulären Sprachen.

Satz 2.13 (Rabin und Scott). *Jede von einem NEA akzeptierbare Sprache kann auch von einem DEA erkannt werden.*

Korollar 2.14. $\mathcal{L}_3 = \text{REG} = \{L(M) \mid M \text{ ist ein DEA}\} = \{L(M) \mid M \text{ ist ein NEA}\}.$

Ähnliche Charakterisierungen durch geeignete Automatentypen sind auch für die anderen Klassen der Chomsky-Hierarchie bekannt. Beispielsweise kann die Klasse $\mathcal{L}_2 = \text{CF}$ der kontextfreien Sprachen durch (nichtdeterministische) Kellerautomaten, die Klasse $\mathcal{L}_1 = \text{CS}$ der kontextsensitiven Sprachen durch (nichtdeterministische) linear beschränkte Automaten und die Klasse \mathcal{L}_0 durch (deterministische oder nicht-deterministische) Turingmaschinen charakterisiert werden; für letztere siehe die Definitionen 2.15 und 2.16. Der Nachweis dieser Äquivalenzen soll hier jedoch nicht erbracht werden.

In der Komplexitätstheorie, die eines der zentralen Themen dieses Buches ist, beschäftigt man sich vor allem mit dem Beweis „unterer Schranken“ für die Berechnungskomplexität von Problemen. Die Schwierigkeit besteht darin, dass es dafür nicht genügt, die Laufzeit eines konkreten Algorithmus zu analysieren, der ein gegebenes Problem löst. Vielmehr muss man beweisen, dass überhaupt kein Algorithmus, der dieses Problem löst, eine bessere Laufzeit haben kann als die zu zeigende Schranke. Unter den Algorithmen, die man beim Beweis unterer Schranken betrachten muss, sind sogar solche, die noch gar nicht entworfen wurden. Daher muss man zunächst den Begriff des Algorithmus mathematisch streng formalisieren, denn andernfalls könnte man nicht über die Gesamtheit aller Algorithmen sprechen.

Seit den dreißiger Jahren des vorigen Jahrhunderts wurden verschiedene formale Algorithmenmodelle vorgeschlagen. Je zwei dieser Modelle stellten sich schließlich als äquivalent in dem Sinn heraus, dass das eine in das andere überführt werden kann und umgekehrt. In gewissem Sinn könnte man sagen, dass es sich bei einer solchen Transformation eines Modells in ein anderes um eine Art von Übersetzung zwischen zwei unterschiedlichen Programmiersprachen handelt. Die Tatsache, dass sich all diese Modelle als äquivalent erwiesen, liefert ein (natürlich lediglich informales) Argument für die Plausibilität der These von Church, welche sagt, dass ein jedes dieser Algorithmenmodelle genau den (natürlich etwas vagen, formal nicht beschreibbaren) Begriff des „intuitiv Berechenbaren“ erfasst.

Das hier adoptierte Algorithmenmodell ist die Turingmaschine, die 1936 von Alan Turing (1912 bis 1954) in seiner bahnbrechenden Arbeit eingeführt wurde [Tur36]. Die Turingmaschine ist ein ganz simples, abstraktes Computermodell. Es wird nun durch Angabe seiner Syntax und Semantik definiert. Wie bei endlichen Automaten kann man zwischen deterministischen und nichtdeterministischen Turingmaschinen unterscheiden. Aus Bequemlichkeit beginnen wir mit dem letzteren Modell. Deterministische Turingmaschinen ergeben sich dann unmittelbar als ein Spezialfall der nichtdeterministischen Turingmaschinen.

Zunächst werden einige technische Details und die Arbeitsweise von Turingmaschinen beschrieben. Eine Turingmaschine ist mit k Arbeitsbändern ausgestattet, die beidseitig unendlich und in Bandfelder unterteilt sind. Jedes Band hat einen Lese-Schreib-Kopf, der stets auf genau einem Feld dieses Bandes steht. Jedes Feld kann ein Symbol enthalten. Die Abwesenheit eines Symbols in einem Feld wird durch ein spezielles Symbol angezeigt, das Leerzeichen \square , das zwar zum Arbeitsalphabet der Maschine gehört, jedoch nicht zu ihrem Eingabealphabet. Die eigentliche Berechnung wird auf den Arbeitsbändern durchgeführt, und wenn sie beginnt, steht das Eingabewort auf dem Eingabeband, während alle anderen Bandfelder leer sind. Am Ende der Berechnung hält die Maschine mit der Ausgabe auf ihrem Ausgabeband. Man kann festlegen, dass vom Eingabeband nur gelesen und auf das Ausgabeband nur geschrieben werden darf, aber eine solche Festlegung ist nicht nötig und das Eingabe- und Ausgabeband können ebenso normale Lese-Schreib-Arbeitsbänder sein. Auch kann man per Konvention weitere Variationen der technischen Details und Einschränkungen festlegen. Zum Beispiel könnte man verlangen, dass die Bänder nur in einer Richtung unendlich sind (man spricht dann von einem einseitig unendlichen Band) oder dass sich die Bandköpfe nur in einer Richtung bewegen dürfen oder dass die Anzahl der Bänder beschränkt ist usw.

Ein Schritt in der Berechnung einer Turingmaschine besteht aus den folgenden Aktionen: Jeder Kopf liest das Symbol, das in dem aktuell besuchten Feld steht, und entweder lässt er dieses unverändert oder überschreibt es mit einem neuen Symbol, dann bewegt sich der Kopf ein Feld weiter nach links oder ein Feld weiter nach rechts, oder er bleibt auf dem aktuellen Feld stehen. Gleichzeitig kann die Maschine ihren aktuellen internen Zustand, der in ihrem inneren Gedächtnis („finite control“) gespeichert ist, entweder ändern oder beibehalten. Abbildung 2.5 zeigt eine zweibändige Turingmaschine.

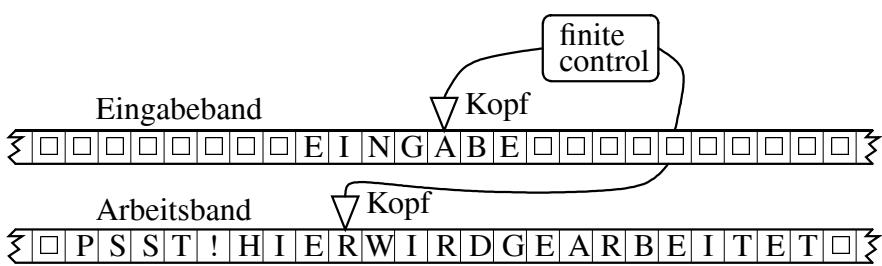


Abb. 2.5. Eine Turingmaschine

Definition 2.15 (Syntax von Turingmaschinen). Eine nichtdeterministische Turingmaschine mit k Bändern (kurz eine k -Band-NTM) ist ein Septupel

$$M = (\Sigma, \Gamma, Z, \delta, z_0, \square, F),$$

wobei Σ das Eingabealphabet ist, Γ das Arbeitsalphabet mit $\Sigma \subseteq \Gamma$, Z eine endliche, nichtleere Zustandsmenge mit $Z \cap \Gamma = \emptyset$, $\delta : Z \times \Gamma^k \rightarrow \mathfrak{P}(Z \times \Gamma^k \times \{L, R, N\}^k)$ die Überführungsfunktion, $z_0 \in Z$ der Startzustand, $\square \in \Gamma - \Sigma$ das Leerzeichen und F die Menge der Endzustände mit $F \subseteq Z$. Hier bezeichnet $\mathfrak{P}(Z)$ die Potenzmenge von Z , die Menge aller Teilmengen von Z .

Statt $(z', \mathbf{b}, \mathbf{x}) \in \delta(z, \mathbf{a})$ mit $z, z' \in Z$, $\mathbf{a}, \mathbf{b} \in \Gamma^k$ und $\mathbf{x} \in \{L, R, N\}^k$ schreiben wir auch $(z, \mathbf{a}) \mapsto (z', \mathbf{b}, \mathbf{x})$. Ein solcher Turingbefehl hat die folgende Bedeutung. Ist z der aktuelle Zustand von M und liest der i -te Kopf auf dem i -ten Band von M , $1 \leq i \leq k$, aktuell ein Feld, auf dem das Symbol a_i steht, wobei $\mathbf{a} = (a_1, a_2, \dots, a_k)$, dann werden die folgenden Aktionen ausgeführt:

- a_i wird durch b_i ersetzt, wobei $\mathbf{b} = (b_1, b_2, \dots, b_k)$,
- z' ist der neue Zustand von M , und
- M bewegt ihren i -ten Kopf gemäß $x_i \in \{L, R, N\}$, wobei $\mathbf{x} = (x_1, x_2, \dots, x_k)$, d.h., der i -te Kopf bewegt sich um ein Feld nach links, falls $x_i = L$, oder um ein Feld nach rechts, falls $x_i = R$, oder er bewegt sich überhaupt nicht, falls $x_i = N$.

Den Spezialfall einer deterministischen Turingmaschine mit k Bändern (kurz einer k -Band-DTM) erhält man, falls die Überführungsfunktion δ von $Z \times \Gamma^k$ in $Z \times \Gamma^k \times \{L, R, N\}^k$ abbildet.

Für $k = 1$ ergibt sich die einbändrige Turingmaschine, die wir einfach mit NTM oder DTM abkürzen. Jede k -Band-NTM und k -Band-DTM kann durch eine äquivalente einbändrige Turingmaschine simuliert werden, wobei sich die Laufzeit lediglich quadriert. Wenn Effizienz eine Rolle spielt, kann es jedoch vernünftig sein, mehr als ein Band zu verwenden.

Turingmaschinen können Sprachen erkennen bzw. akzeptieren, sie können aber auch Funktionen berechnen. Diese Unterscheidung ist bei der Definition der Semantik von Turingmaschinen zu berücksichtigen. Der Einfachheit halber beschränken wir uns auf den Fall der einbändrigen Turingmaschine; die Verallgemeinerung zur k -Band-Turingmaschine liegt auf der Hand.

Definition 2.16 (Semantik von Turingmaschinen). Sei $M = (\Sigma, \Gamma, Z, \delta, z_0, \square, F)$ eine (einbändrige) NTM. Eine Konfiguration von M ist ein Wort $k \in \Gamma^* Z \Gamma^*$. Dabei bedeutet $k = \alpha z \beta$, dass $\alpha \beta$ die aktuelle Bandinschrift ist (d.h., $\alpha \beta$ ist das Wort, dessen Buchstaben aktuell auf dem Band stehen), dass der Kopf gerade das erste Symbol von β liest, und dass z der aktuelle Zustand von M ist.

Auf der Menge $\mathfrak{K}_M = \Gamma^* Z \Gamma^*$ aller Konfigurationen von M definieren wir eine Binärrelation \vdash_M , die den Übergang von einer Konfiguration $k \in \mathfrak{K}_M$ in eine Konfiguration $k' \in \mathfrak{K}_M$ durch Anwendung der Überführungsfunktion δ beschreibt. Für alle Wörter $\alpha = a_1 a_2 \dots a_m$ und $\beta = b_1 b_2 \dots b_n$ in Γ^* , mit $m \geq 0$ und $n \geq 1$, und für alle Zustände $z \in Z$ definieren wir

$$\alpha z \beta \vdash_M \begin{cases} a_1 a_2 \dots a_m z' c b_2 \dots b_n & \text{falls } (z, b_1) \mapsto (z', c, N), m \geq 0, n \geq 1 \\ a_1 a_2 \dots a_m c z' b_2 \dots b_n & \text{falls } (z, b_1) \mapsto (z', c, R), m \geq 0, n \geq 2 \\ a_1 a_2 \dots a_{m-1} z' a_m c b_2 \dots b_n & \text{falls } (z, b_1) \mapsto (z', c, L), m \geq 1, n \geq 1. \end{cases}$$

Zwei Spezialfälle sind separat zu betrachten:

1. Ist $n = 1$ und $(z, b_1) \mapsto (z', c, R)$ (d.h., der Kopf von M bewegt sich nach rechts und trifft auf ein Leerzeichen, \square), so sei $a_1 a_2 \cdots a_m z b_1 \vdash_M a_1 a_2 \cdots a_m c z' \square$.
2. Ist $m = 0$ und $(z, b_1) \mapsto (z', c, L)$ (d.h., der Kopf von M bewegt sich nach links und trifft auf ein Leerzeichen, \square), so sei $z b_1 b_2 \cdots b_n \vdash_M z' \square c b_2 \cdots b_n$.

Die Startkonfiguration von M bei Eingabe x ist stets $z_0 x$. Die Endkonfigurationen von M bei Eingabe x haben die Form $\alpha z \beta$ mit $z \in F$ und $\alpha, \beta \in \Gamma^*$.

Sei \vdash_M^* die reflexive, transitive Hülle von \vdash_M . Das heißt, für je zwei Konfigurationen $k, k' \in \mathfrak{K}_M$ gilt $k \vdash_M^* k'$ genau dann, wenn es eine endliche Folge k_0, k_1, \dots, k_t von Konfigurationen in \mathfrak{K}_M gibt, so dass

$$k = k_0 \vdash_M k_1 \vdash_M \cdots \vdash_M k_t = k',$$

wobei $k = k_0 = k_t = k'$ möglich ist. Ist $k_0 = z_0 x$ die Startkonfiguration von M bei Eingabe x , so heißt diese Konfigurationsfolge eine endliche Berechnung von $M(x)$. Ist dabei außerdem k_t eine Endkonfiguration (d.h., der Zustand in k_t ist ein Endzustand), so sagen wir, M hält bei Eingabe x . Die von M akzeptierte Sprache ist definiert als

$$L(M) = \{x \in \Sigma^* \mid z_0 x \vdash_M^* \alpha z \beta \text{ mit } z \in F \text{ und } \alpha, \beta \in \Gamma^*\}.$$

Die Menge F der Endzustände von M kann auch in die Menge F_a der akzeptierenden Endzustände und die Menge F_r der ablehnenden Endzustände zerlegt werden, wobei $F = F_a \cup F_r$ und $F_a \cap F_r = \emptyset$. In diesem Fall ist die von M akzeptierte Sprache definiert durch

$$L(M) = \{x \in \Sigma^* \mid z_0 x \vdash_M^* \alpha z \beta \text{ mit } z \in F_a \text{ und } \alpha, \beta \in \Gamma^*\}.$$

Seien Σ und Δ Alphabete. Eine Turingmaschine M berechnet eine Wortfunktion $f : \Sigma^* \rightarrow \Delta^*$, falls für jedes $x \in \Sigma^*$ und für jedes $y \in \Delta^*$ gilt:

- $x \in D_f$ genau dann, wenn M bei Eingabe x nach endlich vielen Schritten hält, und
- für jedes $x \in D_f$ gilt $f(x) = y$ genau dann, wenn $z_0 x \vdash_M^* z y$ für ein $z \in F$ gilt,

wobei D_f den Definitionsbereich von f bezeichnet. Eine Wortfunktion heißt berechenbar, falls es eine sie berechnende Turingmaschine gibt.

Eine Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ heißt berechenbar, falls die durch

$$g(\text{bin}(x_1) \# \text{bin}(x_2) \# \cdots \# \text{bin}(x_k)) = \text{bin}(f(x_1, x_2, \dots, x_k))$$

definierte Wortfunktion $g : \{0, 1, \#\}^* \rightarrow \{0, 1\}^*$ berechenbar ist.

Im Falle von NTMs können Konfigurationen mehr als eine Folgekonfiguration haben. So erhalten wir einen *Berechnungsbaum*, dessen Wurzel die Startkonfiguration ist und dessen Blätter die Endkonfigurationen sind. Bäume sind spezielle Graphen (siehe Definition 2.49 und Problem 2.3) und bestehen wie diese aus Knoten und Kanten. Die Knoten des Berechnungsbaums von $M(x)$ sind die Konfigurationen von M bei Eingabe x . Für je zwei Konfigurationen k und k' aus \mathfrak{K}_M gibt es eine gerichtete Kante von k zu k' genau dann, wenn $k \vdash_M k'$. Ein *Berechnungspfad* im Baum von

$M(x)$ ist eine Folge von Konfigurationen $k_0 \vdash_M k_1 \vdash_M \dots \vdash_M k_t \vdash_M \dots$. Der Berechnungsbaum einer NTM kann unendliche Pfade haben. Im Falle einer DTM ist jede Konfiguration (außer natürlich der Startkonfiguration) eindeutig durch ihre unmittelbare Vorgängerkonfiguration bestimmt. Deshalb degeneriert der Berechnungsbaum einer DTM zu einer linearen Kette, die mit der Startkonfiguration beginnt und mit einer Endkonfiguration endet, falls die Maschine bei der betrachteten Eingabe hält; andernfalls geht die Berechnungskette ins Unendliche.

Definition 2.16 führte den Begriff der berechenbaren Funktion ein, der in der Berechenbarkeitstheorie ganz zentral ist. Die Ausdrücke „berechenbare Funktion“ und „rekursive Funktion“ werden synonym verwendet. Für Sprachen, die Probleme codieren, heißt der Begriff, der der Berechenbarkeit von Funktionen entspricht, „Entscheidbarkeit“.

Definition 2.17 (Berechenbarkeit und Entscheidbarkeit). \mathbb{P} sei die Klasse aller berechenbaren Funktionen (die auch als partiell-rekursive Funktionen bezeichnet werden), und $\mathbb{R} = \{f \mid f \in \mathbb{P} \text{ und } f \text{ ist total}\}$ sei die Klasse aller totalen (d.h. überall definierten) berechenbaren Funktionen (die auch als allgemein-rekursive Funktionen bezeichnet werden).

Die charakteristische Funktion einer Sprache L ist durch

$$c_L(x) = \begin{cases} 1 & \text{falls } x \in L \\ 0 & \text{falls } x \notin L \end{cases}$$

definiert. Eine Sprache L heißt entscheidbar, falls $c_L \in \mathbb{R}$.

REC bezeichne die Klasse aller entscheidbaren Sprachen.

Man kann die Inklusion $CS \subseteq REC$ zeigen, d.h., jede kontextsensitive Sprache ist entscheidbar. Diese Inklusion ist echt: $CS \neq REC$.

Beispiel 2.18. Betrachte die Sprache $L = \{a^n b^n c^n \mid n \geq 1\}$. Eine Turingmaschine, die L akzeptiert, ist durch

$$M = (\{a, b, c\}, \{a, b, c, \$, \square\}, \{z_0, z_1, \dots, z_6\}, \delta, z_0, \square, \{z_6\})$$

definiert, wobei die Liste der Turingbefehle von M gemäß der Überführungsfunktion δ in Tabelle 2.6 angegeben ist. Tabelle 2.7 gibt die Bedeutung der einzelnen Zustände von M sowie die mit jedem Zustand von M verbundene Absicht an. L ist entscheidbar, da c_L mittels einer Maschine berechenbar ist, die man durch eine geringfügige Modifikation von M erhält. Man beachte, dass M die Eigenschaft hat, dass sie niemals den Bandbereich verlässt, auf dem die Eingabe steht.³ Eine solche Turingmaschine heißt *linear beschränkter Automat* (kurz *LBA*). Da die Klasse CS durch linear beschränkte Automaten charakterisiert werden kann, ist L sogar kontextsensitiv. L ist jedoch nicht kontextfrei, was $CF \neq CS$ zeigt; siehe Problem 2.2.

³ Genauer gesagt kann M leicht so modifiziert werden, dass sie den rechten Rand der Eingabe erkennt und daher diesen Bandbereich nie verlässt.

Tabelle 2.6. Die Überführungsfunktion δ des LBA M für $L = \{a^n b^n c^n \mid n \geq 1\}$

$(z_0, a) \mapsto (z_1, \$, R)$	$(z_2, \$) \mapsto (z_2, \$, R)$	$(z_5, c) \mapsto (z_5, c, L)$
$(z_1, a) \mapsto (z_1, a, R)$	$(z_3, c) \mapsto (z_3, c, R)$	$(z_5, \$) \mapsto (z_5, \$, L)$
$(z_1, b) \mapsto (z_2, \$, R)$	$(z_3, \square) \mapsto (z_4, \square, L)$	$(z_5, b) \mapsto (z_5, b, L)$
$(z_1, \$) \mapsto (z_1, \$, R)$	$(z_4, \$) \mapsto (z_4, \$, L)$	$(z_5, a) \mapsto (z_5, a, L)$
$(z_2, b) \mapsto (z_2, b, R)$	$(z_4, \square) \mapsto (z_6, \square, R)$	$(z_5, \square) \mapsto (z_0, \square, R)$
$(z_2, c) \mapsto (z_3, \$, R)$	$(z_4, c) \mapsto (z_5, c, L)$	$(z_0, \$) \mapsto (z_0, \$, R)$

Tabelle 2.7. Interpretation der Zustände von M

Zustand	Bedeutung	Absicht
z_0	Startzustand	beginne neuen Zyklus
z_1	ein a gemerkt	suche nächstes b
z_2	ein a und ein b gemerkt	suche nächstes c
z_3	ein a , ein b und ein c gelöscht	suche rechten Rand
z_4	rechter Rand erreicht	gehe zurück und teste, ob alle a, b, c gelöscht
z_5	Test nicht erfolgreich	gehe zurück und beginne neuen Zyklus
z_6	Test erfolgreich	akzeptiere

Ein weiterer wichtiger Begriff in der Berechenbarkeitstheorie ist der rekursiven Aufzählbarkeit. Synonym kann man den Ausdruck „Semi-Entscheidbarkeit“ verwenden. Eine Sprache L ist *semi-entscheidbar*, falls $\hat{c}_L \in \mathbb{P}$, wobei die partielle charakteristische Funktion \hat{c}_L von L so definiert ist: $\hat{c}_L(x) = 1$, falls $x \in L$, und andernfalls ist $\hat{c}_L(x)$ nicht definiert. Anders gesagt bedeutet eine Semi-Entscheidung für L , dass ein Algorithmus (z.B. als Turingmaschine implementiert) die Antwort „ja“ für jede Eingabe x in L gibt, aber bei Eingaben, die nicht zu L gehören, niemals hält. Nun wird der hierzu äquivalente Begriff der rekursiv aufzählbaren Mengen definiert.

Definition 2.19 (Rekursive Aufzählbarkeit). Eine Sprache A heißt rekursiv aufzählbar, falls A entweder die leere Menge oder das Bild einer totalen berechenbaren Funktion f ist. Diese Funktion $f \in \mathbb{R}$ zählt A rekursiv auf in dem Sinn, dass $A = \{y \mid f(x) = y \text{ für ein } x\}$ gilt.

RE bezeichne die Klasse aller rekursiv aufzählbaren Sprachen.

Viel ist über die rekursiv aufzählbaren Sprachen bekannt. Hier beschränken wir uns auf ein paar wenige Resultate ohne Beweis. Viele solcher Beweise wenden die Methode der *Diagonalisierung* an. Eine wichtige Voraussetzung dieser Methode ist die *Gödelisierung von Turingmaschinen*, die hier nicht formal eingeführt, sondern lediglich grob skizziert werden soll, wobei die etwas mühselige Aufgabe, die formalen Details auszuführen, als Übung 2.9 gestellt wird.

Das Ziel ist dabei, alle Algorithmen einer gegebenen Klasse von Algorithmen systematisch durchzunummerieren; beispielsweise will man vielleicht alle Turingmaschinen nummerieren. Der Witz besteht darin, dass alle (syntaktisch korrekten) Turingmaschinen als Wörter über einem geeigneten Alphabet codiert werden. Solche Codewörter können lexikographisch geordnet werden, setzt man eine Ordnung

des zugrunde liegenden Alphabets voraus. Wörter, die keine syntaktisch korrekte Turingmaschine codieren, fallen heraus. In der verbleibenden geordneten Folge taucht jede vorstellbare Turingmaschine auf. Hat das zugrunde liegende Alphabet k Symbole, so können die Codewörter für Turingmaschinen mit Zahlen in k -adischer Darstellung identifiziert werden. Man erhält in dieser Weise eine Nummerierung von Turingmaschinen; die einer Maschine zugeordnete Nummer heißt ihre *Gödelnummer*, benannt nach dem berühmten Mathematiker Kurt Gödel (1906 bis 1978). Der entscheidende Punkt ist die Beobachtung, dass man die Gödelnummer einer gegebenen Turingmaschine *effektiv* (d.h. mittels eines Algorithmus) bestimmen kann und umgekehrt.

Einer gegebenen Gödelisierung M_0, M_1, M_2, \dots von Turingmaschinen, die Funktionen berechnen, entspricht eine Nummerierung $\varphi_0, \varphi_1, \varphi_2, \dots$ der Funktionen in \mathbb{P} , wobei φ_i für jedes i durch M_i berechnet wird. Ähnlich entspricht einer gegebenen Gödelisierung N_0, N_1, N_2, \dots von Turingmaschinen, die Sprachen akzeptieren, eine Nummerierung L_0, L_1, L_2, \dots von Sprachen derart, dass N_i für jedes i die Sprache L_i akzeptiert, d.h., $L(N_i) = L_i$. Natürlich können verschiedene Turingmaschinen dieselbe Funktion berechnen bzw. dieselbe Sprache akzeptieren. Somit berechnen unendlich viele Maschinen (denen unendlich viele verschiedene Gödelnummern zugeordnet sind) ein und dieselbe Funktion, bzw. sie akzeptieren ein und dieselbe Sprache.

Die Gesamtheit aller Sprachen, die so einer fixierten Gödelisierung von Turingmaschinen entsprechen, ist RE, die Klasse der rekursiv aufzählbaren Mengen. Satz 2.20 listet ohne Beweis eine Reihe von Bedingungen auf, die alle äquivalent zur rekursiven Aufzählbarkeit sind, sowie weitere grundlegende Eigenschaften der Klassen RE und REC.

Satz 2.20 (Charakterisierungen und Eigenschaften von REC und RE).

1. $A \in \text{REC}$ genau dann, wenn $A \in \text{RE}$ und $\overline{A} \in \text{RE}$. Somit ist $\text{REC} \subseteq \text{RE}$.
2. $A \in \text{RE}$ genau dann, wenn A die Projektion einer entscheidbaren Menge B ist, d.h., $A \in \text{RE}$ genau dann, wenn es eine Menge $B \in \text{REC}$ gibt, so dass gilt:

$$(\forall x)[x \in A \iff (\exists y)[(x, y) \in B]].$$

3. Sei $\varphi_0, \varphi_1, \dots$ eine fixierte Gödelisierung von \mathbb{P} . $D_i = D_{\varphi_i}$ bzw. $R_i = R_{\varphi_i}$ bezeichnen den Definitionsbereich bzw. Wertebereich der i -ten Funktion φ_i in \mathbb{P} . Dann gilt:

$$\text{RE} = \{D_i \mid i \in \mathbb{N}\} = \{R_i \mid i \in \mathbb{N}\}.$$

4. Das (spezielle) Halteproblem, welches definiert ist als $H = \{i \in \mathbb{N} \mid i \in D_i\}$, ist rekursiv aufzählbar, aber nicht entscheidbar. Folglich ist $\text{REC} \neq \text{RE}$.
5. $\text{RE} = \mathcal{L}_0$.
6. $A \in \text{RE}$ genau dann, wenn $\hat{c}_A \in \mathbb{P}$.

Zusammengefasst kann Fakt 2.10 zur folgenden Aussage erweitert und verschärft werden, wobei für Mengenklassen \mathcal{C} und \mathcal{D} die *echte Inklusion* durch $\mathcal{C} \subset \mathcal{D}$ bezeichnet wird, d.h., $\mathcal{C} \subset \mathcal{D} \iff (\mathcal{C} \subseteq \mathcal{D} \text{ und } \mathcal{C} \neq \mathcal{D})$.

Satz 2.21. $\mathcal{L}_3 = \text{REG} \subset \mathcal{L}_2 = \text{CF} \subset \mathcal{L}_1 = \text{CS} \subset \text{REC} \subset \mathcal{L}_0 = \text{RE}$.

Schließlich definieren wir noch den Begriff der Orakel-Turingmaschine.

Definition 2.22 (Orakel-Turingmaschine).

Eine Orakelmenge (oder kurz ein Orakel) ist eine Menge von Wörtern. Eine Orakel-Turingmaschine M , etwa mit Orakel B , ist eine Turingmaschine, die mit einem besonderen Arbeitsband ausgestattet ist, dem so genannten Orakelband oder Frageband, und deren Zustandsmenge einen besonderen Fragezustand, $z_?$, und die zwei besonderen Antwortzustände z_{ja} und z_{nein} enthält. Solange M nicht im Fragezustand $z_?$ ist, arbeitet sie genau wie eine normale Turingmaschine. Erreicht M im Laufe ihrer Berechnung jedoch den Fragezustand $z_?$, so unterbricht sie ihre Berechnung und befragt ihr Orakel über das Wort q , das aktuell auf ihrem Orakelband steht. Das Orakel B kann man sich als eine Art „black box“ vorstellen: B beantwortet die Frage, ob es q enthält oder nicht, in einem Schritt der Berechnung von M , egal, wie schwer es ist (oder ob es überhaupt möglich ist), die Menge B zu entscheiden. Ist $q \in B$, so geht M in den Antwortzustand z_{ja} über und setzt die Berechnung fort. Andernfalls (wenn $q \notin B$) setzt M die Berechnung im Antwortzustand z_{nein} fort. Wir sagen dann, die Berechnung von M bei Eingabe x erfolgt relativ zum Orakel B , und wir schreiben $M^B(x)$.

Sei $L(M^B)$ die von M^B akzeptierte Sprache. Eine Mengenklasse \mathcal{C} heißt relativierbar, falls sie in dieser Weise durch Orakel-Turingmaschinen relativ zum leeren Orakel repräsentiert werden kann. Eine Sprache $L \in \mathcal{C}$ wird durch eine Orakel-Turingmaschine M repräsentiert, falls $L = L(M^\emptyset)$. Für eine relativierbare Klasse \mathcal{C} und für ein Orakel B definieren wir die Klasse \mathcal{C} relativ zu B durch

$$\mathcal{C}^B = \{L(M^B) \mid \text{die Orakel-Turingmaschine } M \text{ repräsentiert eine Menge in } \mathcal{C}\}.$$

Für eine relativierbare Klasse \mathcal{C} und eine Klasse \mathcal{B} von Orakelmengen definieren wir $\mathcal{C}^\mathcal{B} = \bigcup_{B \in \mathcal{B}} \mathcal{C}^B$.

Natürlich gilt $\mathcal{C}^\emptyset = \mathcal{C}$, und wir lassen daher das leere Orakel weg. Weiter verwenden wir die Abkürzungen DOTM bzw. NOTM für *deterministische* bzw. für *nichtdeterministische Orakel-Turingmaschine*. Ist die Laufzeit einer DOTM bzw. einer NOTM durch ein Polynom (in der Eingabegröße) beschränkt, so schreiben wir DPOTM bzw. NPOTM; siehe auch Abschnitt 3.2.

2.3 Logik

2.3.1 Aussagenlogik

„I did not have sex with that woman“ („Ich hatte keinen Sex mit jener Frau“), wird ein früherer US-Präsident gern zitiert. Hat er eine wahre Aussage gemacht? Mit Logik ist diese Frage nicht zu beantworten. Was Logik jedoch kann, ist, den Wahrheitsgehalt von komplizierten Aussagen zu bewerten, die sich aus mehreren einfachen („atomaren“) Aussagen zusammensetzen, welche durch logische Operatoren wie *und*, *oder* und *nicht* verknüpft sind. Der Wahrheitswert einer solchen zusammengesetzten Aussage, die auch als eine (boolesche) Formel bezeichnet wird, ergibt

sich nämlich aus den gegebenen Wahrheitswerten der atomaren Aussagen in der Formel nach den Gesetzen der Logik. Betrachte zum Beispiel die folgende Aussage:

„Ich hatte keinen Sex mit jener Frau, oder wenn ich jemals doch Sex mit jener Frau hatte und kein Lügner bin, dann habe ich gerade nichts als die Wahrheit gesagt.“

Man kann diese Aussage gefahrlos machen, selbst unter Eid, einfach weil sie immer wahr ist, vorausgesetzt, wir würden ihren Inhalt „textlich“ interpretieren. Um den Wahrheitswert der Formel S , die den obigen Satz darstellt, zu bestimmen, zerlegt man sie in ihre atomaren Bestandteile und untersucht ihre logische Struktur.

Was sind die atomaren Bestandteile von S ? Sei A die Aussage: „Ich hatte keinen Sex mit jener Frau.“ Die Aussage „Ich hatte (jemals) doch Sex mit jener Frau“ ist die logische Negation von A und wird mit $\neg A$ bezeichnet. Weiter sei B die Aussage „Ich bin ein Lügner“, deren Negation $\neg B$ ist: „Ich bin kein Lügner.“ Schließlich sei C die Aussage: „Ich habe (gerade) nichts als die Wahrheit gesagt.“

Wir verwenden die Symbole \neg , \vee , \wedge , \Rightarrow und \Leftrightarrow für die logischen Operationen *Negation* („nicht“), *Disjunktion* („oder“), *Konjunktion* („und“), *Implikation* („wenn..., dann...“) und *Äquivalenz* („genau dann, wenn“). In dieser Notation hat S die logische Struktur

$$S = A \vee ((\neg A \wedge \neg B) \Rightarrow C). \quad (2.5)$$

Boolesche Operationen werden durch ihre Wahrheitstafeln definiert. Zum Beispiel „kippt“ die Negation \neg den Wahrheitswert ihres Arguments: Ist A wahr, so ist $\neg A$ falsch, und ist A falsch, so ist $\neg A$ wahr. Der Kürze wegen stellen wir die konstanten Wahrheitswerte *wahr* und *falsch* durch 1 bzw. 0 dar. Tabelle 2.8 gibt die Wahrheitstafeln für die übrigen der oben erwähnten booleschen Operationen an, für jede mögliche Belegung ihrer beiden Argumente mit Wahrheitswerten. Insgesamt gibt es genau $2^4 = 16$ verschiedene binäre boolesche Operationen.

Tabelle 2.8. Wahrheitstafeln für einige boolesche Operationen

x	y	$x \vee y$	$x \wedge y$	$x \Rightarrow y$	$x \Leftrightarrow y$
0	0	0	0	1	1
0	1	1	0	1	0
1	0	1	0	0	0
1	1	1	1	1	1

Angenommen, die Person, die die Aussage S macht, sagt die Wahrheit. In diesem Fall ist S natürlich wahr. Nun nimm an, die Person, die die Aussage S macht, lügt, d.h., S ist falsch. Nach Definition von „ \vee “ in Tabelle 2.8 müssen dann sowohl die Aussage A („Ich hatte keinen Sex mit jener Frau“) und die Aussage

$$(\neg A \wedge \neg B) \Rightarrow C$$

falsch sein. Da A falsch ist, wissen wir, dass $\neg A$ wahr ist. Doch da die Person lügt, ist auch B („Ich bin ein Lügner“) in diesem Fall wahr, woraus folgt, dass $\neg B$ falsch ist. Also ist die Voraussetzung $\neg A \wedge \neg B$ der Implikation in (2.5) falsch. Nach Definition von „ \Rightarrow “ in Tabelle 2.8 ist die Implikation $(\neg A \wedge \neg B) \Rightarrow C$ wahr, egal, ob ihre Konklusion C wahr ist oder nicht. Widerspruch. Es folgt, dass der gerade betrachtete Fall nicht eintreten kann: S ist wahr, unabhängig davon, ob die Person, die S ausspricht, lügt oder nicht.

Das obige Argument zeigt, dass der Begriff „Wahrheit“ und die Definition eines „Lügners“ im wirklichen Leben etwas vage und dehnbar sind, besonders in der Politik. Im strengen und abstrakten Gebiet der Logik interpretieren wir die Inhalte von Aussagen jedoch nicht „textlich“. In der Logik ist S keine „Tautologie“, also keine boolesche Formel, die unter jeder möglichen Wahrheitswertbelegung ihrer Variablen wahr ist. Abstrahieren wir vom „textlichen Inhalt“ der Formel S und betrachten ihre atomaren Teilformeln A , B und C lediglich als boolesche Variablen, die entweder wahr oder falsch sein können, dann ist S nicht immer wahr, denn setzt man die Variablen A , B und C zum Beispiel alle falsch, so erhält man eine Wahrheitswertbelegung, die S falsch macht.

Definition 2.23 (Syntax von Booleschen Formeln).

- Die booleschen Konstanten wahr und falsch werden durch 1 bzw. 0 repräsentiert. Seien x_1, x_2, \dots boolesche Variablen, d.h., $x_i \in \{0, 1\}$ für jedes i . Boolesche Variablen und Konstanten heißen auch atomare Formeln. Variablen und ihre Negationen heißen Literale. Per Konvention legen wir fest, dass die Indizes von Variablen weggelassen werden können, d.h., wir schreiben manchmal z.B. x, y, z, \dots statt x_1, x_2, x_3, \dots
- Boolesche Formeln (oder kurz Formeln) sind induktiv wie folgt definiert:
 1. Jede atomare Formel ist eine Formel.
 2. Ist φ eine Formel, so ist $\neg\varphi$ eine Formel.
 3. Sind φ und ψ Formeln, so sind $\varphi \vee \psi$ und $\varphi \wedge \psi$ Formeln.
- Zur Abkürzung schreiben wir
 - $\varphi \Rightarrow \psi$ für $\neg\varphi \vee \psi$,
 - $\varphi \iff \psi$ für $(\varphi \wedge \psi) \vee (\neg\varphi \wedge \neg\psi)$,
 - $\bigvee_{i=1}^n \varphi_i$ für $\varphi_1 \vee \varphi_2 \vee \dots \vee \varphi_n$ und
 - $\bigwedge_{i=1}^n \varphi_i$ für $\varphi_1 \wedge \varphi_2 \wedge \dots \wedge \varphi_n$.
- Eine boolesche Formel φ ist in konjunktiver Normalform (kurz KNF), falls φ die Form

$$\begin{aligned}\varphi(x_1, x_2, \dots, x_n) &= \bigwedge_{i=1}^m \left(\bigvee_{j=1}^{k_i} \ell_{i,j} \right) \\ &= (\ell_{1,1} \vee \dots \vee \ell_{1,k_1}) \wedge \dots \wedge (\ell_{m,1} \vee \dots \vee \ell_{m,k_m})\end{aligned}$$

hat, wobei die $\ell_{i,j}$ Literale über $\{x_1, x_2, \dots, x_n\}$ sind. Als die Klauseln von φ werden die Disjunktionen $\left(\bigvee_{j=1}^{k_i} \ell_{i,j}\right)$ von Literalen bezeichnet.

- Eine boolesche Formel φ ist in k -KNF, falls φ in KNF ist und jede Klausel von φ höchstens k Literale hat.

Analog zum oben definierten Begriff der KNF kann man die *disjunktive Normalform* (kurz DNF) von booleschen Formeln definieren, siehe Übung 2.10.

Definition 2.24 (Semantik von Booleschen Formeln).

- Sei $\varphi(x_1, x_2, \dots, x_n)$ eine gegebene boolesche Formel. Eine Wahrheitswertbelegung von φ ist eine Abbildung $\alpha : \{x_1, x_2, \dots, x_n\} \rightarrow \{0, 1\}$, die jeder Variablen von φ einen Wahrheitswert zuweist. Wertet man φ gemäß α aus, ergibt sich ein Wahrheitswert $\varphi(\alpha) \in \{0, 1\}$ für φ .
- Eine Wahrheitswertbelegung α erfüllt eine boolesche Formel φ , falls φ unter α wahr ist, d.h., $\varphi(\alpha) = 1$. Eine erfüllende Wahrheitswertbelegung α einer Formel φ heißt auch ein Modell von φ . Eine boolesche Formel ist erfüllbar, falls es eine erfüllende Belegung für sie gibt. Eine boolesche Formel ist gültig (oder eine Tautologie), falls sie unter jeder Wahrheitswertbelegung wahr ist.
- Je zwei Formeln φ und ψ (mit denselben Variablen) heißen (semantisch) äquivalent (bezeichnet mit $\varphi \equiv \psi$), falls für jede Belegung α gilt: $\varphi(\alpha) = \psi(\alpha)$.

Formeln mit verschiedenen Variablen können ebenfalls semantisch äquivalent sein. Beispielsweise ist jede Tautologie, egal von welchen Variablen sie abhängt, äquivalent zur Konstanten *wahr*. Tabelle 2.9 listet einige Äquivalenzen auf, die zur Vereinfachung von booleschen Formeln verwendet werden können. Der Beweis der Äquivalenzen aus Tabelle 2.9 wird in die Übung 2.12 verschoben.

Tabelle 2.9. Äquivalenzen zwischen booleschen Formeln

Name	Regel	Name	Regel
Idempotenz	$\varphi \vee \varphi \equiv \varphi$ $\varphi \wedge \varphi \equiv \varphi$	Assoziativität	$(\varphi \vee \psi) \vee \rho \equiv \varphi \vee (\psi \vee \rho)$ $(\varphi \wedge \psi) \wedge \rho \equiv \varphi \wedge (\psi \wedge \rho)$
Kommutativität	$\varphi \vee \psi \equiv \psi \vee \varphi$ $\varphi \wedge \psi \equiv \psi \wedge \varphi$	Distributivität	$\varphi \vee (\psi \wedge \rho) \equiv (\varphi \vee \psi) \wedge (\varphi \vee \rho)$ $\varphi \wedge (\psi \vee \rho) \equiv (\varphi \wedge \psi) \vee (\varphi \wedge \rho)$
Tautologieregeln	$1 \vee \varphi \equiv 1$ $1 \wedge \varphi \equiv \varphi$	Absorption	$\varphi \vee (\varphi \wedge \psi) \equiv \varphi$ $\varphi \wedge (\varphi \vee \psi) \equiv \varphi$
Unerfüllbarkeitsregeln	$0 \vee \varphi \equiv \varphi$ $0 \wedge \varphi \equiv 0$	deMorgansche Regeln	$\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi$ $\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi$
Doppelnegation	$\neg\neg\varphi \equiv \varphi$		

Beispiel 2.25 (Boolesche Formel).

1. Die durch

$$\begin{aligned}\varphi(w, x, y, z) &= (x \vee y \vee \neg z) \wedge (x \vee \neg y \vee \neg z) \wedge (w \vee \neg y \vee z) \wedge (\neg w \vee z); \\ \psi(w, x, y, z) &= (\neg w \vee x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z) \wedge (\neg w \vee y \vee z)\end{aligned}$$

definierten booleschen Formeln φ und ψ sind erfüllbar, siehe Übung 2.11(a). Hier ist φ eine Formel in 3-KNF. Im Gegensatz dazu ist die Formel ψ nicht in 3-KNF, da ihre erste Klausel vier Literale enthält.

2. Übung 2.11(c) fragt nach einem Beweis dafür, dass die Formel

$$\varphi = ((\neg x \wedge \neg y) \Rightarrow \neg y)$$

eine Tautologie ist. Es sei dazu angemerkt, dass jede Formel φ genau dann gültig ist, wenn ihre Negation $\neg\varphi$ nicht erfüllbar ist.

In Kapitel 3 werden Varianten des Erfüllbarkeitsproblems für boolesche Formeln untersucht, siehe Definition 3.44 in Abschnitt 3.5. Kapitel 5 studiert die Komplexität bestimmter Probleme für *quantifizierte boolesche Formeln*, siehe insbesondere Definition 5.34 in Abschnitt 5.2 und auch Abschnitt 5.6. Die Menge der quantifizierten booleschen Formeln erhält man durch Hinzufügen von *existenziellen* und *universellen Quantoren* zu den in Definition 2.23 eingeführten booleschen Formeln. Diese Menge stellt also eine Erweiterung der Menge der booleschen Formeln dar.

Definition 2.26 (Quantifizierte Boolesche Formel).

Die Menge der quantifizierten booleschen Formeln (kurz QBFs) erweitert die Menge der in den Definitionen 2.23 und 2.24 definierten booleschen Formeln durch Hinzunahme des Existenzquantors \exists und des Allquantors \forall . Das heißt, jede boolesche Formel ist eine QBF, und ist $F(x)$ eine boolesche Formel (mit oder ohne Quantoren), so sind auch $(\exists x)[F(x)]$ und $(\forall x)[F(x)]$ QBFs.

Die eckigen Klammern geben dabei den Wirkungsbereich eines Quantors an; wenn dieser aus dem Zusammenhang klar ist, darf man auf diese Klammern auch verzichten. Auch schreiben wir gelegentlich \vee statt \exists und \wedge statt \forall .

Jedes Vorkommen einer Variablen ist entweder durch einen Quantor gebunden oder frei. Ein Vorkommen einer Variablen x in einer QBF F heißt gebunden (oder quantifiziert), falls x in einer Teilformel der Form $(\exists x)G$ oder $(\forall x)G$ von F vorkommt; andernfalls ist dieses Vorkommen von x frei. Eine QBF F heißt geschlossen, falls alle Variablen in F quantifiziert sind. Andernfalls (d.h., falls freie Variablen in F vorkommen) heißt F offen.

Die Semantik von QBFs wird in offensichtlicher Weise definiert, siehe die Definitionen 2.24 und 2.32. Die Details seien dem Leser als Übung 2.12(b) überlassen. Die Begriffe der Erfüllbarkeit, Gültigkeit und semantischen Äquivalenz aus Definition 2.24 lassen sich unmittelbar auf quantifizierte boolesche Formeln übertragen.

Jede geschlossene QBF F lässt sich zu entweder wahr oder falsch auswerten. Im Gegensatz dazu ist eine offene Formel F eine boolesche Funktion ihrer $k \geq 1$ freien (d.h. nicht quantifizierten) Variablen, welche von $\{0, 1\}^k$ in $\{0, 1\}$ abbildet. In Kapitel 5 werden wir uns auf geschlossene QBFs beschränken.

Die in Tabelle 2.9 dargestellten Äquivalenzen können ebenso auch für quantifizierte boolesche Formeln bewiesen werden, siehe Übung 2.12. Wegen der Hinzunahme von Quantoren können sogar zusätzliche Äquivalenzen gezeigt werden. Insbesondere können die deMorganschen Regeln verallgemeinert werden zu:

$$\neg(\exists x)[F(x)] \equiv (\forall x)[\neg F(x)] \quad \text{und} \quad \neg(\forall x)[F(x)] \equiv (\exists x)[\neg F(x)]. \quad (2.6)$$

Beispiel 2.27 (Quantifizierte Boolesche Formel). Betrachte die offene QBF

$$F = (\forall x) (\exists y) [(x \wedge y) \vee \neg z] \vee \neg (\forall x) [x \vee \neg y]. \quad (2.7)$$

Die freien Variablen vorkommen von F sind z und das Vorkommen von y ganz rechts; alle anderen vorkommenden Variablen sind quantifiziert. Natürlich kann ein und dieselbe Variable sowohl frei als auch quantifiziert in einer Formel vorkommen. Beispiel 2.29 unten zeigt, dass F erfüllbar ist.

Um die geschlossene QBF

$$G = (\forall x) [x \wedge (\exists y) [(x \wedge y) \implies \neg x]]$$

auszuwerten, betrachte zunächst die Teilformel

$$H(x) = (\exists y) [(x \wedge y) \implies \neg x]$$

von G . Die Variable y ist existenziell quantifiziert; belegt man y mit dem Wahrheitswert 0, so vereinfacht man $H(x)$ zu

$$H(x) \equiv ((x \wedge 0) \implies \neg x) \equiv (0 \implies \neg x) \equiv 1.$$

Folglich gilt

$$G \equiv (\forall x) [x \wedge H(x)] \equiv (\forall x) [x \wedge 1] \equiv (\forall x) [x] \equiv 0 \wedge 1 \equiv 0.$$

Somit ist G falsch.

Jede QBF kann in eine äquivalente QBF in Pränexform transformiert werden, d.h., alle Quantoren erscheinen dann als ein Linksspräfix der Formel. Außerdem können aufeinander folgende Quantoren derselben Art zu einem einzigen Quantor dieser Art zusammengefasst werden, welcher dann mehrere Variablen quantifiziert; wir schreiben sie hier als Mengen, aber verwenden eine Codierung durch eine Standard-Paarungsfunktion, wenn wir Berechnungsprobleme für QBFs in Kapitel 5 behandeln. Durch Umbenennen der quantifizierten Variablen können die Variablenmengen nach jedem Quantor paarweise disjunkt gemacht werden.

Definition 2.28 (Pränexform einer QBF). Eine QBF F ist in Pränexform, falls F die folgende Form hat:

$$F(x_1, \dots, x_k) = (\mathfrak{Q}_1 y_1) \cdots (\mathfrak{Q}_n y_n) \varphi(x_1, \dots, x_k, y_1, \dots, y_n),$$

wobei $\mathfrak{Q}_i \in \{\exists, \forall\}$ für jedes i mit $1 \leq i \leq n$ und φ eine boolesche Formel ohne Quantoren ist und x_1, \dots, x_k die in F vorkommenden freien Variablen sind.

Beispiel 2.29 (Pränexform einer QBF). Betrachte erneut die offene QBF F aus (2.7):

$$F(y, z) = (\forall x) (\exists y) [(x \wedge y) \vee \neg z] \vee \neg (\forall x) [x \vee \neg y].$$

F wird folgendermaßen in eine äquivalente QBF in Pränexform transformiert:

Schritt 1: Benenne die quantifizierten Variablen so um, dass F in eine äquivalente Formel F_1 umgewandelt wird, in der keine Variable sowohl frei als auch gebunden auftritt und in der alle quantifizierten Variablen disjunkt sind:

$$F_1(y, z) = (\forall x) (\exists u) [(x \wedge u) \vee \neg z] \vee \neg (\forall v) [v \vee \neg y].$$

Schritt 2: Transformiere F_1 durch Anwendung der deMorganschen Regeln (siehe (2.6)) in eine äquivalente Formel F_2 in Pränexform:

$$F_2(y, z) = (\forall x) (\exists u) (\exists v) [(x \wedge u) \vee \neg z \vee (\neg v \wedge y)].$$

Schritt 3: Fasse aufeinander folgende Quantoren derselben Art in F_2 zu einem Quantor dieser Art zusammen, welcher nun über eine Variablenmenge quantifiziert:

$$F_3(y, z) = (\forall x) (\exists \{u, v\}) [(x \wedge u) \vee \neg z \vee (\neg v \wedge y)].$$

F_3 und F sind äquivalente QBFs. Um zu sehen, dass F_3 (und somit F) erfüllbar ist, wähle eine Belegung, die die freien Variablen y und z wahr macht. Wertet man F_3 unter dieser Belegung aus, so erhält man eine geschlossene QBF, die durch Anwenden der Tautologie- und Unerfüllbarkeitsregeln aus Tabelle 2.9 zu

$$(\forall x) (\exists \{u, v\}) [(x \wedge u) \vee \neg v]$$

vereinfacht werden kann. Da es für jede Belegung von x mit Wahrheitswerten Belegungen von u und v gibt, so dass die Teilformel $(x \wedge u) \vee \neg v$ wahr wird, ist F_3 (und daher F) erfüllbar.

2.3.2 Prädikatenlogik

Nicht alles, was man vielleicht gern ausdrücken möchte, kann mittels der Aussagenlogik formuliert werden. Betrachte beispielsweise die Aussage: „Es gibt einen Apfelbaum, dessen sämtliche Äpfel rot sind.“ Um diesen Satz mit den Mitteln der Logik zu formalisieren, müssen wir Eigenschaften von Objekten, wie die Farbe von Äpfeln, formal ausdrücken können. Solche Eigenschaften nennt man *Prädikate*. Des Weiteren möchte man oft bestimmte funktionale Zusammenhänge zwischen Objekten ausdrücken. Man erinnere sich zum Beispiel an die Definition der \mathcal{O} -Notation:

$$\mathcal{O}(g) = \{f : \mathbb{N} \rightarrow \mathbb{N} \mid (\exists c > 0) (\exists n_0 \in \mathbb{N}) (\forall n \geq n_0) [f(n) \leq c \cdot g(n)]\}.$$

Ausgeschrieben liest sich diese Definition so: „ $\mathcal{O}(g)$ besteht aus all den Funktionen f von \mathbb{N} in \mathbb{N} , für die Konstanten $c > 0$ und $n_0 \in \mathbb{N}$ existieren, so dass für alle $n \geq n_0$, $f(n) \leq c \cdot g(n)$ gilt.“ Diese Aussage beinhaltet existentielle und universale Quantoren, die Funktionen f , g und die Multiplikation sowie die Relation \leq . (Eine Binärrelation wie \leq ist eine Menge von Paaren von Elementen aus einem Universum und somit ein zweistelliges Prädikat.)

Die Prädikatenlogik erweitert die Aussagenlogik durch Hinzunahme von Prädikatensymbolen und Funktionssymbolen zu den quantifizierten booleschen Formeln. Wieder definieren wir sowohl die Syntax als auch die Semantik der Prädikatenlogik. Um Formeln der Prädikatenlogik einzuführen, benötigen wir zunächst den Begriff der Terme, aus welchen diese Formeln bestehen.

Definition 2.30 (Syntax der Prädikatenlogik).

- Seien x_1, x_2, \dots Variablen. Ein Prädikatensymbol hat die Form P_i^k und ein Funktionssymbol hat die Form f_i^k , wobei der obere Index $k \in \mathbb{N}$ die Stelligkeit dieser Symbole bezeichnet und der untere Index $i \in \{1, 2, \dots\}$ zu ihrer Unterscheidung dient. Funktionssymbole der Stelligkeit null werden auch Konstanten genannt.
- Terme sind wie folgt induktiv definiert:
 1. Jede Variable ist ein Term.
 2. Ist f_i^k ein k -stelliges Funktionssymbol und sind t_1, t_2, \dots, t_k Terme, so ist $f_i^k(t_1, t_2, \dots, t_k)$ ein Term. Für $k = 0$ ist f_i^0 eine Konstante, d.h. ein Element des gegebenen Universums.
- In der Prädikatenlogik werden Formeln induktiv wie folgt definiert:
 1. Ist P_i^k ein k -stelliges Prädikatensymbol und sind t_1, t_2, \dots, t_k Terme, so ist $P_i^k(t_1, t_2, \dots, t_k)$ eine (atomare) Formel.
 2. Ist F eine Formel, so ist $\neg F$ eine Formel.
 3. Sind F und G Formeln, so sind $F \vee G$ und $F \wedge G$ Formeln.
 4. Ist x eine Variable und ist F eine Formel, so sind $(\exists x)F$ und $(\forall x)F$ Formeln.

Beispiel 2.31 (Syntax der Prädikatenlogik). Betrachte die durch

$$\begin{aligned} F = & (\forall x) [Q(a(x, 1), s(x)) \wedge (\exists y) [P(y) \wedge P(s(y))] \\ & \wedge ((\neg Q(x, y) \wedge P(x)) \implies \neg P(s(x)))], \end{aligned}$$

definierte Formel F , mit den Prädikatensymbolen P und Q und den Funktionssymbolen a und s . Zur Vereinfachung verzichten wir hier auf obere und untere Indizes.

Um der Syntax der Formeln in der Prädikatenlogik Bedeutung zu verleihen, definieren wir nun den Begriff einer *Struktur*, die aus einem Universum von Objekten und einer Abbildung besteht, welche die syntaktischen Bausteine der Formel geeignet interpretiert – die Prädikatensymbole (also P und Q in der Formel F oben) und die Terme (also x , y , 1 , a und s in F). Auf der Grundlage dieser Interpretation kann dann der Wahrheitswert einer solchen Formel definiert werden.

Definition 2.32 (Semantik der prädikatenlogischen Formeln).

- Eine Struktur ist ein Paar $\mathfrak{A} = (U_{\mathfrak{A}}, I_{\mathfrak{A}})$, wobei $U_{\mathfrak{A}}$ eine als Universum bezeichnete nichtleere Menge und $I_{\mathfrak{A}}$ eine Abbildung von der Menge

$$\{x_i, P_i^k, f_i^k \mid i, k \in \mathbb{N} \text{ und } i \geq 1\}$$

in die Menge

$$U_{\mathfrak{A}} \cup \{P \mid P \text{ ist ein Prädikat auf } U_{\mathfrak{A}}\} \cup \{f \mid f \text{ ist eine Funktion auf } U_{\mathfrak{A}}\}$$

ist. Das heißt, $I_{\mathfrak{A}}$ interpretiert jede Variable x_i , jedes Prädikatsymbol P_i^k und jedes Funktionssymbol f_i^k , für welche $I_{\mathfrak{A}}$ definiert ist. Dabei wird x_i auf ein Element von $U_{\mathfrak{A}}$ abgebildet, P_i^k auf ein k -stelliges Prädikat auf $U_{\mathfrak{A}}$ bzw. f_i^k auf eine k -stellige Funktion auf $U_{\mathfrak{A}}$. Wieder dürfen obere und untere Indizes von x_i , P_i^k und f_i^k weggelassen werden. Wir verwenden die Abkürzung $x^{\mathfrak{A}}$ für $I_{\mathfrak{A}}(x)$, $P^{\mathfrak{A}}$ für $I_{\mathfrak{A}}(P)$, und $f^{\mathfrak{A}}$ für $I_{\mathfrak{A}}(f)$.

- Eine Struktur $\mathfrak{A} = (U_{\mathfrak{A}}, I_{\mathfrak{A}})$ ist zu einer Formel F passend, falls $I_{\mathfrak{A}}$ für alle freien Variablen, Prädikatsymbole und Funktionssymbole, die in F vorkommen, definiert ist.
- Sei F eine Formel, und sei $\mathfrak{A} = (U_{\mathfrak{A}}, I_{\mathfrak{A}})$ eine zu F passende Struktur. Für jeden Term t , der in F vorkommt, definieren wir den Wert von t in \mathfrak{A} , bezeichnet mit $\mathfrak{A}(t)$, induktiv wie folgt:
 1. Ist $t = x$ eine Variable, so ist $\mathfrak{A}(t) = x^{\mathfrak{A}}$.
 2. Ist $t = f(t_1, t_2, \dots, t_k)$, wobei t_1, t_2, \dots, t_k Terme sind und f ein k -stelliges Funktionssymbol ist, so ist $\mathfrak{A}(t) = f^{\mathfrak{A}}(\mathfrak{A}(t_1), \mathfrak{A}(t_2), \dots, \mathfrak{A}(t_k))$.
- Der Wahrheitswert der Formel F in der Struktur \mathfrak{A} , bezeichnet mit $\mathfrak{A}(F)$, ist induktiv wie folgt definiert:
 1. Ist $F = P(t_1, t_2, \dots, t_k)$, wobei P ein k -stelliges Prädikatsymbol ist und t_1, t_2, \dots, t_k Terme sind, so ist

$$\mathfrak{A}(F) = \begin{cases} 1 & \text{falls } (\mathfrak{A}(t_1), \mathfrak{A}(t_2), \dots, \mathfrak{A}(t_k)) \in P^{\mathfrak{A}} \\ 0 & \text{sonst.} \end{cases}$$

2. Ist $F = \neg G$, so ist

$$\mathfrak{A}(F) = \begin{cases} 1 & \text{falls } \mathfrak{A}(G) = 0 \\ 0 & \text{sonst.} \end{cases}$$

3. Ist $F = G \vee H$, so ist

$$\mathfrak{A}(F) = \begin{cases} 1 & \text{falls } \mathfrak{A}(G) = 1 \text{ oder } \mathfrak{A}(H) = 1 \\ 0 & \text{sonst.} \end{cases}$$

4. Ist $F = G \wedge H$, so ist

$$\mathfrak{A}(F) = \begin{cases} 1 & \text{falls } \mathfrak{A}(G) = \mathfrak{A}(H) = 1 \\ 0 & \text{sonst.} \end{cases}$$

5. Für jede Variable x und für jedes Element $a \in U_{\mathfrak{A}}$ bezeichne $\mathfrak{A}_{(x:a)} = (U_{\mathfrak{A}}, I_{\mathfrak{A}_{(x:a)}})$ die Struktur, die mit \mathfrak{A} übereinstimmt, bis auf dass $x^{\mathfrak{A}_{(x:a)}} = a$ gilt, unabhängig davon, ob $I_{\mathfrak{A}}$ auf x definiert ist oder nicht. Ist $F = (\exists x) G$, so ist

$$\mathfrak{A}(F) = \begin{cases} 1 & \text{falls es ein } a \in U_{\mathfrak{A}} \text{ gibt mit } \mathfrak{A}_{(x:a)}(G) = 1 \\ 0 & \text{sonst.} \end{cases}$$

6. Ist $F = (\forall x) G$, so ist

$$\mathfrak{A}(F) = \begin{cases} 1 & \text{falls f\"ur alle } a \in U_{\mathfrak{A}}, \mathfrak{A}_{(x:a)}(G) = 1 \\ 0 & \text{sonst.} \end{cases}$$

- Eine zu einer Formel F passende Struktur \mathfrak{A} erfüllt F (oder ist ein Modell von F), falls $\mathfrak{A}(F) = 1$. Eine Formel F ist erfüllbar, falls ein Modell f\"ur sie existiert. F heit gltig, falls jede zu F passende Struktur ein Modell von F ist.
- Je zwei Formeln F und G sind (semantisch) quivalent (bezeichnet mit $F \equiv G$), falls f\"ur jede zu F und G passende Struktur \mathfrak{A} gilt: $\mathfrak{A}(F) = \mathfrak{A}(G)$.

Beispiel 2.33 (Semantik der Prdikatenlogik). Betrachte die Formel

$$G = (\forall x) [Q(a(x, 1), s(x)) \wedge (P(x) \implies \neg P(s(x)))].$$

Eine zu G passende Struktur $\mathfrak{A} = (U_{\mathfrak{A}}, I_{\mathfrak{A}})$ ist gegeben durch:

- $U_{\mathfrak{A}} = \mathbb{N}$, die Menge der natrlichen Zahlen,
- $I_{\mathfrak{A}}(P) = P^{\mathfrak{A}} = \{n \in U_{\mathfrak{A}} \mid n \text{ ist eine Primzahl}\}$,
- $I_{\mathfrak{A}}(Q) = Q^{\mathfrak{A}} = \{(m, n) \mid m, n \in U_{\mathfrak{A}} \text{ und } m = n\}$,
- $I_{\mathfrak{A}}(a) = a^{\mathfrak{A}}$ ist die Additionsfunktion auf $U_{\mathfrak{A}}$, d.h., $a^{\mathfrak{A}}(m, n) = m + n$,
- $I_{\mathfrak{A}}(s) = s^{\mathfrak{A}}$ ist die Nachfolgerfunktion auf $U_{\mathfrak{A}}$, d.h., $s^{\mathfrak{A}}(n) = n + 1$ und
- $I_{\mathfrak{A}}(1) = 1^{\mathfrak{A}}$ ist die Konstante $1 \in U_{\mathfrak{A}}$.

Natrlich ist das Prdikat $Q(a(x, 1), s(x))$ fr jedes x in \mathbb{N} wahr. Die Implikation $(P(x) \implies \neg P(s(x)))$, die behauptet, dass der Nachfolger einer jeden Primzahl keine Primzahl sei, ist falsch: Es gibt eine Primzahl (nmlich $2 \in \mathbb{N}$), deren Nachfolger, 3 , ebenfalls prim ist. Somit ist die geschlossene Formel G in der gegebenen Struktur \mathfrak{A} falsch. Man berlege sich eine andere zu G passende Struktur, in der G wahr ist.

Im Gegensatz dazu ist die geschlossene Formel

$$\begin{aligned} F = & (\forall x) [Q(a(x, 1), s(x)) \wedge (\exists y) [P(y) \wedge P(s(y))] \\ & \wedge ((\neg Q(x, y) \wedge P(x)) \implies \neg P(s(x)))] \end{aligned}$$

aus Beispiel 2.31 in der oben angegebenen Struktur \mathfrak{A} wahr, denn die Variable y kann als $2 \in U_{\mathfrak{A}}$ interpretiert werden. Dann sind sowohl $y = 2$ als auch $s(y) = 3$ Primzahlen, und der Nachfolger einer jeden ungeraden Primzahl ist nicht prim. Man berlege sich eine andere zu F passende Struktur, in der F falsch ist.

Es ist wichtig, sich das Zusammenspiel zwischen Syntax und Semantik von prdikatenlogischen Formeln vor Augen zu halten, insbesondere zwischen den syntaktischen Objekten einer Struktur \mathfrak{A} und der semantischen Interpretation der Variablen, Prdikatensymbole und Funktionssymbole als $x^{\mathfrak{A}}$, $P^{\mathfrak{A}}$ und $f^{\mathfrak{A}}$. Indem man unterschiedliche Strukturen fr eine gegebene Formel F definiert, erhlt man unterschiedliche Interpretationen von F .

Die oben festgehaltenen Aussagen und Eigenschaften, die auf (quantifizierte) boolesche Formeln der Aussagenlogik zutreffen, lassen sich unmittelbar auf Formeln

der Prädikatenlogik übertragen. Beispielsweise ist auch jede solche Formel genau dann gültig, wenn ihre Negation unerfüllbar ist.

Wie oben bemerkt, erweitert die Prädikatenlogik die Aussagenlogik. Tatsächlich erhält man die Aussagenlogik, wenn man in der Prädikatenlogik verlangt, dass alle Prädikatensymbole die Stelligkeit null haben. Dann sind die Begriffe Term, Variable und Quantor überflüssig und die null-stelligen Prädikatensymbole spielen die Rolle der atomaren Formeln der Aussagenlogik. Genau genommen genügt es schon, Variablen (und somit Quantoren) aus der Prädikatenlogik zu verbannen, um diese zur Aussagenlogik (ohne Quantoren) entarten zu lassen.

Um Verwirrungen zu vermeiden, sei hier betont, dass die in der Aussagenlogik verwendeten booleschen Variablen und Konstanten von den Variablen und Konstanten der Prädikatenlogik zu unterscheiden sind. Die ersten werden einfach als Wahrheitswerte interpretiert, wohingegen die letzteren als Objekte (oder Individuen) des Universums in einer gegebenen Struktur zu interpretieren sind.

Obwohl die Prädikatenlogik ausdrucksstärker als die Aussagenlogik ist, kann man doch noch immer nicht alles in dieser formulieren, was man vielleicht gern ausdrücken möchte. Die hier vorgestellte Prädikatenlogik ist die so genannte Logik erster Stufe („*first-order logic*“), in welcher nur Variablen quantifiziert werden dürfen. Erlaubt man außerdem die Quantifizierung von Prädikaten- und Funktionsymbolen, so erhält man die so genannte Logik zweiter Stufe („*second-order logic*“), die die Ausdrucksstärke der Logik erster Stufe übertrifft. Dieses Thema soll hier jedoch nicht weiter vertieft werden.

2.4 Algebra, Zahlentheorie und Graphentheorie

Die Algorithmen und Probleme, die in den nachfolgenden Kapiteln behandelt werden, erfordern einige Grundlagen der Algebra und insbesondere Begriffe aus der Gruppentheorie, Zahlentheorie und Graphentheorie. Dieser Abschnitt kann ebenso gut vorerst übersprungen werden; die hier präsentierten Definitionen und Resultate kann man später wieder aufsuchen, wenn sie benötigt werden. Wieder verzichten wir hier auf die meisten Beweise.

2.4.1 Algebra und Zahlentheorie

Zunächst werden einige fundamentale algebraische Strukturen eingeführt.

Definition 2.34 (Gruppe, Ring und Körper).

- Eine Gruppe $\mathfrak{G} = (S, \circ)$ besteht aus einer nichtleeren Menge S und einer Binäroperation \circ auf S , so dass die folgenden Axiome erfüllt sind:
 - **Abschluss:** $(\forall x \in S) (\forall y \in S) [x \circ y \in S]$.
 - **Assoziativität:** $(\forall x \in S) (\forall y \in S) (\forall z \in S) [(x \circ y) \circ z = x \circ (y \circ z)]$.
 - **Neutrales Element:** $(\exists e \in S) (\forall x \in S) [e \circ x = x \circ e = x]$.
 - **Inverses Element:** $(\forall x \in S) (\exists x^{-1} \in S) [x \circ x^{-1} = x^{-1} \circ x = e]$.

Das Element e heißt das neutrale Element der Gruppe \mathfrak{G} . Das Element x^{-1} heißt das inverse Element von x . Definiere die Ordnung eines Elements x von \mathfrak{G} als die kleinste positive Zahl k mit $x^k = \underbrace{x \circ x \circ \cdots \circ x}_{k\text{-mal}} = e$.

- $\mathfrak{M} = (S, \circ)$ ist ein Monoid, falls Assoziativität und Abschluss unter \circ in \mathfrak{M} gelten. Ein Monoid \mathfrak{M} muss kein neutrales Element haben, und nicht jedes Element in \mathfrak{M} muss invertierbar sein.
- Eine Gruppe $\mathfrak{G} = (S, \circ)$ (bzw. ein Monoid $\mathfrak{M} = (S, \circ)$) ist kommutativ (oder abelsch), falls für alle $x, y \in S$ gilt: $x \circ y = y \circ x$. Die Anzahl der Elemente einer endlichen Gruppe \mathfrak{G} wird als die Ordnung von \mathfrak{G} und mit $\|\mathfrak{G}\|$ bezeichnet.
- Sei $\mathfrak{G} = (S, \circ)$ eine Gruppe. $\mathfrak{H} = (T, \circ)$ ist eine Untergruppe von \mathfrak{G} (bezeichnet mit $\mathfrak{H} \leq \mathfrak{G}$), falls $T \subseteq S$ und die Gruppenaxiome für \mathfrak{H} gelten.
- Ein Ring ist ein Tripel $\mathfrak{R} = (S, +, \cdot)$, so dass $(S, +)$ eine abelsche Gruppe und (S, \cdot) ein Monoid ist und die Distributivgesetze für alle x, y und z in S gelten:

$$\begin{aligned} x \cdot (y + z) &= (x \cdot y) + (x \cdot z); \\ (x + y) \cdot z &= (x \cdot z) + (y \cdot z). \end{aligned}$$

- Ein Ring $\mathfrak{R} = (S, +, \cdot)$ ist kommutativ, falls der Monoid (S, \cdot) kommutativ ist.
- Sei $\mathfrak{R} = (S, +, \cdot)$ ein Ring. Das neutrale Element der Gruppe $(S, +)$ heißt das Nullelement (kurz die Null) von \mathfrak{R} . Das neutrale Element des Monoids (S, \cdot) heißt, falls es existiert, das Einselement (kurz die Eins) von \mathfrak{R} .
- Sei $\mathfrak{R} = (S, +, \cdot)$ ein Ring mit Eins. Ein Element x von \mathfrak{R} ist invertierbar, falls es in dem Monoid (S, \cdot) invertierbar ist.
- Ein Körper ist ein kommutativer Ring mit Eins, in dem jedes von Null verschiedene Element invertierbar ist.

Das neutrale Element und die inversen Elemente sind eindeutig bestimmt, falls sie existieren. Betrachte die folgenden einfachen Beispiele.

Beispiel 2.35 (Gruppe, Ring und Körper).

1. Sei $k \in \mathbb{N}$. Die Menge $\mathbb{Z}_k = \{0, 1, \dots, k-1\}$ ist eine endliche Gruppe bezüglich der Addition modulo k und hat das neutrale Element 0. Die Arithmetik modulo einer Zahl wird in Problem 2.1 am Ende dieses Kapitels erklärt. Bezuglich Addition und Multiplikation modulo k ist \mathbb{Z}_k ein kommutativer Ring mit Eins, siehe auch Problem 2.1. Ist p eine Primzahl (d.h., $p \geq 2$ ist nur durch 1 und durch p teilbar), dann ist \mathbb{Z}_p ein Körper bezüglich Addition und Multiplikation modulo p .
2. Der größte gemeinsame Teiler $\text{ggT}(n, m)$ zweier ganzer Zahlen m und n wurde in Abschnitt 2.1 definiert. Definiere für ein beliebiges festes $k \in \mathbb{N}$ die Menge

$$\mathbb{Z}_k^* = \{i \mid 1 \leq i \leq k-1 \text{ und } \text{ggT}(i, k) = 1\}.$$

Bezuglich der Multiplikation modulo k ist \mathbb{Z}_k^* eine endliche Gruppe mit dem neutralen Element 1.

Wenn die Operation \circ einer Gruppe $\mathfrak{G} = (S, \circ)$ aus dem Zusammenhang klar ist, wird sie nicht explizit angegeben. Die Gruppe \mathbb{Z}_k^* aus Beispiel 2.35 ist besonders in Abschnitt 7.1 wichtig, in dem das RSA-Kryptosystem eingeführt wird.

Definition 2.36 (Euler-Funktion). Für eine positive Zahl k gibt die Euler-Funktion $\varphi(k)$ die Anzahl der positiven Zahlen $i \leq k$ mit $\text{ggT}(i, k) = 1$ an.

Für $k = 1, 2, 3, 4, 5, 6, \dots$ ergeben sich die Werte $\varphi(k)$ als $1, 1, 2, 2, 4, 2, \dots$; insbesondere gibt $\varphi(k)$ für $k > 1$ die Ordnung der Gruppe \mathbb{Z}_k^* an, d.h., $\varphi(k) = \|\mathbb{Z}_k^*\|$.

Die folgenden Eigenschaften von φ folgen aus der Definition:

- $\varphi(m \cdot n) = \varphi(m) \cdot \varphi(n)$ für alle $m, n \in \mathbb{N}$ mit $\text{ggT}(m, n) = 1$, und
- $\varphi(p) = p - 1$ für jede Primzahl p .

Der Beweis dieser Eigenschaften wird dem Leser als Übung 2.14 überlassen. Aus ihnen folgt unmittelbar der folgende Fakt, der in Abschnitt 7.1 gebraucht wird.

Fakt 2.37 Ist $n = p \cdot q$ für Primzahlen p und q mit $p \neq q$, so ist $\varphi(n) = (p - 1)(q - 1)$.

Der Satz von Euler ist ein Spezialfall des Satzes von Lagrange, welcher sagt, dass die Ordnung einer jeden endlichen Gruppe \mathfrak{G} durch die Ordnung einer jeden Untergruppe von \mathfrak{G} geteilt wird. Da die Ordnung eines jeden Gruppenelements a gleich der Ordnung der von a erzeugten Untergruppe ist, folgt, dass die Ordnung von a die Gruppenordnung teilt. Bezeichnet e das neutrale Element von \mathfrak{G} , so ergibt sich $a^k = e$. Da \mathbb{Z}_n^* eine endliche multiplikative Gruppe der Ordnung $\varphi(n)$ ist, folgt daraus der Satz von Euler. Der Spezialfall des Satzes von Euler, bei dem n eine zu a teilerfremde Primzahl ist, ist bekannt als Fermats Kleiner Satz.

Satz 2.38 (Euler). Für jedes $a \in \mathbb{Z}_n^*$ gilt $a^{\varphi(n)} \equiv 1 \pmod{n}$.

Korollar 2.39 (Kleiner Satz von Fermat). Ist p eine Primzahl und ist $a \in \mathbb{Z}_p^*$, dann gilt $a^{p-1} \equiv 1 \pmod{p}$.

Wir definieren nun einige zahlentheoretische Begriffe, die für die kryptographischen Protokolle in den nachfolgenden Kapiteln zentral sind.

Definition 2.40 (Primitivwurzel). Eine Primitivwurzel einer Zahl $n \in \mathbb{N}$ ist ein Element $r \in \mathbb{Z}_n^*$ für das $r^d \not\equiv 1 \pmod{n}$ für jedes d mit $1 \leq d < \varphi(n)$ gilt.

Definiert man diesen Begriff allgemein für eine (endliche) Gruppe anstatt für \mathbb{Z}_n^* , so bezeichnet man ihn als primitives Element. Eine Primitivwurzel r von n erzeugt die gesamte Gruppe \mathbb{Z}_n^* . Das heißt, es gilt $\mathbb{Z}_n^* = \{r^i \mid 0 \leq i < \varphi(n)\}$. Man erinnere sich, dass \mathbb{Z}_p^* für jede Primzahl p eine Gruppe der Ordnung $\varphi(p) = p - 1$ ist. \mathbb{Z}_p^* hat genau $\varphi(p - 1)$ Primitivwurzeln, siehe auch Übung 2.15.

Beispiel 2.41 (Primitivwurzel). Betrachte $\mathbb{Z}_5^* = \{1, 2, 3, 4\}$. Da $\mathbb{Z}_4^* = \{1, 3\}$, gilt $\varphi(4) = 2$, und die beiden Primitivwurzeln von 5 sind 2 und 3. Sowohl 2 als auch 3 erzeugen ganz \mathbb{Z}_5^* , denn

$$\begin{aligned} 2^0 &= 1; & 2^1 &= 2; & 2^2 &= 4; & 2^3 &\equiv 3 \pmod{5}; \\ 3^0 &= 1; & 3^1 &= 3; & 3^2 &\equiv 4 \pmod{5}; & 3^3 &\equiv 2 \pmod{5}. \end{aligned}$$

Nicht jede natürliche Zahl hat eine Primitivwurzel; 8 ist das kleinste solche Beispiel. Es ist aus der elementaren Zahlentheorie bekannt, dass eine Zahl n genau dann eine Primitivwurzel hat, wenn n entweder zu $\{1, 2, 4\}$ gehört oder von der Form $n = q^k$ oder $n = 2q^k$ für eine ungerade Primzahl q ist.

Definition 2.42 (Diskreter Logarithmus). Sei p eine Primzahl, und sei r eine Primitivwurzel von p . Die modulare Exponentialfunktion mit Basis r und Modulus p ist die Funktion $\exp_{r,p}$, die von \mathbb{Z}_{p-1} in \mathbb{Z}_p^* abbildet und durch

$$\exp_{r,p}(a) = r^a \bmod p$$

definiert ist. Ihre Umkehrfunktion heißt der diskrete Logarithmus und bildet, für feste Zahlen p und r , den Wert $\alpha = \exp_{r,p}(a)$ auf a ab. Ist $\alpha = \exp_{r,p}(a)$, so schreiben wir $a = \log_r \alpha \bmod p$.

Definition 2.43 (Quadratischer Rest und Nichtrest).

- Für $n \in \mathbb{N}$ heißt ein Element $x \in \mathbb{Z}_n^*$ quadratischer Rest modulo n , falls es ein $w \in \mathbb{Z}_n$ mit $x \equiv w^2 \bmod n$ gibt. Andernfalls heißt x ein quadratischer Nichtrest modulo n .
- Definiere die Entscheidungsprobleme

$$\text{QR} = \{(x, n) \mid x \in \mathbb{Z}_n^*, n \in \mathbb{N} \text{ und } x \text{ ist quadratischer Rest modulo } n\};$$

$$\text{QNR} = \{(x, n) \mid x \in \mathbb{Z}_n^*, n \in \mathbb{N} \text{ und } x \text{ ist quadratischer Nichtrest modulo } n\},$$

wobei x und n binär repräsentiert sind.

Die oben definierten Begriffe des quadratischen Restes und Nichtrestes und die zugehörigen Entscheidungsprobleme QR und QNR sind für eine Reihe von Kryptosystemen wichtig, die in den nachfolgenden Kapiteln eingeführt werden.

Aus dem Eulerschen Kriterium unten erhält man einen deterministischen Polynomialzeit-Algorithmus für QR, sofern der Modulus eine ungerade Primzahl ist.

Satz 2.44 (Eulersches Kriterium). Sei p eine ungerade Primzahl. Dann ist x genau dann ein quadratischer Rest modulo p , wenn

$$x^{(p-1)/2} \equiv 1 \bmod p.$$

Beweis. Angenommen, x ist ein quadratischer Rest modulo p , d.h., $x \equiv w^2 \bmod p$ für ein $w \in \mathbb{Z}_p^*$. Nach dem Kleinen Satz von Fermat (siehe Korollar 2.39) gilt $w^{p-1} \equiv 1 \bmod p$. Somit ist

$$x^{(p-1)/2} \equiv (w^2)^{(p-1)/2} \equiv w^{p-1} \equiv 1 \bmod p.$$

Nimm umgekehrt an, dass $x^{(p-1)/2} \equiv 1 \bmod p$ gilt. Sei r eine Primitivwurzel modulo p . Dann gilt $x \equiv r^i \bmod p$ für ein i . Es folgt, dass

$$x^{(p-1)/2} \equiv (r^i)^{(p-1)/2} \equiv r^{i(p-1)/2} \equiv 1 \bmod p.$$

Da r die Ordnung $p - 1$ hat, folgt, dass $p - 1$ ein Teiler von $i(p - 1)/2$ ist. Folglich ist i gerade, und die beiden Quadratwurzeln von x sind $\pm r^{i/2}$. \square

Damit wir uns später darauf beziehen können, werden als nächstes das Legendre- und das Jacobi-Symbol definiert.

Definition 2.45 (Legendre-Symbol und Jacobi-Symbol).

- Für $m \in \mathbb{Z}$ und eine Primzahl p ist das Legendre-Symbol $\left(\frac{m}{p}\right)$ definiert durch

$$\left(\frac{m}{p}\right) = \begin{cases} 0 & \text{falls } m \equiv 0 \pmod{p} \\ 1 & \text{falls } m \text{ ein quadratischer Rest modulo } p \text{ ist} \\ -1 & \text{falls } m \text{ ein quadratischer Nichtrest modulo } p \text{ ist.} \end{cases}$$

- Sei $m \in \mathbb{Z}$, und sei $n > 2$ eine ungerade Zahl, die sich durch $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ als Produkt von Primzahlpotenzen darstellen lässt. Das Jacobi-Symbol $\left(\frac{m}{n}\right)$ verallgemeinert das Legendre-Symbol für zusammengesetzte „Nenner“ n und ist definiert durch

$$\left(\frac{m}{n}\right) = \prod_{i=1}^k \left(\frac{m}{p_i}\right)^{e_i}.$$

Nach dem Eulerschen Kriterium (siehe Satz 2.44) kann man das Legendre-Symbol effizient berechnen, denn für jede ungerade Primzahl p gilt

$$\left(\frac{x}{p}\right) \equiv x^{(p-1)/2} \pmod{p}. \quad (2.8)$$

Unter Verwendung bestimmter zahlentheoretischer Fakten kann auch das Jacobi-Symbol $\left(\frac{m}{n}\right)$ in Polynomialzeit berechnet werden, ohne Kenntnis der Primfaktorisierung von n .

Schließlich geben wir noch einen weiteren nützlichen zahlentheoretischen Fakt ohne Beweis an.

Satz 2.46 (Chinesischer Restesatz). Seien m_1, m_2, \dots, m_k positive ganze Zahlen, die paarweise teilerfremd sind (d.h., $\text{ggT}(m_i, m_j) = 1$ für $i \neq j$). Sei

$$M = \prod_{i=1}^k m_i,$$

und seien a_1, a_2, \dots, a_k beliebige ganze Zahlen. Für jedes i mit $1 \leq i \leq k$ definieren wir $q_i = M/m_i$ und bezeichnen mit q_i^{-1} das inverse Element von q_i in $\mathbb{Z}_{m_i}^*$. Dann hat das System von k Kongruenzen $x \equiv a_i \pmod{m_i}$, mit $1 \leq i \leq k$, die eindeutige Lösung

$$x = \sum_{i=1}^k a_i q_i q_i^{-1} \pmod{M}.$$

2.4.2 Permutationsgruppen

Im Abschnitt 6.5 wird es um Algorithmen für das Graphisomorphie-Problem gehen, und Abschnitt 8.4 wird ein Zero-Knowledge-Protokoll für dieses Problem präsentieren. Das Graphisomorphie-Problem kann als ein Spezialfall bestimmter gruppentheoretischer Probleme aufgefasst werden. Insbesondere sind hier Permutationsgruppen wichtig. Diese werden nun eingeführt; ein illustratives Beispiel für die dabei definierten Begriffe ist in Abschnitt 2.4.3 zu finden.

Definition 2.47 (Permutationsgruppe).

- Eine Permutation ist eine bijektive Abbildung einer Menge auf sich selbst. Für jede natürliche Zahl $n \geq 1$ bezeichne $[n]$ die Menge $\{1, 2, \dots, n\}$. Die Menge aller Permutationen von $[n]$ wird mit \mathfrak{S}_n bezeichnet. Für algorithmische Zwecke stellen wir Permutationen $\pi \in \mathfrak{S}_n$ als Listen von n geordneten Paaren $(i, \pi(i))$ aus $[n] \times [n]$ dar.
- Die Komposition zweier Permutationen π und τ aus \mathfrak{S}_n ist als die Permutation $\pi\tau$ in \mathfrak{S}_n definiert, die man erhält, wenn man zuerst π und dann τ auf die Elemente von $[n]$ anwendet, d.h., $(\pi\tau)(i) = \tau(\pi(i))$ für jedes $i \in [n]$. \mathfrak{S}_n ist eine Permutationsgruppe bezüglich der Komposition von Permutationen. Das neutrale Element ist dabei die identische Permutation, definiert als $\text{id}(i) = i$ für jedes $i \in [n]$. Die Untergruppe von \mathfrak{S}_n , die id als ihr einziges Element enthält, wird mit **id** bezeichnet.
- Sei \mathfrak{T} eine Teilmenge von \mathfrak{S}_n . Definiere die von \mathfrak{T} erzeugte Permutationsgruppe $\langle \mathfrak{T} \rangle$ als die kleinste Untergruppe von \mathfrak{S}_n , die \mathfrak{T} enthält. Untergruppen \mathfrak{G} von \mathfrak{S}_n werden durch ihre Erzeuger (auch Generatoren genannt) dargestellt. In \mathfrak{G} ist der Orbit eines Elements $i \in [n]$ definiert durch

$$\mathfrak{G}(i) = \{\pi(i) \mid \pi \in \mathfrak{G}\}.$$

- Für jede Teilmenge T von $[n]$ bezeichne \mathfrak{G}_n^T die Untergruppe von \mathfrak{S}_n , die jedes Element von T auf sich selbst abbildet. Insbesondere ist für $i \leq n$ und jede Untergruppe \mathfrak{G} von \mathfrak{S}_n der (punktweise) Stabilisator von $[i]$ in \mathfrak{G} definiert durch

$$\mathfrak{G}^{(i)} = \{\pi \in \mathfrak{G} \mid \pi(j) = j \text{ für jedes } j \in [i]\}.$$

- Es gilt $\mathfrak{G}^{(n)} = \text{id}$ und $\mathfrak{G}^{(0)} = \mathfrak{G}$.
- Seien \mathfrak{G} und \mathfrak{H} Permutationsgruppen mit $\mathfrak{H} \leq \mathfrak{G}$. Für $\tau \in \mathfrak{G}$ ist die rechte co-Menge von \mathfrak{H} in \mathfrak{G} definiert durch

$$\mathfrak{H}\tau = \{\pi\tau \mid \pi \in \mathfrak{H}\}.$$

Je zwei rechte co-Mengen von \mathfrak{H} in \mathfrak{G} sind entweder identisch oder disjunkt. Deshalb kann die Permutationsgruppe \mathfrak{G} in rechte co-Mengen von \mathfrak{H} in \mathfrak{G} zerlegt werden:

$$\mathfrak{G} = \mathfrak{H}\tau_1 \cup \mathfrak{H}\tau_2 \cup \dots \cup \mathfrak{H}\tau_k. \quad (2.9)$$

Jede rechte co-Menge \mathfrak{H} in \mathfrak{G} hat die Kardinalität $\|\mathfrak{H}\|$. Die Menge $\{\tau_1, \tau_2, \dots, \tau_k\}$ aus (2.9) heißt die *vollständige rechte Transversale von \mathfrak{H} in \mathfrak{G}* .

Der Begriff des punktweisen Stabilisators ist besonders wichtig für den Entwurf von Algorithmen für Probleme auf Permutationsgruppen. Die entscheidende Struktur, die man hier verwendet, ist der so genannte „*Turm von Stabilisatoren*“ einer gegebenen Permutationsgruppe \mathfrak{G} :

$$\mathbf{id} = \mathfrak{G}^{(n)} \leq \mathfrak{G}^{(n-1)} \leq \dots \leq \mathfrak{G}^{(1)} \leq \mathfrak{G}^{(0)} = \mathfrak{G}.$$

Für jedes i mit $1 \leq i \leq n$ sei \mathfrak{T}_i eine vollständige rechte Transversale von $\mathfrak{G}^{(i)}$ in $\mathfrak{G}^{(i-1)}$. Dann heißt $\mathfrak{T} = \bigcup_{i=1}^{n-1} \mathfrak{T}_i$ ein *starker Generator* von \mathfrak{G} . Es gilt $\mathfrak{G} = \langle \mathfrak{T} \rangle$. Jedes $\pi \in \mathfrak{G}$ hat eine eindeutige Faktorisierung $\pi = \tau_1 \tau_2 \dots \tau_n$, wobei $\tau_i \in \mathfrak{T}_i$.

Die folgenden grundlegenden algorithmischen Resultate über Permutationsgruppen werden später in den Abschnitten 6.5.1 und 6.5.2 nützlich sein. Auf den Beweis von Satz 2.48 wird hier verzichtet.

Satz 2.48. *Sei eine Permutationsgruppe $\mathfrak{G} \leq \mathfrak{S}_n$ durch ihren Generator gegeben. Dann gelten die folgenden zwei Aussagen.*

1. *Für jedes $i \in [n]$ kann der Orbit $\mathfrak{G}(i)$ von i in \mathfrak{G} in Polynomialzeit berechnet werden.*
2. *Der Turm von Stabilisatoren $\mathbf{id} = \mathfrak{G}^{(n)} \leq \mathfrak{G}^{(n-1)} \leq \dots \leq \mathfrak{G}^{(1)} \leq \mathfrak{G}^{(0)} = \mathfrak{G}$ kann in einer Zeit polynomiell in n berechnet werden. Das heißt, es gibt einen Polynomialzeit-Algorithmus, der die vollständigen rechten Transversalen \mathfrak{T}_i von $\mathfrak{G}^{(i)}$ in $\mathfrak{G}^{(i-1)}$ für jedes i mit $1 \leq i \leq n$ und somit einen starken Generator von \mathfrak{G} berechnet.*

2.4.3 Graphentheorie

Die in Definition 2.47 eingeführten Begriffe werden nun für konkrete Beispiele aus der Graphentheorie erklärt. Insbesondere betrachten wir die Automorphismengruppe eines gegebenen Graphen und die Menge der Isomorphismen zwischen zwei gegebenen Graphen. Zu diesem Zweck benötigen wir einige grundlegende Begriffe der Graphentheorie.

Definition 2.49 (Graphisomorphie und Graphautomorphie). Ein Graph G besteht aus einer endlichen Knotenmenge, $V(G)$, und einer endlichen Menge $E(G)$ von Kanten, die Knoten verbinden. G ist ein gerichteter Graph (bzw. ein ungerichteter Graph), falls die Kanten geordnete (bzw. ungeordnete) Paare von Knoten sind. Wir setzen voraus, dass es keine Mehrfachkanten zwischen zwei Knoten und auch keine reflexiven Kanten gibt. Das heißt, es gibt höchstens eine Kante, die je zwei Knoten verbindet, und es gibt keine Kante, die irgendeinen Knoten mit sich selbst verbindet. Die in diesem Buch betrachteten Graphen müssen im Allgemeinen nicht zusammenhängend sein, d.h., sie können mehr als eine Zusammenhangskomponente haben. In diesem Abschnitt konzentrieren wir uns auf ungerichtete Graphen; die entsprechenden Begriffe für gerichtete Graphen können analog definiert werden.

Seien G und H zwei gegebene Graphen. Die disjunkte Vereinigung von G und H , bezeichnet mit $G \cup H$, ist definiert als der Graph mit der Knotenmenge $V(G) \cup V(H)$, wobei $V(G)$ und $V(H)$ durch Umbenennen von Knoten disjunkt gemacht werden, falls nötig, und mit der Kantenmenge $E(G) \cup E(H)$.

Seien G und H Graphen mit derselben Anzahl von Knoten. Ein Isomorphismus zwischen G und H ist eine kantenerhaltende Bijektion von $V(G)$ auf $V(H)$. Das heißt, wenn wir $V(G) = \{1, 2, \dots, n\} = V(H)$ per Konvention festlegen, dann sind G und H genau dann isomorph (kurz $G \cong H$), wenn es eine Permutation $\pi \in S_n$ gibt, so dass für je zwei Knoten $i, j \in V(G)$ gilt:

$$\{i, j\} \in E(G) \iff \{\pi(i), \pi(j)\} \in E(H). \quad (2.10)$$

Ein Automorphismus von G ist eine kantenerhaltende Bijektion von $V(G)$ auf sich selbst. Jeder Graph enthält den trivialen Automorphismus id.

Bezeichne die Menge aller Isomorphismen zwischen G und H mit $\text{Iso}(G, H)$, und bezeichne die Menge aller Automorphismen von G mit $\text{Aut}(G)$. Definiere das Graphisomorphie-Problem (kurz GI) und das Graphautomorphie-Problem (kurz GA) durch

$$\begin{aligned} \text{GI} &= \{\langle G, H \rangle \mid G \text{ und } H \text{ sind isomorphe Graphen}\}; \\ \text{GA} &= \{G \mid G \text{ enthält einen nichttrivialen Automorphismus}\}. \end{aligned}$$

Für algorithmische Zwecke stellen wir Graphen entweder durch Listen ihrer Knoten und Kanten oder durch ihre Adjazenzmatrix dar, welche den Eintrag 1 an der Position (i, j) hat, falls $\{i, j\}$ eine Kante ist, und sonst den Eintrag 0. Diese Darstellung von Graphen wird geeignet über dem Alphabet $\Sigma = \{0, 1\}$ codiert. Paare von Graphen werden mittels einer Standard-Paarungsfunktion $\langle \cdot, \cdot \rangle$ repräsentiert, die bijektiv von $\Sigma^* \times \Sigma^*$ auf Σ^* abbildet, in Polynomialzeit berechenbar ist und polynomialzeit-berechenbare Inverse hat.

Diese Paarungsfunktion kann so erweitert werden, dass sie k -Tupel von Wörtern aus $(\Sigma^*)^k = \underbrace{\Sigma^* \times \Sigma^* \times \cdots \times \Sigma^*}_{k\text{-mal}}$ durch Wörter aus Σ^* codiert.

Beispiel 2.50 (Graphisomorphie und Graphautomorphie).

Die Graphen G und H in Abbildung 2.6 sind isomorph. Ein Isomorphismus π zwischen G und H , der die Adjazenz zwischen den Knoten gemäß (2.10) erhält, ist gegeben durch $\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 1 & 5 & 2 \end{pmatrix}$ oder, als Zyklen dargestellt, durch $\pi = (13)(245)$. Es gibt drei weitere Isomorphismen zwischen G und H , d.h., $\|\text{Iso}(G, H)\| = 4$, siehe Übung 2.16. Jedoch ist weder G noch H isomorph zu F . Das ist unmittelbar klar, wenn man sich die Folge der Knotengrade (d.h., die Anzahlen der aus den einzelnen Knoten auslaufenden Kanten) von G bzw. H ansieht, welche verschieden ist von der Folge der Knotengrade von F : Für G und H ist diese Folge $(2, 3, 3, 4, 4)$, wohingegen F die Folge $(3, 3, 3, 3, 4)$ hat.

Ein nichttrivialer Automorphismus $\varphi : V(G) \rightarrow V(G)$ von G ist gegeben durch $\varphi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & 4 & 3 & 5 \end{pmatrix}$ oder, als Zyklen dargestellt, durch $\varphi = (12)(34)(5)$; ein anderer ist

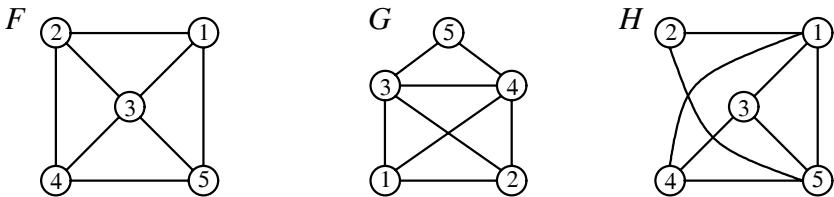


Abb. 2.6. Drei Graphen: G ist isomorph zu H , aber nicht zu F

gegeben durch $\tau = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 4 & 3 & 5 \end{pmatrix}$ oder $\tau = (1)(2)(34)(5)$. Es gibt zwei weitere Automorphismen von G , d.h., $\|\text{Aut}(G)\| = 4$, siehe Übung 2.16.

Die Permutationsgruppen $\text{Aut}(F)$, $\text{Aut}(G)$ und $\text{Aut}(H)$ sind Untergruppen von \mathfrak{S}_5 . Der Turm $\text{Aut}(G)^{(5)} \leq \text{Aut}(G)^{(4)} \leq \dots \leq \text{Aut}(G)^{(1)} \leq \text{Aut}(G)^{(0)}$ der Stabilisatoren von $\text{Aut}(G)$ besteht aus den Untergruppen $\text{Aut}(G)^{(5)} = \text{Aut}(G)^{(4)} = \text{Aut}(G)^{(3)} = \text{id}$, $\text{Aut}(G)^{(2)} = \text{Aut}(G)^{(1)} = \langle \{\text{id}, \tau\} \rangle$ und $\text{Aut}(G)^{(0)} = \text{Aut}(G)$. In der Automorphismengruppe $\text{Aut}(G)$ von G haben die Knoten 1 und 2 den Orbit $\{1, 2\}$, die Knoten 3 und 4 haben den Orbit $\{3, 4\}$, und der Knoten 5 hat den Orbit $\{5\}$.

Nun wird ein nützliches Lemma bewiesen, das später in Abschnitt 6.5.1 benötigt wird.

Lemma 2.51. Für je zwei Graphen G und H gilt:

$$\|\text{Iso}(G, H)\| = \begin{cases} \|\text{Aut}(G)\| = \|\text{Aut}(H)\| & \text{falls } G \cong H \\ 0 & \text{falls } G \not\cong H; \end{cases} \quad (2.11)$$

$$\|\text{Aut}(G \cup H)\| = \begin{cases} 2 \cdot \|\text{Aut}(G)\| \cdot \|\text{Aut}(H)\| & \text{falls } G \cong H \\ \|\text{Aut}(G)\| \cdot \|\text{Aut}(H)\| & \text{falls } G \not\cong H. \end{cases} \quad (2.12)$$

Beweis. $\text{Iso}(G, H)$ und $\text{Aut}(G)$ sind genau dann gleich groß, wenn G und H isomorph sind. Denn wenn G und H isomorph sind, dann impliziert $\text{Aut}(G) = \text{Iso}(G, G)$, dass $\|\text{Iso}(G, H)\| = \|\text{Aut}(G)\|$. Ist umgekehrt $G \not\cong H$, so ist $\text{Iso}(G, H)$ leer, wohingegen $\text{Aut}(G)$ stets den trivialen Automorphismus id enthält. Daraus folgt die Behauptung (2.11) von Lemma 2.51.

Um die Behauptung (2.12) zu beweisen, nehmen wir an, dass G und H jeweils aus einer Zusammenhangskomponente bestehen; andernfalls betrachten wir die co-Graphen \overline{G} und \overline{H} statt G und H , siehe Übung 2.17. Ein Automorphismus von $G \cup H$, der die Knoten von G und H vertauscht, besteht aus einem Isomorphismus in $\text{Iso}(G, H)$ und einem Isomorphismus in $\text{Iso}(H, G)$. Somit gilt $\|\text{Aut}(G \cup H)\| = \|\text{Aut}(G)\| \cdot \|\text{Aut}(H)\| + \|\text{Iso}(G, H)\|^2$, woraus die Behauptung (2.12) mittels (2.11) folgt. \square

Sind G und H isomorphe Graphen und ist τ ein Isomorphismus in $\text{Iso}(G, H)$, dann gilt $\text{Iso}(G, H) = \text{Aut}(G)\tau$. Das heißt, $\text{Iso}(G, H)$ ist eine rechte co-Menge von $\text{Aut}(G)$ in \mathfrak{S}_n . Da je zwei rechte co-Mengen entweder disjunkt oder gleich sind, kann \mathfrak{S}_n in rechte co-Mengen von $\text{Aut}(G)$ gemäß (2.9) zerlegt werden:

$$\mathfrak{S}_n = \text{Aut}(G) \tau_1 \cup \text{Aut}(G) \tau_2 \cup \dots \cup \text{Aut}(G) \tau_k, \quad (2.13)$$

wobei $\|\text{Aut}(G) \tau_i\| = \|\text{Aut}(G)\|$ für jedes i , $1 \leq i \leq k$. Die Menge $\{\tau_1, \tau_2, \dots, \tau_k\}$ von Permutationen in \mathfrak{S}_n ist somit eine vollständige rechte Transversale von $\text{Aut}(G)$ in \mathfrak{S}_n . Bezeichnet man mit $\pi(G)$ den Graphen H , den man erhält, indem man die Permutation $\pi \in \mathfrak{S}_n$ auf die Knoten von G anwendet, womit sich $H \cong G$ ergibt, so folgt:

$$\{\tau_i(G) \mid 1 \leq i \leq k\} = \{H \mid H \cong G\}.$$

Da es genau $n! = n(n-1) \cdots 2 \cdot 1$ Permutationen in \mathfrak{S}_n gibt, folgt

$$\|\{H \mid H \cong G\}\| = k = \frac{\|\mathfrak{S}_n\|}{\|\text{Aut}(G)\|} = \frac{n!}{\|\text{Aut}(G)\|}$$

aus (2.13). Dies beweist die folgende Aussage.

Korollar 2.52. Zu jedem Graphen G mit n Knoten sind $\frac{n!}{\|\text{Aut}(G)\|}$ Graphen isomorph.

Beispielsweise sind genau $5!/4 = 30$ Graphen isomorph zu dem Graphen G aus Abbildung 2.6 in Beispiel 2.50.

Das folgende Lemma wird später in Abschnitt 6.5.1 benötigt. Seien G und H zwei gegebene Graphen mit je n Knoten. Definiere die Menge

$$A(G, H) = \{\langle F, \varphi \rangle \mid F \cong G \wedge \varphi \in \text{Aut}(F)\} \cup \{\langle F, \varphi \rangle \mid F \cong H \wedge \varphi \in \text{Aut}(F)\}.$$

Lemma 2.53. Für je zwei Graphen G und H mit je n Knoten gilt:

$$\|A(G, H)\| = \begin{cases} n! & \text{falls } G \cong H \\ 2n! & \text{falls } G \not\cong H. \end{cases}$$

Beweis. Sind F und G isomorph, so impliziert $\|\text{Aut}(F)\| = \|\text{Aut}(G)\|$, dass

$$\|\{\langle F, \varphi \rangle \mid F \cong G \wedge \varphi \in \text{Aut}(F)\}\| = \frac{n!}{\|\text{Aut}(F)\|} \cdot \|\text{Aut}(F)\| = n!$$

nach Korollar 2.52. Analog gilt $\|\{\langle F, \varphi \rangle \mid F \cong H \wedge \varphi \in \text{Aut}(F)\}\| = n!$.

Sind G und H isomorph, so sind die Mengen $\{\langle F, \varphi \rangle \mid F \cong G \wedge \varphi \in \text{Aut}(F)\}$ und $\{\langle F, \varphi \rangle \mid F \cong H \wedge \varphi \in \text{Aut}(F)\}$ gleich, woraus folgt, dass $\|A(G, H)\| = n!$.

Sind G und H nicht isomorph, so sind die Mengen $\{\langle F, \varphi \rangle \mid F \cong G \wedge \varphi \in \text{Aut}(F)\}$ und $\{\langle F, \varphi \rangle \mid F \cong H \wedge \varphi \in \text{Aut}(F)\}$ disjunkt. Somit gilt $\|A(G, H)\| = 2n!$. \square

2.5 Wahrscheinlichkeitstheorie

Der Zufall ist ein wichtiges Konzept in der Algorithmitik, Komplexitätstheorie und Kryptologie. Um beispielsweise den Begriff der perfekten Geheimhaltung von Kryptosystemen in Kapitel 4 zu definieren und zu diskutieren und um randomisierte Algorithmen und probabilistische Komplexitätsklassen in Kapitel 6 mathematisch korrekt

einzuführen, brauchen wir ein paar elementare Begriffe der Wahrscheinlichkeitstheorie.

Wir beschäftigen uns hier nur mit endlichen Wahrscheinlichkeitsräumen. Diese sind durch eine endliche Menge $\mathcal{E} = \{e_1, e_2, \dots, e_k\}$ von Elementarereignissen spezifiziert, denen jeweils eine Wahrscheinlichkeit $p_i = \Pr(e_i)$ zugeordnet ist, so dass $\sum_{i=1}^k p_i = 1$ gilt. Damit verbindet sich die Vorstellung, dass jedes Elementarereignis e_i einem der möglichen Ausgänge eines Zufallsexperiments entspricht. Beispielsweise kann ein randomisierter Algorithmus Zufallswahlen treffen, die das Resultat der Berechnung beeinflussen.

Die den Elementarereignissen zugeordneten Wahrscheinlichkeiten liefern eine *Wahrscheinlichkeitsverteilung*. Treten alle Ereignisse mit derselben Wahrscheinlichkeit auf (d.h., gilt $p_i = 1/k$ für jedes i mit $1 \leq i \leq k$), so erhält man die *Gleichverteilung*. Die Zuordnung von Wahrscheinlichkeiten kann von den Elementarereignissen e_i auf jede Teilmenge $E \subseteq \mathcal{E}$ erweitert werden, indem man definiert: $\Pr(E) = \sum_{e_i \in E} p_i$. Solch eine Teilmenge E heißt *Ereignis*. Für die Gleichverteilung auf \mathcal{E} ist $\Pr(E) = \|E\|/\|\mathcal{E}\|$ der Anteil der „guten“ Fälle an allen Fällen.

Die folgenden grundlegenden Eigenschaften der Wahrscheinlichkeitsfunktion $\Pr(\cdot)$ sind leicht zu sehen. Für Ereignisse $E, F \subseteq \mathcal{E}$ gilt:

1. $0 \leq \Pr(E) \leq 1$, wobei $\Pr(\emptyset) = 0$ und $\Pr(\mathcal{E}) = 1$.
2. $\Pr(\overline{E}) = 1 - \Pr(E)$, wobei $\overline{E} = \mathcal{E} - E$ das zu E komplementäre Ereignis ist.
3. $\Pr(E \cup F) = \Pr(E) + \Pr(F) - \Pr(E \cap F)$.

Nun werden die Begriffe der bedingten Wahrscheinlichkeit und der (stochastischen) Unabhängigkeit von Ereignissen definiert.

Definition 2.54. Seien A und B Ereignisse mit $\Pr(B) > 0$.

- Die Wahrscheinlichkeit dafür, dass A eintritt, unter der Bedingung, dass B eintritt, ist definiert durch

$$\Pr(A | B) = \frac{\Pr(A \cap B)}{\Pr(B)}.$$

- A und B heißen unabhängig, falls $\Pr(A \cap B) = \Pr(A) \cdot \Pr(B)$; äquivalent, falls $\Pr(A | B) = \Pr(A)$.

Lemma 2.55 (Bayes). Seien A und B Ereignisse mit $\Pr(A) > 0$ und $\Pr(B) > 0$. Dann gilt

$$\Pr(B) \cdot \Pr(A | B) = \Pr(A) \cdot \Pr(B | A).$$

Beweis. Nach Definition gilt

$$\Pr(B) \cdot \Pr(A | B) = \Pr(A \cap B) = \Pr(B \cap A) = \Pr(A) \cdot \Pr(B | A),$$

woraus das Lemma folgt. □

Sind A und B unabhängig, so gilt außerdem:

$$\Pr(A) = \Pr(A | B) \cdot \Pr(B) + \Pr(A | \overline{B}) \cdot (1 - \Pr(B)).$$

Eine *Zufallsvariable* ist eine Funktion, die von \mathcal{E} in \mathbb{R} (oder in \mathbb{Z}) abbildet. Sind beispielsweise die Elementarereignisse e_i die Eingaben für einen randomisierten Algorithmus A , dann könnte man die Zufallsvariable $\mathbf{X}(e_i)$ als die Laufzeit von A bei Eingabe e_i definieren.

Ist $\mathbf{X} : \mathcal{E} \rightarrow \mathbb{R}$ eine Zufallsvariable auf dem Wahrscheinlichkeitsraum \mathcal{E} , so bezeichne „ $\mathbf{X} = x$ “ das Ereignis E , dass \mathbf{X} den Wert $x \in \mathbb{R}$ annimmt, d.h., $E = \{e_i \in \mathcal{E} \mid \mathbf{X}(e_i) = x\}$. Der *Erwartungswert* und die *Varianz* einer Zufallsvariablen \mathbf{X} sind definiert durch:

$$\begin{aligned} E(\mathbf{X}) &= \sum_{e_i \in \mathcal{E}} \mathbf{X}(e_i) \cdot \Pr(e_i); \\ V(\mathbf{X}) &= E((\mathbf{X} - E(\mathbf{X}))^2) = E(\mathbf{X}^2) - (E(\mathbf{X}))^2. \end{aligned}$$

Intuitiv liefert der Erwartungswert $E(\mathbf{X})$ den Wert, den \mathbf{X} bezüglich der zugrunde liegenden Wahrscheinlichkeitsverteilung im Mittel annimmt. Die Varianz ist ein Maß dafür, wie weit die Werte von \mathbf{X} von $E(\mathbf{X})$ abweichen.

Sei p die Wahrscheinlichkeit dafür, dass ein bestimmtes Ereignis E eintritt, und sei \mathbf{X} eine Zufallsvariable, die die Anzahl der unabhängigen Versuche angibt, bis E erstmalig eintritt. Sei μ die Anzahl der unabhängigen Versuche *im Mittel*, bis E erstmalig eintritt, d.h., $\mu = E(\mathbf{X})$. Mit Wahrscheinlichkeit p tritt E bereits beim ersten Versuch ein (welcher somit erfolgreich ist), und mit Wahrscheinlichkeit $1 - p$ benötigt man im Mittel μ weitere Versuche nach dem ersten (nicht erfolgreichen) Versuch. Also ist

$$\mu = p \cdot 1 + (1 - p)(1 + \mu),$$

woraus folgt, dass $E(\mathbf{X}) = \mu = 1/p$. Liefert beispielsweise ein randomisierter Algorithmus ein gewünschtes Resultat mit Wahrscheinlichkeit p , so muss dieser Algorithmus im Mittel $1/p$ Mal (unabhängig) wiederholt werden, bis das gewünschte Resultat erhalten wird.

2.6 Übungen und Probleme

Aufgabe 2.1 Beweise, dass der in Abbildung 2.2 präsentierte erweiterte Euklidische Algorithmus korrekt ist.

Aufgabe 2.2 Beweise Satz 2.2 durch Induktion.

Aufgabe 2.3 Goldener Schnitt: Betrachte ein Rechteck mit den Seitenlängen a und $a + b$, wobei $a \neq 0 \neq b$. Schneidet man aus diesem in geeigneter Weise ein Quadrat der Seitenlänge a heraus, so erhält man ein neues Rechteck mit den Seitenlängen a und b . Die Frage ist, für welche Zahlen a und b das Verhältnis der Seitenlängen dieser beiden Rechtecke gleich ist: $a/b = (a+b)/a$.

- (a) Finde Zahlen $a > 0$ und $b > 0$, die diese Gleichung erfüllen.
- (b) Finde Zahlen $a < 0$ und $b > 0$, die diese Gleichung ebenfalls erfüllen.

Aufgabe 2.4 Zeige, dass die Basis des Logarithmus in der \mathcal{O} -Notation irrelevant ist: Sind $a, b > 1$ zwei verschiedene Basen, so ist $\log_b n = (\log_b a)(\log_a n)$, woraus folgt, dass $\log_b n \in \mathcal{O}(\log_a n)$.

Aufgabe 2.5 Betrachte die Grammatiken G_1 und G_2 aus Beispiel 2.8.

- (a) Zeichne den Syntaxbaum für die Ableitung $S_1 \vdash_{G_1} aS_1b \vdash_{G_1} aaS_1bb \vdash_{G_1} aabb$. Beweise $L(G_1) = \{a^n b^n \mid n \in \mathbb{N}\}$.
- (b) Bestimme die von G_2 erzeugte Sprache $L(G_2)$. Stelle dazu eine Vermutung auf, was $L(G_2)$ sein könnte, und verifiziere diese Vermutung dann durch den formalen Nachweis, dass *genau* die vermuteten Wörter aus dem Startsymbol S_2 abgeleitet werden können.

Aufgabe 2.6 Sei M der DEA auf der linken Seite von Abbildung 2.4, und sei N der NEA auf der rechten Seite von Abbildung 2.4.

- (a) Bestimme die von M akzeptierte Sprache $L(M)$.
- (b) Bestimme die von N akzeptierte Sprache $L(N)$.

Aufgabe 2.7 Beweise, dass genau die von einem NEA akzeptierbaren Sprachen durch eine reguläre Grammatik erzeugt werden können.

Aufgabe 2.8 Beweise Satz 2.13.

Hinweis: Definiere für einen gegebenen NEA M einen DEA, dessen Zustände die Teilmengen der Zustände von M sind, und modifiziere die Überführungsfunktion von M und seine Endzustandsmenge in geeigneter Weise.

Aufgabe 2.9 (a) Gib eine detaillierte, formale Definition des grob skizzierten Begriffs einer Gödelisierung von Turingmaschinen an. Spezifizierte dazu insbesondere das zugrunde liegende Alphabet Σ und erkläre, wie man syntaktisch korrekte Turingmaschinen durch Wörter über Σ codieren kann.

(b) Gib eine detaillierte, formale Definition des Begriffs einer Gödelisierung von deterministischen endlichen Automaten an.

Aufgabe 2.10 Definiere die *disjunktive Normalform* (kurz DNF) einer booleschen Formel dual zur konjunktiven Normalform aus Definition 2.23 durch Vertauschen von \wedge und \vee . Das heißt, eine boolesche Formel φ ist genau dann in DNF, wenn sie eine Disjunktion von Konjunktionen ist:

$$\varphi(x_1, x_2, \dots, x_n) = \bigvee_{i=1}^m \left(\bigwedge_{j=1}^{k_i} \ell_{i,j} \right),$$

wobei die $\ell_{i,j}$ Literale über $\{x_1, x_2, \dots, x_n\}$ sind. Definiere das Erfüllbarkeitsproblem für boolesche Formeln in disjunktiver Normalform durch

$$\text{DNF-SAT} = \{\varphi \mid \varphi \text{ ist eine erfüllbare boolesche Formel in DNF}\}.$$

Entwirf einen deterministischen Algorithmus, der in Polynomialzeit testet, ob eine gegebene Formel in DNF erfüllbar ist oder nicht.

Aufgabe 2.11 Betrachte die in Beispiel 2.25 definierten booleschen Formeln φ und ψ .

- (a) Finde alle erfüllenden Belegungen für φ und ψ .
- (b) Forme φ und ψ in äquivalente Formeln in DNF um. Verwende dazu geeignete Regeln aus Tabelle 2.9.
- (c) Beweise, dass die Formel $\varphi = ((\neg x \wedge \neg y) \implies \neg y)$ eine Tautologie ist. Gib dazu ihre Wahrheitstafel an. Beweise, dass $\neg \varphi$ unerfüllbar ist.

Aufgabe 2.12 (a) Beweise die in Tabelle 2.9 angegebenen Äquivalenzen. Gib dazu die entsprechenden Wahrheitstafeln an.

- (b) Definiere die Semantik von QBFs analog zu Definition 2.24.
- (c) Beweise die Verallgemeinerung der deMorganschen Regel für quantifizierte boolesche Formeln, die in (2.6) angegeben ist.
- (d) Verallgemeinere die Distributivgesetze aus Tabelle 2.9 für quantifizierte boolesche Formeln.

Aufgabe 2.13 (a) Beweise, dass \mathbb{Z} ein Ring bezüglich Addition und Multiplikation ist.

- (b) Ist \mathbb{Z} bezüglich dieser Operationen ein Körper?
- (c) Was kann über die Eigenschaften der algebraischen Strukturen $(\mathbb{N}, +)$, (\mathbb{N}, \cdot) , $(\mathbb{N}, +, \cdot)$, $(\mathbb{Q}, +, \cdot)$ und $(\mathbb{R}, +, \cdot)$ gesagt werden? Hier bezeichnen $+$ und \cdot die gewöhnliche Addition und Multiplikation, \mathbb{Q} ist die Menge der rationalen Zahlen, und \mathbb{R} ist die Menge der reellen Zahlen.

Aufgabe 2.14 (a) Beweise die folgenden Eigenschaften der Euler-Funktion φ :

- $\varphi(m \cdot n) = \varphi(m) \cdot \varphi(n)$ für alle $m, n \in \mathbb{N}$ mit $\text{ggT}(m, n) = 1$, und
- $\varphi(p) = p - 1$ für jede Primzahl p .

(b) Beweise mittels dieser Eigenschaften Fakt 2.37.

Aufgabe 2.15 (a) Wie viele Primitivwurzeln haben \mathbb{Z}_{13}^* und \mathbb{Z}_{14}^* ?

- (b) Welche der beiden Ringe \mathbb{Z}_{13} und \mathbb{Z}_{14} (falls überhaupt einer) sind Körper? Finde alle Primitivwurzeln von \mathbb{Z}_{13}^* und \mathbb{Z}_{14}^* und beweise, dass diese tatsächlich Primitivwurzeln sind.
- (c) Beweise für jede Primitivwurzel von 13 bzw. 14, dass sie ganz \mathbb{Z}_{13}^* bzw. \mathbb{Z}_{14}^* erzeugt.

Aufgabe 2.16 Betrachte die Graphen F , G und H aus Abbildung 2.6 in Beispiel 2.50.

- (a) Bestimme alle Isomorphismen zwischen G und H .
- (b) Bestimme alle Automorphismen von F , G und H .
- (c) Für welche Isomorphismen zwischen G und H ist $\text{Iso}(G, H)$ eine rechte co-Menge von $\text{Aut}(G)$ in \mathfrak{S}_5 , d.h., für welche $\tau \in \text{Iso}(G, H)$ ist $\text{Iso}(G, H)$ gleich $\text{Aut}(G)\tau$?

Bestimme die vollständigen rechten Transversalen von $\text{Aut}(F)$, $\text{Aut}(G)$ und $\text{Aut}(H)$ in \mathfrak{S}_5 .

- (d) Bestimme den Orbit aller Knoten von F in $\text{Aut}(F)$ und den Orbit aller Knoten von H in $\text{Aut}(H)$.
- (e) Bestimme den Turm der Stabilisatoren für die Untergruppen $\text{Aut}(F) \leq \mathfrak{S}_5$ und $\text{Aut}(H) \leq \mathfrak{S}_5$.
- (f) Wie viele Graphen mit fünf Knoten sind isomorph zu F ?

Aufgabe 2.17 Der *co-Graph* \overline{G} eines gegebenen Graphen G ist durch

- die Knotenmenge $V(\overline{G}) = V(G)$ und
- die Kantenmenge $E(\overline{G}) = \{\{i, j\} \mid i, j \in V(\overline{G}) \text{ und } \{i, j\} \notin E(G)\}$

definiert. Beweise die folgenden Aussagen:

- (a) $\text{Aut}(G) = \text{Aut}(\overline{G})$.
- (b) $\text{Iso}(G, H) = \text{Iso}(\overline{G}, \overline{H})$.
- (c) \overline{G} ist zusammenhängend, falls G nicht zusammenhängend ist. Gilt auch die Umkehrung?
-

Problem 2.1 (Arithmetik in \mathbb{Z}_k)

Seien $k \in \mathbb{N}$ und $x, y, z \in \mathbb{Z}$. Die Zahl x ist *kongruent zu y modulo k* (kurz $x \equiv y \pmod k$), falls k die Differenz $y - x$ teilt. Zum Beispiel ist $-3 \equiv 16 \pmod{19}$ und $8 \equiv 0 \pmod{2}$. Die Kongruenz \equiv modulo k definiert eine *Äquivalenzrelation* auf \mathbb{Z} , d.h., sie ist *reflexiv* ($x \equiv x \pmod k$), *symmetrisch* ($x \equiv y \pmod k$ impliziert $y \equiv x \pmod k$) und *transitiv* (aus $x \equiv y \pmod k$ und $y \equiv z \pmod k$ folgt $x \equiv z \pmod k$).

Die Menge $x + k\mathbb{Z} = \{y \in \mathbb{Z} \mid y \equiv x \pmod k\}$ heißt die *Restklasse von x mod k* . Zum Beispiel ist die Restklasse von $3 \pmod 7$ die Menge

$$3 + 7\mathbb{Z} = \{3, 3 \pm 7, 3 \pm 2 \cdot 7, \dots\} = \{3, 10, -4, 17, -11, \dots\}.$$

Wir wählen stets die kleinste natürliche Zahl in $x + k\mathbb{Z}$, um die Restklasse von x mod k zu *repräsentieren*; so repräsentiert etwa 3 die Klasse $3 \pmod 7$. Die Menge aller Restklassen modulo k ist $\mathbb{Z}_k = \{0, 1, \dots, k-1\}$. Auf \mathbb{Z}_k definieren wir die *Addition modulo k* durch $(x + k\mathbb{Z}) + (y + k\mathbb{Z}) = (x + y) + k\mathbb{Z}$ und die *Multiplikation modulo k* durch $(x + k\mathbb{Z}) \cdot (y + k\mathbb{Z}) = (x \cdot y) + k\mathbb{Z}$. In der Arithmetik modulo 7 gilt dann zum Beispiel $(3 + 7\mathbb{Z}) + (6 + 7\mathbb{Z}) = (3 + 6) + 7\mathbb{Z} = 2 + 7\mathbb{Z}$ und $(3 + 7\mathbb{Z}) \cdot (4 + 7\mathbb{Z}) = (3 \cdot 4) + 7\mathbb{Z} = 5 + 7\mathbb{Z}$.

Beweise, dass in der Arithmetik modulo k gilt:

- (a) $(\mathbb{Z}_k, +, \cdot)$ ist ein kommutativer Ring mit Eins.
- (b) Die in Beispiel 2.35 definierte Menge \mathbb{Z}_k^* ist eine multiplikative Gruppe.
- (c) Für jede Primzahl p ist $(\mathbb{Z}_p, +, \cdot)$ ein Körper.
- (d) Beweise, dass in jeder Gruppe das neutrale Element und die inversen Elemente stets eindeutig sind.

- (e) Beweise, dass die invertierbaren Elemente in einem kommutativen Ring \mathfrak{R} mit Eins eine Gruppe bilden, welche die *Einheitengruppe von \mathfrak{R}* genannt wird. Was ist die Einheitengruppe des Ringes \mathbb{Z}_k ?

Problem 2.2 (Pumping-Lemma)

Es gibt zwei Versionen des Pumping-Lemmas, die jeweils als ein Werkzeug verwendet werden können, um zu zeigen, dass bestimmte Sprachen nicht regulär bzw. nicht kontextfrei sind.

Lemma 2.56 (Pumping-Lemma für reguläre Sprachen). *Sei L eine Sprache in REG. Dann gibt es eine Zahl $n \geq 1$ (die von L abhängt), so dass jedes Wort $x \in L$ mit $|x| \geq n$ in der Form $x = uvw$ geschrieben werden kann, wobei $|v| \geq 1$, $|uv| \leq n$, und für jedes $i \geq 0$ gilt $uv^i w \in L$.*

Zeige unter Verwendung von Lemma 2.56, dass die in Beispiel 2.8 definierte kontextfreie Sprache $L = \{a^n b^n \mid n \in \mathbb{N}\}$ nicht regulär ist.

Lemma 2.57 (Pumping-Lemma für kontextfreie Sprachen). *Sei L eine Sprache in CF. Dann gibt es eine Zahl $n \geq 1$ (die von L abhängt), so dass jedes Wort $z \in L$ mit $|z| \geq n$ in der Form $z = uvwxy$ geschrieben werden kann, wobei $|vx| \geq 1$, $|vwx| \leq n$, und für jedes $i \geq 0$ gilt $uv^i wx^i y \in L$.*

Zeige unter Verwendung von Lemma 2.57, dass die in Beispiel 2.18 definierte kontextsensitive Sprache $L = \{a^n b^n c^n \mid n \geq 1\}$ nicht kontextfrei ist.

Problem 2.3 (Baumisomorphie)

Das Graphisomorphie-Problem GI ist auf bestimmten speziellen Graphklassen effizient lösbar, etwa auf der Klasse der Bäume. Ein (*ungerichteter*) Baum ist ein zusammenhängender, kreisfreier Graph, wobei ein *Kreis* eine Folge aufeinander folgender Kanten ist, die zum Ausgangspunkt zurückkehrt. Die *Blätter eines Baums* sind die Knoten vom Grad eins. Entwirf einen effizienten Algorithmus für das *Baumisomorphie-Problem*, das definiert ist durch

$$\text{TI} = \{\langle G, H \rangle \mid G \text{ und } H \text{ sind isomorphe Bäume}\}.$$

Hinweis: Markiere die Knoten der gegebenen zwei Bäume sukzessive durch geeignete Folgen von Zahlen und vergleiche die resultierenden Markierungsfolgen in den einzelnen Schleifen des Algorithmus. Arbeitet dich dabei von den Blättern beginnend zum Zentrum der Bäume vor. Der Algorithmus hält, wenn alle Knoten markiert sind; siehe auch [KST93].

2.7 Zusammenfassung und bibliographische Notizen

Allgemeine Bemerkungen: Als Vorbereitung für die nachfolgenden Kapitel wurden in diesem Kapitel die elementaren Grundlagen einiger Gebiete aus der Informatik

und Mathematik eingeführt. Für jedes dieser Gebiete gibt es viele gute Bücher, von denen einige hier erwähnt werden.

Der Klassiker für eine Einführung in die Algorithmik ist das Buch von Cormen, Leiserson, Rivest und Stein [CLRS01]. Ebenfalls sehr empfehlenswert sind die Bücher von Schöning [Sch01] und von Ottmann und Widmayer [OW02]. Zu den Klassikern der Theorie der formalen Sprachen und Automaten gehören die Bücher von Hopcroft, Motwani und Ullman [HU79, HMU01] und von Salomaa [Sal73]. Eng mit diesen Gebieten verwandt ist die Berechenbarkeitstheorie, deren Grundlagen in den frühen Arbeiten von Turing [Tur36] und anderen gelegt wurden. Zu den Klassikern hier sind die Bücher von Kleene [Kle52], H. Rogers [Rog67], Odifreddi [Odi89] und Homer und Selman [HS01] zu zählen. Die in diesem Absatz erwähnten Gebiete wurden in den Abschnitten 2.1 und 2.2 behandelt; sie bilden das Fundament der Theoretischen Informatik.

In den Abschnitten 2.3, 2.4 und 2.5 wurden verschiedene mathematische Gebiete kurz eingeführt. Wenigstens eine Referenz pro Gebiet soll nun gegeben werden. Shoenfield [Sho67] präsentiert die Grundlagen der Logik aus mathematischer Perspektive. Aus Sicht der Informatik sind auch das sehr gut lesbare Buch von Schöning [Sch95a] und Teil II des Buches von Papadimitriou [Pap95] empfehlenswert. Eine Einführung in die Zahlentheorie findet man in den Büchern von Hardy und Wright [HW79] und Rosen [Ros99]. Für die Graphentheorie sind beispielsweise die Bücher von Harary [Har69], Golumbic [Gol80] und Brandstädt et al. [BLS99] zu empfehlen. Mehr Hintergrund zur Algebra findet man in den Büchern von Anton [Ant00] und Jacobson [Jac74], und als eine Einführung in die Wahrscheinlichkeitstheorie ist etwa das Buch von Feller [Fel68] zu nennen.

Spezielle Bemerkungen: Die Idee nichtdeterministischer Maschinen geht auf Rabin und Scott zurück, die 1976 den Turing Award für ihre Gemeinschaftsarbeit [RS59] erhielten, in der der Begriff des nichtdeterministischen endlichen Automaten eingeführt wurde. Nichtdeterminismus gilt heute als ein ausgesprochen nützliches und wertvolles Konzept, und die klassische Arbeit von Rabin und Scott hat sich als eine stetige Quelle der Inspiration für nachfolgende Arbeiten in diesem Gebiet erwiesen. Ein besonders wichtiger Zweig dieser Forschung untersucht die Klasse NP und hat insbesondere die Theorie der NP-Vollständigkeit hervorgebracht, die in Abschnitt 3.5.3 genauer vorgestellt wird.

Grundlagen der Komplexitätstheorie

3.1 Aufgaben und Ziele der Komplexitätstheorie

Die erste und wichtigste Aufgabe der Komplexitätstheorie ist es, die Berechnungskomplexität von Problemen (oder, synonym dazu, ihre „Härte“) so genau wie möglich zu bestimmen. Betrachte die durch $S = \{x2^{|x|}x \mid x \in \{0,1\}^*\}$ definierte Menge, deren Elemente Wörter über dem Alphabet $\{0,1,2\}$ sind. Wie „hart“ ist S ? Wie schwer ist es also, algorithmisch zu entscheiden, ob ein gegebenes Eingabewort zu S gehört oder nicht? Die erste Antwort ist: Nun, es kommt darauf an!

Erstens muss man den Begriff „Algorithmus“ formal spezifizieren; zweitens muss man formal spezifizieren, welche Art von „Komplexität“ zu messen ist und in welcher Weise. In diesem Buch werden *Algorithmen* durchweg von *Turingmaschinen* repräsentiert, einem einfachen mathematischen Computermodell, das formal in Abschnitt 2.2 beschrieben ist. Ein algorithmisches Gerät wie eine Turingmaschine kann mit ganz unterschiedlichen technischen Fähigkeiten ausgestattet sein, die seine Berechnungskraft und -effizienz beeinflussen. Tatsächlich markiert die Beobachtung, dass verschiedene Algorithmen für ein und dasselbe Problem verschiedene Laufzeiten und einen unterschiedlichen Speicherplatzbedarf haben können, den Beginn der Komplexitätstheorie. Daher ist die zweite und spezifischere Antwort auf die obige Frage eigentlich mindestens vier Antworten:

1. Turingmaschinen mit einem Eingabeband (von dem nur gelesen werden kann) und einem Arbeitsband (auf dem man lesen und schreiben kann) können S in Echtzeit lösen, d.h., die Anzahl der Schritte in der Berechnung ist gleich der Länge der Eingabe; siehe auch Problem 3.1 am Ende dieses Kapitels.
2. Turingmaschinen ohne separates Eingabeband und mit nur einem Arbeitsband (das gleichzeitig als Eingabeband dient) erfordern einen mindestens quadratischen Zeitaufwand in der Eingabegröße, um S zu lösen; siehe auch Problem 3.1.
3. Alternierende Turingmaschinen (welche in Abschnitt 5.6 eingeführt werden) benötigen nicht mehr als logarithmische Zeit in der Eingabegröße, um S zu lösen.
4. Endliche Automaten können S überhaupt nicht lösen. Anzumerken ist hier, dass endliche Automaten als sehr stark eingeschränkte Turingmaschinen aufgefasst

werden können, die mit einem Einweg-Lese-Eingabeband ausgestattet sind (d.h., der Lesekopf darf nur von links nach rechts gehen und muss in jedem Takt genau ein Feld weiter wandern), die über kein Arbeitsband verfügen und die ihre Arbeit in Echtzeit beenden müssen.

Diese Beobachtung ist uns aus dem alltäglichen Leben geläufig. Steht man etwa dem Problem gegenüber, einen Baum von zwei Metern Durchmesser zu fällen, so macht es durchaus einen Unterschied, mit welchen Werkzeugen oder Geräten man ausgerüstet ist, um das Problem zu lösen. Ein Arbeiter, der nur über eine Axt verfügt, wird sicherlich viel mehr Zeit für diese Aufgabe brauchen als ein Arbeiter, der eine Kettensäge benutzt. Ein Arbeiter, der sich mit einem ungeeigneten Werkzeug – etwa einer Nagelfeile – an die Lösung der Aufgabe wagt, braucht wahrscheinlich ewig, oder vielleicht gibt er auch einfach frustriert auf. Also kann man einer gegebenen Aufgabe oder einem Problem eine bestimmte Komplexität nur in Abhängigkeit von den verwendeten Geräten zuweisen. Für ein anderes Problem allerdings kann sich die Kettensäge, die auf den ersten Blick im Vergleich mit der Axt als das mächtigere Werkzeug erscheint, als eigentlich unpassender und somit nicht so wirkungsvoll oder weniger effizient erweisen. Steht man zum Beispiel dem Problem gegenüber, fünfzig Holzklötze in handliche Kaminscheite zu spalten, so sollte man die Axt nehmen, nicht die Kettensäge, und man wird damit viel schneller fertig sein.

Die nächste Frage ist: Welche Art von Komplexität möchte man messen? Beim Beispiel des Baumes könnte man etwa die Zeit messen, die nötig ist, ihn zu fällen, und diesen Wert für die Komplexität des Problems nehmen. Man könnte auch die physikalische Kraft oder Energie messen, die man zur Erledigung dieser Aufgabe braucht, wodurch man einen anderen Wert für die Komplexität desselben Problems erhalten würde. Das heißt, man würde dann eine andere Art von Komplexität messen. Diese Beobachtungen legen nahe, dass die *Berechnungskomplexität* eines Problems durch die folgenden drei Merkmale bestimmt wird:

- das verwendete *Berechnungsmodell* oder *algorithmische Gerät* – z.B. die in Abschnitt 2.2 definierte Zweiwege-Mehrband-Turingmaschine;
- das *Berechnungsparadigma* oder der *Akzeptierungsmodus* dieses Berechnungsmodells – z.B. die deterministische oder nichtdeterministische oder probabilistische oder alternierende oder nichtmehrdeutige usw. Turingmaschine;
- das verwendete *Komplexitätsmaß* oder die *Ressource* – z.B. die zur Lösung des Problems nötige Zeit (d.h. die Anzahl der bei der Berechnung ausgeführten Schritte) oder der Raum (d.h. die Anzahl der bei der Berechnung benutzten Bandzellen) usw.

Offen gesagt ist die im ersten Absatz dieses Kapitels erwähnte Menge S kein welterschütternd interessantes Problem. Komplexitätstheorie untersucht wichtige, interessante, natürliche Probleme aus nahezu jedem Wissenschaftsgebiet, einschließlich so unterschiedlicher Gebiete wie Logik, Graphentheorie, Algebra und Zahlentheorie, Algorithmik, Kryptographie, Codierungs- und Informationstheorie, Datenkompression, formale Sprachen und Automaten, Schaltkreistheorie, Bioinformatik, Spieltheorie, Wirtschafts- und Politikwissenschaften (speziell Social-Choice-

Theorie), Künstliche Intelligenz, Quantenmechanik und viele weitere. Solche Probleme hinsichtlich ihrer Berechnungskomplexität zu klassifizieren, ist eine der Hauptaufgaben der Komplexitätstheorie.

Eine andere wichtige Aufgabe der Komplexitätstheorie besteht darin, die Berechnungskraft verschiedener algorithmischer Geräte und Automaten und der zugehörigen Berechnungsparadigma miteinander zu vergleichen und die „*Trade-offs*“ unterschiedlicher Komplexitätsmaße zu bestimmen. Beispielsweise werden die Zeit-versus-Raum-Frage und die Determinismus-versus-Nichtdeterminismus-Frage in den Abschnitten 3.2, 3.3 und 3.4 behandelt. Insbesondere werden in Abschnitt 3.2 die Komplexitätsmaße Zeit und Raum im Worst-Case-Komplexitätsmodell definiert und die entsprechenden Komplexitätsklassen eingeführt. Eine Komplexitätsklasse ist eine Menge von Problemen, die gemäß einem gegebenen Berechnungsmodell und -paradigma durch Algorithmen gelöst werden können, welche höchstens einen vorgegebenen Betrag der jeweiligen Komplexitätsressource verbrauchen. In Abschnitt 3.3 werden die Sätze über die lineare Beschleunigung und Raumkompression und die Hierarchiesätze für Zeit und Raum bewiesen. Diese Resultate beantworten die Frage danach, um wie viel eine Komplexitätsressource vergrößert werden muss, damit echt mehr Probleme von einem bezüglich dieser Ressource beschränkten Algorithmus gelöst werden können. Abschnitt 3.4 erkundet dann den Bereich zwischen logarithmischem und polynomialem Raum.

Wie oben erwähnt kann man mittels Komplexitätsklassen die Komplexität eines Problems beschreiben, oder *klassifizieren*, indem man eine der Klasse entsprechende obere und eine passende untere Schranke für das Problem angibt. Für eine Komplexitätsklasse \mathcal{C} und ein Problem A erfordert der Beweis, dass \mathcal{C} eine *obere Schranke* für A ist, „lediglich“ den Entwurf eines Algorithmus, der das Problem löst und dafür höchstens so viel der betreffenden Komplexitätsressource braucht, wie \mathcal{C} bereitstellt. Das heißt, ein geeigneter Algorithmus – und nur einer – ist genug, um eine obere Schranke zu liefern. Hingegen ist es gewöhnlich viel schwerer, zu beweisen, dass \mathcal{C} eine *untere Schranke* für A ist. Man muss dann zeigen, dass *jeder* Algorithmus zur Lösung des Problems notwendigerweise *mindestens* so viel der betreffenden Komplexitätsressource braucht, wie \mathcal{C} bereitstellt. Anders gesagt muss man argumentieren, dass *kein Algorithmus*, egal wie raffiniert er ist, das Problem mit weniger Ressourcen als denen von \mathcal{C} lösen kann.

Um die Berechnungskomplexität zweier Probleme zu vergleichen, werden in Abschnitt 3.5 *komplexitätsbeschränkte Reduzierbarkeiten* eingeführt. Ein Problem A auf ein Problem B zu reduzieren bedeutet, dass A höchstens so hart wie B ist. Insbesondere werden die polynomialzeit-beschränkte und die durch logarithmischen Raum beschränkte Many-one-Reduzierbarkeit untersucht. Im Zusammenhang damit werden die Begriffe der *Härte* und der *Vollständigkeit* eines Problems für eine Komplexitätsklasse eingeführt. Die vollständigen Probleme in einer Komplexitätsklasse sind bezüglich der betreffenden Reduzierbarkeit die härtesten Probleme dieser Klasse. Das heißt, *jedes* Problem in der Klasse kann auf jedes für die Klasse vollständige Problem reduziert werden. In diesem Sinn repräsentiert ein jedes \mathcal{C} -vollständige Problem die ganze Klasse \mathcal{C} . Allerdings sind nicht in allen Klassen vollständige Pro-

bleme bekannt. Beispielsweise haben so genannte „*Promise*“-Klassen höchstwahrscheinlich keine vollständigen Probleme; siehe die Abschnitte 6.2.1 und 6.5.2.

Speziell werden die wichtigen Begriffe der NL-*Vollständigkeit* und NP-*Vollständigkeit* untersucht. Zu den Problemen, die vollständig in NL bzw. NP sind, gehören Variationen des Erfüllbarkeitsproblems für boolesche Formeln und bestimmte Graphenprobleme. Ein Beispiel eines NL-vollständigen Problems ist das Graph-Erreichbarkeitsproblem: Gegeben ein gerichteter Graph G und zwei ausgezeichnete Knoten s und t von G , gibt es einen Pfad in G von s zu t ? Ein anderes Beispiel eines NL-vollständigen Problems ist 2-SAT: Gegeben eine boolesche Formel φ mit zwei Literalen pro Klausel, gibt es eine Belegung der Variablen von φ mit Wahrheitswerten, unter der φ wahr ist? Im Gegensatz dazu ist das allgemeine Erfüllbarkeitsproblem (ohne Einschränkung der Anzahl der Literale pro Klausel) ein typisches NP-vollständiges Problem gemäß dem Satz von Cook. Sein Resultat zeigt, dass die Berechnung einer beliebigen gegebenen NP-Maschine bei irgendeiner Eingabe als eine boolesche Formel codiert werden kann, die genau dann erfüllbar ist, wenn die Berechnung ihre Eingabe akzeptiert. Dieses Ergebnis stellt einen wirklichen Durchbruch dar, denn es gibt uns das erste natürliche NP-vollständige Problem in die Hand. Mittels geeigneter Reduktionen vom Erfüllbarkeitsproblem kann dann die NP-*Vollständigkeit* vieler anderer Probleme gezeigt werden, zum Beispiel des Problems 3-SAT (das Erfüllbarkeitsproblem, eingeschränkt auf Formeln mit drei Literalen pro Klausel), des Dreifärbbarkeitsproblems für Graphen und Tausender weiterer Probleme.

Abschnitt 3.6 studiert die wichtigen Klassen P und NP, deterministische und nichtdeterministische polynomiale Zeit. Für die P-versus-NP-Frage, eine der wichtigsten offenen Fragen in der theoretischen Informatik, ist der Begriff der NP-*Vollständigkeit* besonders nützlich. Es gibt jedoch auch Teilklassen von NP, die höchstwahrscheinlich keine vollständigen Probleme haben. Beispielsweise ist UP („*Unambiguous Polynomial Time*“) die Klasse der NP-Probleme, deren Instanzen entweder gar keine Lösung oder aber eine eindeutige Lösung haben. Auch ist das Graphisomorphie-Problem heute einer der prominentesten Kandidaten für ein Problem in NP, das weder in P liegt noch NP-vollständig ist. Denn einerseits konnte bisher, trotz beträchtlicher Anstrengungen in der Vergangenheit, noch kein effizienter Algorithmus für dieses Problem gefunden werden. Und andererseits ist es sehr unwahrscheinlich, dass dieses Problem NP-vollständig ist, aus Gründen, die später in Abschnitt 5.7 erklärt werden.

3.2 Komplexitätsmaße und -klassen

In diesem Abschnitt werden die Komplexitätsmaße Zeit und Raum im Worst-Case-Komplexitätsmodell eingeführt. Intuitiv ist die *Zeitkomplexität* einer Berechnung die Anzahl der Schritte eines Algorithmus bei einer gegebenen Eingabe, und die *Raumkomplexität* einer Berechnung ist der von einem Algorithmus bei einer Eingabe benötigte Speicherplatz. Die Komplexitätsfunktionen für Zeit und Raum hängen von der Eingabegröße ab. *Worst-Case-Komplexität* bedeutet, dass für einen Algorithmus M und für jede Eingabegröße n als Wert der Komplexitätsfunktion von M bei

Länge n die *maximale* Komplexität der Berechnungen von M über alle Eingaben der Länge n genommen wird.

Turingmaschinen, die unser Algorithmenmodell darstellen, und die Begriffe Konfiguration und Berechnung einer Turingmaschine wurden formal in Abschnitt 2.2 definiert. Sei M eine Turingmaschine mit dem Eingabealphabet $\Sigma = \{0, 1\}$, und sei $x \in \Sigma^*$ ein Eingabewort. $M(x)$ bezeichnet die *Berechnung von M bei Eingabe x* . Ist M eine deterministische Turingmaschine (kurz DTM), dann ist ihre Berechnung einfach eine Folge von Konfigurationen. Ist M eine nichtdeterministische Turingmaschine (kurz NTM), dann ist ihre Berechnung ein Baum, dessen Knoten die Konfigurationen, dessen Wurzel die Startkonfiguration und dessen Blätter die Endkonfigurationen sind. Es ist vernünftig, zu verlangen, dass eine Berechnung genau dann eine Komplexität haben möge, wenn sie in endlich vielen Schritten terminiert. Für den Fall einer nichtdeterministischen Turingmaschine genügt es dabei, dass mindestens ein Pfad im Berechnungsbaum terminiert. Die *Sprache von M* , mit $L(M)$ bezeichnet, ist die Menge aller Eingaben x , die M akzeptiert, d.h., für welche $M(x)$ mindestens einen akzeptierenden Berechnungspfad hat. (DTMs haben niemals mehr als einen Berechnungspfad.)

Definition 3.1 (Deterministische Komplexitätsmaße Zeit und Raum). Sei M eine DTM mit $L(M) \subseteq \Sigma^*$, und sei $x \in \Sigma^*$ eine Eingabe. Definiere für die Berechnung $M(x)$ die Zeitfunktion und die Raumfunktion, bezeichnet mit Time_M bzw. Space_M , die beide von Σ^* in \mathbb{N} abbilden, wie folgt:

$$\begin{aligned} \text{Time}_M(x) &= \begin{cases} m & \text{falls } M(x) \text{ genau } m+1 \text{ Konfigurationen hat} \\ \text{undefiniert} & \text{sonst;} \end{cases} \\ \text{Space}_M(x) &= \begin{cases} \text{Anzahl der Bandzellen in einer größten Konfiguration von } M(x) & \text{falls } M(x) \text{ terminiert} \\ \text{undefiniert} & \text{sonst.} \end{cases} \end{aligned}$$

Definiere die Funktionen $\text{time}_M : \mathbb{N} \rightarrow \mathbb{N}$ und $\text{space}_M : \mathbb{N} \rightarrow \mathbb{N}$ durch:

$$\begin{aligned} \text{time}_M(n) &= \begin{cases} \max_{x:|x|=n} \text{Time}_M(x) & \text{falls } \text{Time}_M(x) \text{ für jedes } x \\ & \text{mit } |x|=n \text{ definiert ist} \\ \text{undefiniert} & \text{sonst;} \end{cases} \\ \text{space}_M(n) &= \begin{cases} \max_{x:|x|=n} \text{Space}_M(x) & \text{falls } \text{Space}_M(x) \text{ für jedes } x \\ & \text{mit } |x|=n \text{ definiert ist} \\ \text{undefiniert} & \text{sonst.} \end{cases} \end{aligned}$$

Dass die Zeitkomplexitätsfunktion immer dann undefiniert sein sollte, wenn die Berechnung nicht terminiert, ist unmittelbar klar. Dass die Raumkomplexitätsfunktion in diesem Fall ebenso undefiniert sein sollte, erfordert jedoch vielleicht etwas mehr Überlegung. Schließlich kann eine nicht terminierende, unendliche Berechnung durchaus auf einem beschränkten Raum stattfinden. Beispielsweise braucht eine Turingmaschine, die ihre Köpfe nie bewegt, lediglich eine Bandzelle, doch ihre Berechnung muss nicht terminieren. Die Frage, ob einer solchen unendlichen Berechnung auf beschränktem Raum dennoch eine spezifische Raumkomplexität zugeschrieben werden sollte, ist in der Tat eine philosophische (oder axiomatische) Sache.

In diesem Buch wollen wir uns auf den Standpunkt stellen, dass eine jede Berechnung genau dann eine Komplexität haben möge, wenn sie ein Resultat liefert, also genau dann, wenn sie terminiert. Und wirklich ist diese Annahme gerade die erste von zwei Bedingungen, mit denen ein abstrakter Begriff eines Komplexitätsmaßes axiomatisch begründet werden kann. Die zweite Bedingung sagt, dass es einen Algorithmus gibt, der für einen gegebenen Algorithmus M , eine Eingabe $x \in \Sigma^*$ und einen Wert $m \in \mathbb{N}$ entscheidet, ob die Berechnung von $M(x)$ genau die Komplexität m hat. Diese beiden Bedingungen sind als die Blumschen Axiome bekannt. Die Zeit- und Raumfunktionen aus Definition 3.1 sind Komplexitätsmaße im Sinne von Blum; siehe Übung 3.3. In diesem Buch werden durchweg nur diese beiden Komplexitätsmaße betrachtet. Es gibt jedoch auch viele andere Komplexitätsmaße, die die Blumschen Axiome erfüllen, darunter auch einige ziemlich pathologische.

Axiome von Blum

Einige elementare Begriffe aus der Berechenbarkeitstheorie wurden in Abschnitt 2.2 eingeführt, insbesondere die Klasse \mathbb{P} der partiell-rekursiven (d.h. berechenbaren) Funktionen und die Klasse \mathbb{R} der totalen berechenbaren Funktionen, siehe Definition 2.17. Der Definitionsbereich einer Funktion f wird mit D_f bezeichnet.

Sei $\varphi_0, \varphi_1, \varphi_2, \dots$ eine feste Gödelisierung (d.h. eine effektive Nummerierung) aller einstelligen Funktionen in \mathbb{P} . Sei $\Phi \in \mathbb{P}$ eine Funktion, die von $\mathbb{N} \times \Sigma^*$ in \mathbb{N} abbildet, und wir schreiben $\Phi_i(x)$ als eine Abkürzung von $\Phi(i, x)$. Dann ist Φ genau dann ein *Blumsches Komplexitätsmaß*, wenn die folgenden beiden Axiome erfüllt sind:

Axiom 1: Für jedes $i \in \mathbb{N}$ ist $D_{\Phi_i} = D_{\varphi_i}$.

Axiom 2: Die Menge $\{(i, x, m) \mid \Phi_i(x) = m\}$ ist entscheidbar.

Deterministische Zeit- und Raumklassen

Definition 3.2 (Deterministische Zeit- und Raumklassen). Seien t und s Funktionen in \mathbb{R} , die von \mathbb{N} in \mathbb{N} abbilden. Definiere die folgenden deterministischen Komplexitätsklassen mit der Ressourcenfunktion t bzw. s :

$$\begin{aligned} \text{DTIME}(t) &= \left\{ A \left| \begin{array}{l} A = L(M) \text{ für eine DTM } M \text{ und für} \\ \text{jedes } n \in \mathbb{N} \text{ gilt } \text{time}_M(n) \leq t(n) \end{array} \right. \right\}; \\ \text{DSPACE}(s) &= \left\{ A \left| \begin{array}{l} A = L(M) \text{ für eine DTM } M \text{ und für} \\ \text{jedes } n \in \mathbb{N} \text{ gilt } \text{space}_M(n) \leq s(n) \end{array} \right. \right\}. \end{aligned}$$

Eine deterministische Turingmaschine M entscheidet ihre Sprache. Falls $A = L(M)$ gilt, so sind sowohl $\text{time}_M(n)$ als auch $\text{space}_M(n)$ für jedes $n \in \mathbb{N}$ definiert. Im Gegensatz dazu akzeptiert eine nichtdeterministische Turingmaschine ihre Sprache. Daher wird der nichtdeterministische Fall etwas anders in den Definitionen 3.3 und 3.4 unten behandelt.

Die Ressourcenfunktionen t und s in Definition 3.1 heißen die *Namen* von $\text{DTIME}(t)$ bzw. $\text{DSPACE}(s)$. Ist M eine Turingmaschine mit mehr als einem Band,

dann ist $\text{Space}_M(x)$, die Größe einer größten Konfiguration von $M(x)$, definiert als die maximale Anzahl der Bandzellen, wobei das Maximum über alle Konfigurationen bezüglich aller Bänder in der Berechnung genommen wird. Wenn es ein separates Eingabeband gibt, von dem nur gelesen werden kann, so wird nur der auf den Arbeitsbändern gebrauchte Raum berücksichtigt. Diese Annahme ist sinnvoll, da man ja auch sublineare Raumfunktionen wie etwa logarithmischen Raum betrachten möchte und die Raumschranke $\log n$ trivialerweise von der Anzahl n der Eingabesymbole auf dem Eingabeband überschritten wird. Für Zeitklassen dagegen ist es vernünftig, nur Ressourcenfunktionen $t \geq \text{id}$ zu betrachten, wobei $\text{id}(n) = n$ die Identitätsfunktion bezeichnet, denn in weniger als n Schritten könnte man nicht einmal die n Eingabesymbole lesen. Eine Ausnahme stellen hier die „alternierenden Turingmaschinen“ dar, die in Abschnitt 5.6 eingeführt werden. Für diese ist es nämlich doch sinnvoll, „logarithmische Zeit“ zu betrachten, wie wir später sehen werden.

Nichtdeterministische Zeit- und Raumklassen

Um nun die nichtdeterministischen Komplexitätsmaße und -klassen zu definieren, sei $M(x)$ der Berechnungsbaum einer nichtdeterministischen Turingmaschine M bei Eingabe x . Die Berechnung entlang eines festen Pfades α in $M(x)$ ist nichts anderes als eine deterministische Berechnung: eine Folge von Konfigurationen. Definiere die Zeit- und Raumfunktionen für jeden Pfad α in $M(x)$ analog zur Definition 3.1 und bezeichne sie mit $\text{Time}_M(x, \alpha)$ bzw. $\text{Space}_M(x, \alpha)$.

Definition 3.3 (Nichtdeterministische Komplexitätsmaße Zeit und Raum). Sei M eine NTM mit $L(M) \subseteq \Sigma^*$, sei $x \in \Sigma^*$ eine Eingabe, und bezeichne α einen Pfad in $M(x)$. Definiere für die Berechnung $M(x)$ die Zeitfunktion und die Raumfunktion, bezeichnet mit NTime_M bzw. NSpace_M , die beide von Σ^* in \mathbb{N} abbilden, wie folgt:

$$\begin{aligned}\text{NTime}_M(x) &= \begin{cases} \min\{\text{Time}_M(x, \alpha) \mid M(x) \text{ akzeptiert auf } \alpha\} & \text{falls } x \in L(M) \\ \text{undefiniert} & \text{sonst;} \end{cases} \\ \text{NSpace}_M(x) &= \begin{cases} \min\{\text{Space}_M(x, \alpha) \mid M(x) \text{ akzeptiert auf } \alpha\} & \text{falls } x \in L(M) \\ \text{undefiniert} & \text{sonst.} \end{cases}\end{aligned}$$

Definition 3.4 (Nichtdeterministische Zeit- und Raumklassen). Seien t und s Funktionen in \mathbb{R} , die von \mathbb{N} in \mathbb{N} abbilden. Wir sagen, M akzeptiert eine Menge A in der Zeit t , falls

- $\text{NTime}_M(x) \leq t(|x|)$ für jedes $x \in A$ gilt, und
- x von M nicht akzeptiert wird, falls $x \notin A$.

Wir sagen, M akzeptiert eine Menge A im Raum s , falls

- $\text{NSpace}_M(x) \leq s(|x|)$ für jedes $x \in A$ gilt, und
- x von M nicht akzeptiert wird, falls $x \notin A$.

Definiere die folgenden nichtdeterministischen Komplexitätsklassen mit der Resourcenfunktion t bzw. s :

$$\begin{aligned} \text{NTIME}(t) &= \left\{ A \mid \begin{array}{l} A = L(M) \text{ für eine NTM } M, \\ \text{die } A \text{ in der Zeit } t(n) \text{ akzeptiert} \end{array} \right\}; \\ \text{NSPACE}(s) &= \left\{ A \mid \begin{array}{l} A = L(M) \text{ für eine NTM } M, \\ \text{die } A \text{ im Raum } s(n) \text{ akzeptiert} \end{array} \right\}. \end{aligned}$$

Natürlich möchte man zum Beispiel $\text{DTIME}(n^2)$ und $\text{DTIME}(n^2 + 1)$ nicht als zwei echt verschiedene Komplexitätsklassen auffassen.¹ Stattdessen ist es sinnvoll, Familien \mathcal{F} von „ähnlichen“ Ressourcenfunktionen zu betrachten und z.B. die \mathcal{F} entsprechende deterministische Zeitklasse durch $\text{DTIME}(\mathcal{F}) = \bigcup_{f \in \mathcal{F}} \text{DTIME}(f)$ zu definieren. Eine solche Familie \mathcal{F} enthält alle Ressourcenfunktionen mit ähnlicher Wachstumsrate. Betrachte beispielsweise die folgenden Funktionenfamilien, die jeweils von \mathbb{N} in \mathbb{N} abbilden:

- $\mathbb{L}\text{in}$ enthält alle linearen Funktionen,
- $\mathbb{P}\text{ol}$ enthält alle Polynome,
- $2^{\mathbb{L}\text{in}}$ enthält alle Exponentialfunktionen, deren Exponent linear in n ist, und
- $2^{\mathbb{P}\text{ol}}$ enthält alle Exponentialfunktionen, deren Exponent polynomiell in n ist.

Allgemeiner kann für jede Funktion $t : \mathbb{N} \rightarrow \mathbb{N}$ die Familie aller Funktionen definiert werden, die linear in t (bzw. polynomiell in t) sind:

$$\begin{aligned} \mathbb{L}\text{in}(t) &= \{f \mid f = \ell \circ t \text{ und } \ell \in \mathbb{L}\text{in}\}; \\ \mathbb{P}\text{ol}(t) &= \{f \mid f = p \circ t \text{ und } p \in \mathbb{P}\text{ol}\}, \end{aligned}$$

wobei die Komposition zweier Funktionen, g und h , die durch $g \circ h(n) = g(h(n))$ definierte Funktion ist. Ähnlich definiert man die Familien $2^{\mathbb{L}\text{in}(t)}$ und $2^{\mathbb{P}\text{ol}(t)}$. Es gilt $\mathbb{L}\text{in} = \mathbb{L}\text{in}(\text{id})$, $\mathbb{P}\text{ol} = \mathbb{P}\text{ol}(\text{id})$, $2^{\mathbb{L}\text{in}} = 2^{\mathbb{L}\text{in}(\text{id})}$ und $2^{\mathbb{P}\text{ol}} = 2^{\mathbb{P}\text{ol}(\text{id})}$.

Die Komplexitätsmaße Zeit und Raum sind, wie auch die resultierenden Komplexitätsklassen, offenbar invariant unter endlicher Variation, da eine endliche Anzahl von Ausnahmen immer mittels „table-lookup“ behandelt werden kann, wobei die Tabelle von Ausnahmen im Programm einer Turingmaschine „hart verdrahtet“ wird. Die \mathcal{O} - und o -Notationen tragen dieser Tatsache Rechnung. Definition 3.5 unten definiert diese und andere Bezeichnungen für Wachstumsraten.

Definition 3.5 (Asymptotische Wachstumsraten). Definiere für Funktionen f und g von \mathbb{N} in \mathbb{N} die folgenden Bezeichnungen:

- $f(n) \leq_{ae} g(n)$ bedeutet, dass $f(n) \leq g(n)$ für alle bis auf endlich viele $n \in \mathbb{N}$ gilt. Analog werden die Bezeichnungen $<_{ae}$, \geq_{ae} und $>_{ae}$ definiert. Der Index „ae“ von „ \leq_{ae} “ usw. steht für „fast überall“ (englisch „almost everywhere“).
- $f(n) \leq_{io} g(n)$ bedeutet, dass $f(n) \leq g(n)$ für unendlich viele $n \in \mathbb{N}$ gilt. Analog werden die Bezeichnungen $<_{io}$, \geq_{io} und $>_{io}$ definiert. Der Index „io“ von „ \leq_{io} “ usw. steht für „fast überall“ (englisch „infinitely often“).

¹ Tatsächlich zeigt der lineare Beschleunigungssatz in Abschnitt 3.3, dass sie nicht verschieden sind.

- $f \in \mathcal{O}(g) \iff$ es gibt eine reelle Konstante $c > 0$ mit $f(n) + 1 \leq_{\text{ae}} c \cdot (g(n) + 1)$.
- $f \in o(g) \iff$ für alle reellen Konstanten $c > 0$ gilt $f(n) + 1 <_{\text{ae}} c \cdot (g(n) + 1)$.
- $f \preceq g \iff \limsup_{n \rightarrow \infty} \frac{f(n)+1}{g(n)+1} < \infty$. Es gilt: $f \in \mathcal{O}(g) \iff f \preceq g$.
Intuitiv bedeutet $f \preceq g$, dass f größerenordnungsmäßig nicht schneller als g wächst, wobei höchstens endlich viele Ausnahmen erlaubt sind.
- $f \prec g \iff \limsup_{n \rightarrow \infty} \frac{f(n)+1}{g(n)+1} = 0$. Es gilt: $f \in o(g) \iff f \prec g$.
Intuitiv bedeutet $f \prec g$, dass g größerenordnungsmäßig echt schneller als f wächst, wobei höchstens endlich viele Ausnahmen erlaubt sind.
- $f \preceq_{\text{io}} g \iff \liminf_{n \rightarrow \infty} \frac{f(n)+1}{g(n)+1} < \infty$.
Intuitiv bedeutet $f \preceq_{\text{io}} g$, dass f größerenordnungsmäßig nicht schneller als g wächst, wenigstens für unendlich viele Argumente nicht.
- $f \prec_{\text{io}} g \iff \liminf_{n \rightarrow \infty} \frac{f(n)+1}{g(n)+1} = 0$.
Intuitiv bedeutet $f \prec_{\text{io}} g$, dass g größerenordnungsmäßig echt schneller als f wächst, wenigstens für unendlich viele Argumente.

Man schreibt $f \succeq g$ für $g \preceq f$, $f \succ g$ für $g \prec f$, $f \succeq_{\text{io}} g$ für $g \preceq_{\text{io}} f$ und $f \succ_{\text{io}} g$ für $g \prec_{\text{io}} f$.

Die additive Konstante 1 in den Nennern der Brüche oben, deren unterer bzw. oberer Limes berechnet wird, soll lediglich verhindern, dass die Nenner den Wert null haben, damit die Brüche wohldefiniert sind. Dagegen soll in Ausdrücken der Form $\limsup_{n \rightarrow \infty} (f(n) + 1)/(g(n) + 1)$ die additive Konstante 1 im Zähler verhindern, dass der obere Limes gegen null geht, ohne dass g echt schneller als f wächst. Beispielsweise sollten die konstanten Funktionen 0 und 2, die beide überhaupt nicht wachsen, $0 \preceq 2 \preceq 0$ erfüllen. Ohne die additive 1 jedoch würde $0 \prec 2$ gelten, was nicht wünschenswert ist und der Intuition widerspricht.

Satz 3.6. 1. $\text{DTIME}(t) \subseteq \text{DSPACE}(t)$.

2. Ist $s \geq \log$, so gilt $\text{DSPACE}(s) \subseteq \text{DTIME}(2^{\text{Lin}(s)})$.

Beweis. Die erste Aussage ist unmittelbar klar, da eine Turingmaschine in der Zeit t ihre Köpfe um höchstens t Bandzellen bewegen kann.

Um die zweite Aussage zu beweisen, sei M eine DTM, die im Raum $s(n)$ und in der Zeit $t(n)$ arbeitet. Angenommen, M hat q Zustände, k Arbeitsbänder und ein Eingabeband, und ein Arbeitsalphabet mit ℓ Symbolen. Für jede Eingabe x der Länge n ist die Zeitschranke $t(n)$ von M nach oben durch die Anzahl der verschiedenen Konfigurationen von $M(x)$ beschränkt. Denn gäbe es eine Konfiguration, die doppelt in der Berechnung von $M(x)$ vorkommt, dann würde M , die ja deterministisch arbeitet, in eine Endlosschleife geraten und nie halten, ein Widerspruch.

Wie viele verschiedene Konfigurationen kann $M(x)$ haben? Es gibt q mögliche Zustände, n mögliche Kopfpositionen auf dem Eingabeband, $(s(n))^k$ mögliche Kopfpositionen auf den k Arbeitsbändern und $\ell^{k \cdot s(n)}$ mögliche Bandinschriften. Folglich gibt es geeignete positive Konstanten a, b und c , so dass gilt:

$$\begin{aligned} t(n) &\leq q \cdot n \cdot (s(n))^k \cdot \ell^{k \cdot s(n)} \leq q \cdot 2^{\log n} \cdot 2^{a \cdot s(n)} \\ &\leq q \cdot 2^{b \cdot s(n)} \leq 2^{c \cdot s(n)}, \end{aligned}$$

wobei die dritte Ungleichung die Annahme $s \geq \log$ benutzt. Somit ist t in $2^{\text{Lin}(s)}$. \square

Tabelle 3.1. Einige typische Komplexitätsklassen

Raumklassen	Zeitklassen
$L = \text{DSPACE}(\log)$	$\text{REALTIME} = \text{DTIME}(\text{id})$
$NL = \text{NSPACE}(\log)$	$\text{LINTIME} = \text{DTIME}(\text{Lin})$
$\text{LINSPACE} = \text{DSPACE}(\text{Lin})$	$P = \text{DTIME}(\text{IPol})$
$\text{NLINSPACE} = \text{NSPACE}(\text{Lin})$	$NP = \text{NTIME}(\text{IPol})$
$\text{PSPACE} = \text{DSPACE}(\text{IPol})$	$E = \text{DTIME}(2^{\text{Lin}})$
$\text{NPSPACE} = \text{NSPACE}(\text{IPol})$	$NE = \text{NTIME}(2^{\text{Lin}})$
$\text{EXPSPACE} = \text{DSPACE}(2^{\text{IPol}})$	$\text{EXP} = \text{DTIME}(2^{\text{IPol}})$
$\text{NEXPSPACE} = \text{NSPACE}(2^{\text{IPol}})$	$\text{NEXP} = \text{NTIME}(2^{\text{IPol}})$

Tabelle 3.1 zeigt einige der wichtigsten deterministischen und nichtdeterministischen Komplexitätsklassen. Wie bereits erwähnt, ist die logarithmische Ressourcenfunktion nur für Raumklassen sinnvoll, nicht aber für Zeitklassen. Es wird sich später zeigen, dass $\text{PSPACE} = \text{NPSPACE}$ gilt, siehe Korollar 3.31 unten. Alle anderen Paare von Komplexitätsklassen in Tabelle 3.1 sind jedoch entweder beweisbar verschiedene Klassen oder es ist nicht bekannt, ob sie gleich sind oder nicht.

Es ist kein Zufall, dass die Polynomial- und Exponentiafunktionen als die wichtigsten Ressourcenfunktionen angesehen werden. Für praktische Zwecke ist es üblich, die in Polynomialzeit laufenden Algorithmen als „machbar“ oder „praktikabel“ (englisch „feasible“) zu betrachten, wohingegen Algorithmen mit einer exponentiellen unteren Zeitschranke als „widerspenstig“ (englisch „intractable“) eingestuft werden.

Dogma 3.7 *Polynomialzeit erfasst den intuitiven Begriff der Effizienz, und Exponentialzeit erfasst den intuitiven Begriff der Ineffizienz.*

Dogma 3.7 wird durch die Beobachtung gestützt, dass sich Polynomial- und Exponentiafunktionen signifikant in ihrer asymptotischen Wachstumsrate unterscheiden. Tabelle 3.2 vergleicht die Wachstumsraten bestimmter typischer polynomialer und exponentieller Zeitfunktionen $t(n)$, wobei angenommen wird, dass ein Computer eine Million Instruktionen pro Sekunde ausführt und auf ihm ein Algorithmus mit Zeitschranke $t(n)$ für bestimmte Eingaben läuft, deren Größe n im praktisch relevanten Bereich bis zu $n = 60$ liegt. Man stellt fest, dass alle in Tabelle 3.2 dargestellten Polynome in n bis $n = 60$ eine erträgliche Ausführungszeit haben. Hingegen benötigt ein 3^n -zeitbeschränkter Algorithmus bereits Jahre, um das Problem für Instanzen der Größe $n = 30$ zu lösen, er benötigt Jahrtausende für Eingaben der Größe $n = 40$, und für $n = 50$ und $n = 60$ nimmt seine Ausführungszeit wahrhaft astronomische Dimensionen an.

Tabelle 3.3 enthüllt noch mehr: Selbst wenn sich die rasante Entwicklung der Computertechnologie, die in den vergangenen Jahrzehnten erreicht wurde, ungebremst fortsetzen würde (bis die natürlichen physikalischen Grenzen erreicht sind),

Tabelle 3.2. Vergleich einiger Polynomial- und Exponentialfunktionen

$t(n)$	$n = 10$	$n = 20$	$n = 30$	$n = 40$	$n = 50$	$n = 60$
n	.00001 Sek.	.00002 Sek.	.00003 Sek.	.00004 Sek.	.00005 Sek.	.00006 Sek.
n^2	.0001 Sek.	.0004 Sek.	.0009 Sek.	.0016 Sek.	.0025 Sek.	.0036 Sek.
n^3	.001 Sek.	.008 Sek.	.027 Sek.	.064 Sek.	.125 Sek.	.256 Sek.
n^5	.1 Sek.	3.2 Sek.	24.3 Sek.	1.7 Min.	5.2 Min.	13.0 Min.
2^n	.001 Sek.	1.0 Sek.	17.9 Min.	12.7 Tage	35.7 Jahre	366 Jhdte.
3^n	.059 Sek.	58 Min.	6.5 Jahre	3855 Jhdte.	$2 \cdot 10^8 \text{ Jhdte.}$	$1.3 \cdot 10^{13} \text{ Jhdte.}$

so könnte damit keineswegs die absolute Ausführungszeit von Exponentialzeit-Algorithmen für in der Praxis relevante Eingabegrößen signifikant vermindert werden. Tabelle 3.3 zeigt, was passieren würde, wenn uns ein Computer zur Verfügung stünde, der 100-mal oder 1000-mal schneller als die heute verwendeten Computer ist. Für die Funktion $t_i(n)$, mit $i \in \{1, 2, \dots, 6\}$, ist N_i die maximale Eingabegröße, die ein $t_i(n)$ -zeitbeschränkter Algorithmus innerhalb einer Stunde bearbeiten kann. Wie man sieht, addiert eine tausendfache Erhöhung der Rechengeschwindigkeit lediglich etwa 10 zur Größe einer größten Probleminstanz, die ein 2^n -zeitbeschränkter Algorithmus innerhalb einer Stunde lösen kann. Im Gegensatz dazu kann in einer Stunde ein n^5 -zeitbeschränkter Algorithmus mit demselben Zuwachs an Rechengeschwindigkeit Eingabegrößen bearbeiten, die etwa viermal größer sind. Die Werte in beiden Tabellen sind aus Garey und Johnson [GJ79] zitiert.

Tabelle 3.3. Was passiert, wenn die Computer schneller werden?

$t_i(n)$	Computer heute	100-mal schneller	1000-mal schneller
$t_1(n) = n$	N_1	$100 \cdot N_1$	$1000 \cdot N_1$
$t_2(n) = n^2$	N_2	$10 \cdot N_2$	$31.6 \cdot N_2$
$t_3(n) = n^3$	N_3	$4.64 \cdot N_3$	$10 \cdot N_3$
$t_4(n) = n^5$	N_4	$2.5 \cdot N_4$	$3.98 \cdot N_4$
$t_5(n) = 2^n$	N_5	$N_5 + 6.64$	$N_5 + 9.97$
$t_6(n) = 3^n$	N_6	$N_6 + 4.19$	$N_6 + 6.29$

Natürlich ist ein Dogma nur ein Dogma, eine Glaubenssache, und als solche sollte Dogma 3.7 kritisch hinterfragt werden. Offensichtlich ist ein Algorithmus mit Laufzeit $n^{10^{77}}$ – was formal ein Polynom ist, dessen Grad zufällig ungefähr gleich der derzeit geschätzten Anzahl von Atomen im sichtbaren Universum ist – ineffizient und nicht praktikabel und nicht einmal für triviale Eingabegrößen wie $n = 2$ nützlich. Auch könnte man zu Recht argumentieren, dass selbst ein Polynom vom Grad, sagen wir, zehn weit davon entfernt ist, als Laufzeit eines praktikablen oder effizienten Algorithmus zu dienen, denn diese wäre nicht einmal für bescheidene Eingabegrößen nützlich. Andererseits darf eine exponentielle Zeitschranke wie etwa $2^{0.0001 \cdot n}$ für die meisten praktisch relevanten Eingabegrößen durchaus als machbar gelten – bevor die exponentielle Wachstumsrate zuschlägt und die Ausführungszeit

ihren Tribut zollen muss. Jedoch ist für die überwältigende Mehrheit natürlicher Probleme, die einen in Polynomialzeit laufenden Algorithmus haben, die Zeitschranke tatsächlich ein Polynom niedrigen Grades, wie z.B. $\mathcal{O}(n^2)$ oder $\mathcal{O}(n^3)$; Polynome vom Grad vier oder fünf oder noch höher treten viel seltener auf. Probleme, die zwar in Polynomialzeit, aber nur für ein Polynom mit beweisbar hohem Grad gelöst werden können, existieren allerdings; L. Hemaspaandra und Ogihsara erwähnen einige solche Ergebnisse auf Seite 264 ihres Buches [HO02]. In Abschnitt 7.2.4 wird ein gefeierter deterministischer Algorithmus für das Primzahlproblem erwähnt, der in seiner ursprünglichen Version in der Zeit $\mathcal{O}(n^{12})$ läuft, der aber inzwischen bis zur Schranke $\mathcal{O}(n^6)$ verbessert werden konnte. Trotz alledem wollen wir uns in diesem Buch durchweg an Dogma 3.7 halten.

3.3 Beschleunigungs-, Kompressions- und Hierarchiesätze

Die zentrale Frage in diesem Abschnitt ist: *Wie stark muss eine Ressource vergrößert werden, damit echt mehr berechnet werden kann?* Betrachte beispielsweise die deterministische Zeitklasse $\text{DTIME}(t_1)$, für eine Ressourcenfunktion t_1 . Wie viel stärker als t_1 muss eine andere Funktion t_2 wachsen, damit die Ungleichheit $\text{DTIME}(t_1) \neq \text{DTIME}(t_2)$ gilt? Die linearen Raumkompressions- und Beschleunigungssätze (siehe die Sätze 3.10 und 3.11 unten) sagen, dass ein linearer Zuwachs der gegebenen Ressourcenfunktion *nicht* genügt, um eine echt größere Komplexitätsklasse zu erhalten.

Bevor wir uns diesen Sätzen zuwenden, ist festzuhalten, dass es möglich ist, beliebig komplexe Probleme zu konstruieren, Probleme also, die eine jede vorgegebene Komplexitätsschranke übersteigen.

Fakt 3.8 *Für jede Funktion $t \in \mathbb{R}$ gibt es ein Problem A_t , so dass $A_t \notin \text{DTIME}(t)$.*

Beweis. Der Beweis beruht auf einer Diagonalisierung. Sei M_0, M_1, M_2, \dots eine Gödelisierung (d.h. eine effektive Nummerierung) aller DTMs. Definiere

$$A_t = \{0^i \mid M_i \text{ akzeptiert } 0^i \text{ nicht in } t(i) \text{ Schritten}\}.$$

Angenommen, A_t wäre in $\text{DTIME}(t)$. Dann gibt es ein j mit $L(M_j) = A_t$ und $\text{time}_{M_j}(n) \leq t(n)$ für jedes $n \in \mathbb{N}$. Folglich gilt:

$$\begin{aligned} 0^j \in A_t &\iff M_j \text{ akzeptiert } 0^j \text{ nicht in } t(j) \text{ Schritten} \\ &\iff 0^j \notin L(M_j) = A_t, \end{aligned}$$

ein Widerspruch. Es folgt, dass A_t nicht in $\text{DTIME}(t)$ liegt. □

Da Komplexitätsklassen wie z.B. $\text{DTIME}(t)$ unter endlicher Variation abgeschlossen sind (siehe Übung 3.2), bedeutet „ $A_t \in \text{DTIME}(t)$ “: „Für eine DTM M gilt $L(M) = A_t$ und $\text{time}_M(n) \leq_{\text{ae}} t(n)$.“ (Bezeichnungen wie „ \leq_{ae} “ wurden in Definition 3.5 erklärt.) Folglich bedeutet „ $A_t \notin \text{DTIME}(t)$ “ oben: „Für jede DTM M mit

$L(M) = A_t$ gilt $\text{time}_M(n) >_{\text{io}} t(n)$.“ Jedoch schließt „ $\text{time}_M(n) >_{\text{io}} t(n)$ “ nicht aus, dass $\text{time}_M(n) \leq_{\text{io}} t(n)$ für unendlich viele andere $n \in \mathbb{N}$ trotzdem gelten kann. In diesem Sinn ist Rabins Satz [Rab60] unten viel stärker als Fakt 3.8. Auf den Beweis von Rabins Satz, der in seiner Diagonalisierung ein cleveres Prioritätsargument benutzt, wird hier verzichtet.

Satz 3.9 (Rabin). Für jede Funktion $t \in \mathbb{R}$ gibt es eine entscheidbare Menge D_t , so dass für jede DTM M , die D_t entscheidet, $\text{time}_M(n) >_{\text{ae}} t(n)$ gilt.

Nun wenden wir uns den Raumkompressions- und Beschleunigungssätzen zu.

Satz 3.10 (Lineare Raumkompression). Für jede Funktion $s \in \mathbb{R}$ gilt:

$$\text{DSPACE}(s) = \text{DSPACE}(\text{Lin}(s)).$$

Beweis. Es genügt, $\text{DSPACE}(2s) \subseteq \text{DSPACE}(s)$ zu zeigen. Sei M eine DTM, die bei Eingaben x der Länge n im Raum $2s(n)$ arbeitet. Es ist nützlich, wenn wir ohne Beschränkung der Allgemeinheit die folgenden Eigenschaften von M fordern: (a) M habe nur ein Band, welches (b) einseitig unendlich sei, (c) die Bandzellen seien mit 1, 2, etc. nummeriert, und (d) der Kopf von M mache nur auf geradzahligen Zellen eine Linkswendung. (Sollte die gegebene Maschine nicht alle diese Eigenschaften haben, so ist es nicht schwer, sie durch eine äquivalente Maschine mit den gewünschten Eigenschaften zu ersetzen; siehe Übung 3.4.) Sei Γ das Arbeitsalphabet von M .

Das Ziel ist es, eine neue DTM N zu konstruieren, die bei einer Eingabe x der Länge n die Berechnung von $M(x)$ simuliert, dabei aber im Raum $s(n)$ arbeitet. Die Idee ist, dass N die Simulation von M „verzögert“: N wartet ab und beobachtet, was M als Nächstes tun wird, bevor sie selbst mehrere Schritte von M gleichzeitig tut. Der Preis, den N dafür zahlt, ist eine größere Zahl von Zuständen, auch hat sie ein größeres Arbeitsalphabet, nämlich $\Gamma \times \Gamma$. Zur Umsetzung der Idee wird das Band von M in Blöcke von je zwei benachbarten Zellen unterteilt, d.h., die Blöcke sind Paare von Zellen mit den Nummern $(2i-1, 2i)$, für $i \geq 1$. Jeder solche Block wird nun als eine Bandzelle von N und jedes geordnete Paar von Symbolen $(a, b) \in \Gamma \times \Gamma$ wird nun als ein Symbol von N aufgefasst. Dann simuliert $N(x)$ die Berechnung von $M(x)$, bis auf dass N ihren Kopf nach links oder rechts nur dann bewegt, wenn der Kopf von M eine Grenze zwischen zwei Blöcken nach links oder rechts überschreitet. Alle Schritte von M innerhalb eines Blockes kann N mit ihrem inneren Gedächtnis (durch neue Zustände) simulieren. Offenbar führt $N(x)$ genau dieselbe Berechnung wie $M(x)$ aus und benötigt nur den Raum $s(n)$. \square

Der lineare Beschleunigungssatz unten macht eine ähnliche Aussage über die Komplexitätsressource Zeit. Sein Beweis ist etwas komplizierter, und die Aussage des Satzes ist etwas eingeschränkter: Lineare Beschleunigung kann nur für solche Ressourcenfunktionen erreicht werden, die bis auf endlich viele Ausnahmen echt stärker wachsen als die Identitätsfunktion. Dies ist jedoch keine massive Einschränkung, denn für deterministische Klassen ist es sowieso nicht sinnvoll, Zeitfunktionen unterhalb der Identität zu betrachten.

Satz 3.11 (Lineare Beschleunigung). Für jede Funktion $t \in \mathbb{R}$ mit $\text{id} \prec t$ gilt:

$$\text{DTIME}(t) = \text{DTIME}(\text{Lin}(t)).$$

Beweis. Sei A eine beliebige Menge in $\text{DTIME}(t)$, und sei M eine DTM, so dass $L(M) = A$ gilt und M bei Eingaben der Länge n in der Zeit $t(n)$ arbeitet. Das Ziel ist nun, eine DTM N mit $L(N) = A$ zu konstruieren, die aber mindestens m -mal schneller als M ist, für eine Konstante $m > 1$. Das heißt, m Schritte von M sollen durch einen Schritt von N simuliert werden. Wieder ist die Idee, dass sich Geduld auszahlt: N „verzögert“ die Simulation von M , d.h., N wartet schlau ab und beobachtet, was M in den nächsten m Schritten tun wird, und dann tut sie das alles auf einen Schlag, mit einem einzigen ihrer Schritte. Und wieder komprimiert N die Eingabe unter Verwendung eines größeren Alphabets und von mehr Zuständen. Jedoch kann N ihre komprimierte Codierung nicht benutzen, bevor sie nicht jedes Eingabzeichen gelesen und die Eingabe in die komprimierte Codierung überführt hat, die später verwendet werden soll. Anders gesagt können die Kopfbewegungen auf dem Eingabeband nicht beschleunigt werden. Deshalb wird die Berechnung von N , bei einer Eingabe x der Länge n , in zwei Phasen unterteilt:

Phase 1: Vorbereitung. Sei $m > 1$ eine feste ganze Zahl, deren Wert später spezifiziert wird. In dieser Phase kopiert N die Eingabe $x \in \Sigma^*$ auf ein Arbeitsband, wobei sie das Eingabeband löscht, und sie codiert die Eingabe wie folgt. Zunächst unterteilt sie das Eingabewort $x = x_1x_2 \cdots x_n$ der Länge n in Blöcke der Länge m , wobei der i -te Block durch das Wort $\beta_i = x_{1+(i-1)m}x_{2+(i-1)m} \cdots x_{im}$, $i \geq 1$, repräsentiert wird.

Somit kann x als $x = \beta_1\beta_2 \cdots \beta_{k+1}$ geschrieben werden, mit $k = \lfloor n/m \rfloor$ und $|\beta_{k+1}| < m$. Hier bezeichnet $\lfloor r \rfloor$ für jede reelle Zahl r die größte ganze Zahl s mit $s \leq r$. Dabei ist β_{k+1} genau dann leer, wenn m die Zahl n teilt. Da ein nichtleeres β_{k+1} mit seinen höchstens m Symbolen im inneren Gedächtnis von N behandelt werden kann, können wir der Einfachheit halber annehmen, dass m tatsächlich n teilt, also $k = n/m$ und β_{k+1} leer ist. Dann schreibt N die folgende redundante Codierung des Eingabewortes auf ihr Arbeitsband:²

$$(\square^m, \beta_1, \beta_2) (\beta_1, \beta_2, \beta_3) (\beta_2, \beta_3, \beta_4) \cdots (\beta_{k-2}, \beta_{k-1}, \beta_k) (\beta_{k-1}, \beta_k, \square^m).$$

Jedes Tripel der Form $(\beta_{i-1}, \beta_i, \beta_{i+1})$, mit $1 < i < k$, oder $(\square^m, \beta_1, \beta_2)$ oder $(\beta_{k-1}, \beta_k, \square^m)$ wird als nur *ein* Symbol von N aufgefasst. Nachdem N die Eingabe in dieser komprimierten (und gleichzeitig etwas redundanten) Form auf ihr Arbeitsband kopiert und ihren Kopf zurück zum am weitesten links stehenden Zeichen bewegt hat, $(\square^m, \beta_1, \beta_2)$, wird dieses Arbeitsband im Weiteren so benutzt, als wäre es das Eingabeband. Das ursprüngliche Eingabeband, das während der ersten Phase gelöscht wurde, wird im Weiteren hingegen als Arbeitsband benutzt.

Phase 1 erfordert $n + k = (1 + 1/m)n$ Schritte.

² Diese Redundanz der Codierung ist notwendig, da ohne sie keine Beschleunigung möglich wäre, falls der Kopf sehr oft zwischen zwei benachbarten Blöcken hin und her wandert.

Phase 2: Simulation. Die obige komprimierte (und etwas redundante) Codierung wird auch für alle Arbeitsbänder von N in Phase 2 verwendet. Es genügt, die Simulation für nur ein Band zu beschreiben. Angenommen, der aktuelle Inhalt dieses Bandes in der Berechnung von $M(x)$ ist ein Wort a der Länge ℓ . Wie oben beschrieben, hat die Codierung von $a = a_1 a_2 \cdots a_\ell$ die Form

$$(\square^m, \alpha_1, \alpha_2)(\alpha_1, \alpha_2, \alpha_3)(\alpha_2, \alpha_3, \alpha_4) \cdots (\alpha_{z-2}, \alpha_{z-1}, \alpha_z)(\alpha_{z-1}, \alpha_z, \square^m),$$

wobei gilt: (1) a ist in $z + 1$ Blöcke unterteilt, also $a = \alpha_1 \alpha_2 \cdots \alpha_{z+1}$, (2) für jedes i mit $1 \leq i \leq z$ hat der Block $\alpha_i = a_{1+(i-1)m} a_{2+(i-1)m} \cdots a_{im}$ die Länge m , und (3) der Block α_{z+1} mit $|\alpha_{z+1}| < m$ wird durch das innere Gedächtnis von N behandelt.

Nun simuliert $N(x)$ eine Folge von m Schritten von $M(x)$ wie folgt. Liest der Kopf von M gegenwärtig eine in einem Block α_j enthaltene Bandzelle, so steht der Kopf von N aktuell auf dem Symbol $(\alpha_{j-1}, \alpha_j, \alpha_{j+1})$.³ Nach m Schritten hat sich der Kopf von M um höchstens m Bandzellen bewegt. Folglich muss er dann auf einer Bandzelle stehen, die zu einem der Blöcke α_{j-1} , α_j oder α_{j+1} gehört, und keiner der anderen Blöcke ist von M verändert worden. Da der Kopf von N gerade auf $(\alpha_{j-1}, \alpha_j, \alpha_{j+1})$ steht, kann sie alle Änderungen von M in nur einem einzigen eigenen Schritt ausführen. Danach bewegt N ihren Kopf auf das Symbol, das dem Block entspricht, in dem sich der Kopf von M nach diesen m Schritten befindet:

- $(\alpha_{j-2}, \alpha_{j-1}, \alpha_j)$, falls M eine Bandzelle in Block α_{j-1} liest;³
- $(\alpha_{j-1}, \alpha_j, \alpha_{j+1})$, falls M eine Bandzelle in Block α_j liest;
- $(\alpha_j, \alpha_{j+1}, \alpha_{j+2})$, falls M eine Bandzelle in Block α_{j+1} liest.³

Akzeptiert oder verwirft M die Eingabe x innerhalb dieser m Schritte, so tut dies auch N . Folglich gilt $L(N) = L(M)$. Phase 2 erfordert höchstens $\lceil t(n)/m \rceil$ Schritte, d.h., in der Simulationsphase ist $N(x)$ ungefähr m -mal schneller als $M(x)$. Hier bezeichnet $\lceil r \rceil$ für jede reelle Zahl r die kleinste ganze Zahl s mit $s \geq r$.

Um den Zeitaufwand von $N(x)$ insgesamt abzuschätzen, wird nun die Annahme $i \prec t$ benutzt, d.h., $n \in o(t(n))$. Somit gilt

$$(\forall c > 0) [n <_{ae} c \cdot t(n)], \quad (3.1)$$

wobei die Bezeichnung „ $f(n) <_{ae} g(n)$ “ für Funktionen f und g in Definition 3.5 erläutert wurde. Addiert man nun den Zeitaufwand beider Phasen auf, so benötigt $N(x)$ nicht mehr als

$$\left(1 + \frac{1}{m}\right)n + \left\lceil \frac{t(n)}{m} \right\rceil <_{ae} \left(1 + \frac{1}{m}\right) \frac{1}{m(1 + \frac{1}{m})} t(n) + \left\lceil \frac{t(n)}{m} \right\rceil \leq \left\lceil \frac{2t(n)}{m} \right\rceil + 1$$

Schritte, wobei die erste Ungleichung aus (3.1) mit der spezifischen Konstante

³ Der Fall, dass \square^m die erste oder die dritte Komponente dieses Tripels ist, wird analog behandelt.

$$\hat{c} = \frac{1}{m(1 + \frac{1}{m})} = \frac{1}{m+1}$$

folgt. Die endlich vielen Ausnahmen, die in der \leq_{ae} -Notation erlaubt sind, können mittels „table-lookup“ behandelt werden. Eine beliebige lineare Beschleunigung ist also durch eine geeignete Wahl von m möglich. \square

Zur konkreten Veranschaulichung sei $t(n) = d \cdot n$ für eine Konstante $d > 1$ gewählt, und $N(x)$ habe die Laufzeit

$$T(n) = \left(1 + \frac{1}{m}\right)n + \frac{t(n)}{m} = \left(1 + \frac{1}{m}\right)n + \frac{d \cdot n}{m} = \left(1 + \frac{d+1}{m}\right)n,$$

wobei wir der Einfachheit halber annehmen, dass m sowohl n als auch $t(n)$ teilt. Da $d > 1$ gilt, impliziert die Wahl von $m > (d+1)/(d-1)$ die Ungleichung $T(n) < d \cdot n = t(n)$ und somit eine echte Beschleunigung. Da die Funktion $t(n) = d \cdot n$, mit $d > 1$, nicht echt stärker als die Identitätsfunktion wächst, zeigt das obige Beispiel, dass die Annahme $\text{id} \prec t$ in Satz 3.11 etwas stärker ist als notwendig. Dieses Beispiel erklärt auch, dass der obige Beweis für $d = 1$ nicht gelingt, d.h., er funktioniert nicht für $t = \text{id}$. Tatsächlich zeigte Rosenberg [Ros67], dass $\text{DTIME}(t) \neq \text{DTIME}(\text{Lin}(t))$ für $t = \text{id}$ gilt.

Satz 3.12 (Rosenberg). $\text{REALTIME} \neq \text{LINTIME}$.

Lineare Raumkompression und Beschleunigung sind auch für nichtdeterministische Komplexitätssklassen bekannt. Interessanterweise kann bei solchen Klassen sogar für die Zeitressource $t = \text{id}$ eine lineare Beschleunigung erreicht werden. Auf den Beweis dieses starken Resultats, das Book und Greibach [BG70] zu verdanken ist, wird hier verzichtet.

Satz 3.13. 1. Für jede Funktion $s \in \mathbb{R}$ gilt $\text{NSPACE}(s) = \text{NSPACE}(\text{Lin}(s))$.
2. Für jede Funktion $t \in \mathbb{R}$ mit $t \geq \text{id}$ gilt $\text{NTIME}(t) = \text{NTIME}(\text{Lin}(t))$.

Doch kehren wir nun zu der im ersten Absatz dieses Abschnitts aufgeworfenen Frage zurück: *Wie stark muss eine Ressource vergrößert werden, damit echt mehr berechnet werden kann?* Wir wissen aus den Sätzen 3.10 und 3.11, dass ein linearer Zuwachs der gegebenen Ressourcenfunktionen nicht genügt, um echt größere Klassen zu erhalten. Sind s_1 und s_2 zum Beispiel Raumfunktionen mit

$$s_2 \preceq s_1 \iff (\exists c > 0) [s_2(n) \leq_{\text{ae}} c \cdot s_1(n)], \quad (3.2)$$

dann wächst s_2 nach Satz 3.10 nicht stark genug, um s_1 an Wirkung zu übertreffen. Also muss man wenigstens verlangen, dass $s_2 \preceq s_1$ nicht gilt. Negiert man nun beide Seiten der Äquivalenz (3.2), so erhält man:

$$s_2 \succ_{\text{io}} s_1 \iff (\forall c > 0) [s_2(n) >_{\text{io}} c \cdot s_1(n)].$$

Der Raumhierarchiesatz (Satz 3.15 unten) sagt, dass es tatsächlich genügt, $s_2 \succ_{\text{io}} s_1$ zu verlangen, um eine echt größere Komplexitätsklasse zu erhalten. Somit komplementieren die Sätze 3.10 und 3.15 einander, und sie sind beide optimal.⁴ Der Zeithierarchiesatz (Satz 3.19 unten) hat eine ähnliche Aussage, wenn er auch eine stärkere Voraussetzung benötigt: Die zwei gegebenen Funktionen müssen sich um mindestens einen logarithmischen Faktor unterscheiden.

Damit die Beweise der Hierarchiesätze für Raum und Zeit funktionieren, müssen die Ressourcenfunktionen eine technische Eigenschaft besitzen: Sie müssen *raumkonstruierbar* bzw. *zeitkonstruierbar* sein. Alle üblichen Ressourcenfunktionen, wie etwa die Logarithmusfunktion, die Polynome in \mathbb{P}^{Pol} , die Exponentialfunktionen in $2^{\mathbb{L}^{\text{Lin}}}$ und in $2^{\mathbb{P}^{\text{Pol}}}$ usw., sind raumkonstruierbar, und alle diese Funktionen bis auf die Logarithmusfunktion sind zeitkonstruierbar; siehe Übung 3.6.

Definition 3.14 (Raum- und Zeitkonstruierbarkeit). Seien f , s und t beliebige Funktionen in \mathbb{R} , die von \mathbb{N} in \mathbb{N} abbilden.

- Die Funktion s heißt *raumkonstruierbar*, falls es eine DTM M gibt, so dass für jedes n gilt: M benötigt bei irgendeiner Eingabe der Länge n nicht mehr als $s(n)$ Bandzellen, um das Wort $\#\#^{s(n)-2}\$$ auf ihr Band zu schreiben, wobei $\#$ und $\$$ spezielle Symbole sind, mit denen der linke und der rechte Rand markiert werden. Man sagt dann, M hat den Raum $s(n)$ ausgelegt.
- Die Funktion f heißt *konstruierbar* in der Zeit t , falls es eine DTM M gibt, so dass für jedes n gilt: M arbeitet bei irgendeiner Eingabe der Länge n genau $t(n)$ Takte und schreibt dabei das Wort $\#\#^{f(n)-2}\$$ auf ihr Band, wobei $\#$ und $\$$ spezielle Symbole sind, mit denen der linke und der rechte Rand markiert werden. Man sagt, t ist *zeitkonstruierbar*, falls t in der Zeit t konstruierbar ist.

Satz 3.15 (Raumhierarchiesatz). Gilt $s_1 \prec_{\text{io}} s_2$ und ist s_2 *raumkonstruierbar*, so gilt:

$$\text{DSPACE}(s_2) \not\subseteq \text{DSPACE}(s_1).$$

Beweis. Der Satz wird nur für den Fall $s_1 \geq \log$ bewiesen. Mit einem Resultat von Sipser [Sip80] kann man diese eigentlich unnötige Einschränkung jedoch loswerden.

Um eine Menge A in der Differenz $\text{DSPACE}(s_2) - \text{DSPACE}(s_1)$ durch Diagonalisierung zu konstruieren, wählen wir eine feste Gödelisierung M_0, M_1, M_2, \dots aller DTMs mit einem Arbeitsband; Übung 3.8 erklärt, weshalb es genügt, ohne Beschränkung der Allgemeinheit nur einbändrige DTMs zu betrachten. Definiere eine DTM N mit einem Eingabeband und drei Arbeitsbändern wie folgt. Bei einer Eingabe $x \in \{0, 1\}^*$ der Länge n arbeite die DTM N so:

1. N legt den Raum $s_2(n)$ auf allen drei Arbeitsbändern aus.

⁴ Einige Bücher geben den Raumhierarchiesatz mit der stärkeren Voraussetzung $s_1 \in o(s_2)$ an, d.h., $s_1 \prec s_2$. Jedoch impliziert $s_1 \prec s_2$ zwar $s_1 \prec_{\text{io}} s_2$, aber umgekehrt impliziert $s_1 \prec_{\text{io}} s_2$ nicht $s_1 \prec s_2$. Indem man diese unnötig starke Bedingung voraussetzt, erhält man nicht das bestmögliche, also nicht das zu Satz 3.10 komplementäre, Resultat und lässt eine Lücke zwischen den Annahmen des Raumhierarchiesatzes und des Satzes über lineare Raumkompression.

2. Sei x von der Form $x = 1^i y$, wobei $0 \leq i \leq n$ und $y \in \{\varepsilon\} \cup 0\{0,1\}^*$. Das heißt, x beginnt mit einem (womöglich leeren) Präfix von i Einsen, gefolgt entweder vom leeren Wort (falls $x = 1^n$) oder gefolgt von einer Null und einem (womöglich leeren) Wort aus $\{0,1\}^*$. Die DTM N interpretiert i als eine Maschinennummer und schreibt das geeignete codierte Programm von M_i auf ihr erstes Arbeitsband. Ist dies nicht möglich, weil das Programm von M_i größer als der ausgelegte Raum $s_2(n)$ ist, so bricht N die Berechnung ab und lehnt x ab. Andernfalls startet N die Simulation der Berechnung von $M_i(x)$ auf dem zweiten Arbeitsband, wobei sie das Programm von M_i auf dem ersten Arbeitsband verwendet und die Symbole von x von ihrem eigenen Eingabeband liest.
3. Das dritte Arbeitsband enthält einen Binärzähler, der anfangs den Wert null enthält und in jedem Takt der Simulation von $M_i(x)$ um eins erhöht wird. Gelingt die Simulation von $M_i(x)$ auf dem zweiten Arbeitsband von N , bevor der Zähler auf dem dritten Arbeitsband von N überläuft, so akzeptiert $N(x)$ genau dann, wenn $M_i(x)$ ablehnt. Andernfalls lehnt N die Eingabe x ab.

Einige technische Erläuterungen sind hier für das Verständnis hilfreich:

- Der Zähler auf dem dritten Arbeitsband von N garantiert, dass $N(x)$ hält, selbst wenn $M_i(x)$ nicht terminieren sollte.
- Es gibt eine Konstante c_i , so dass die Simulation von $M_i(x)$ auf dem zweiten Arbeitsband von N im Raum höchstens $c_i \cdot \text{space}_{M_i}(n)$ gelingt. Weshalb? Die DTM N muss in der Lage sein, eine *jede* DTM M_i , mit $i \in \mathbb{N}$, zu simulieren. Hat M_i für ein i insgesamt z_i Zustände und ℓ_i Symbole in ihrem Arbeitsalphabet, so kann N diese Zustände und Symbole binär codieren, nämlich als Wörter über $\{0,1\}$ der Länge $\lceil z_i \rceil$ bzw. $\lceil \ell_i \rceil$. Diese Codierung verursacht zusätzlich konstante Raumkosten für die simulierende Maschine N , wobei die Konstante, nennen wir sie c_i , nur von M_i abhängt.

Definiere $A = L(N)$. Offenbar ist A in $\text{DSPACE}(s_2)$. Um zu zeigen, dass A nicht in $\text{DSPACE}(s_1)$ liegt, sei zum Zwecke eines Widerspruchs $A \in \text{DSPACE}(s_1)$ angenommen. Dann existiert ein i , so dass $\text{space}_{M_i}(n) \leq s_1(n) \prec_{\text{io}} s_2(n)$ und $A = L(M_i)$ gilt. Bekanntlich bedeutet $s_1 \prec_{\text{io}} s_2$ aber:

$$(\forall c > 0) [s_2(n) >_{\text{io}} c \cdot s_1(n)]. \quad (3.3)$$

Folglich gibt es eine reelle Konstante $c > 0$ und unendlich viele Argumente n_0, n_1, n_2, \dots in \mathbb{N} , so dass $s_2(n_k) > c \cdot s_1(n_k)$ für jedes k gilt. Aus dieser unendlichen Folge von Argumenten sei ein n_j so groß gewählt, dass die folgenden drei Bedingungen gelten:

- Das Programm von M_i kann im Raum $s_2(n_j)$ berechnet und auf das zweite Arbeitsband von N geschrieben werden.
- Die Simulation von $M_i(1^{i_0 n_j - i})$ gelingt im Raum $s_2(n_j)$.
- $\text{time}_{M_i}(n_j) \leq 2^{s_2(n_j)}$.

Die Bedingung (a) kann für ein hinreichend großes n_j erfüllt werden, da die Größe des Programms von M_i nicht von der Eingabe der Maschine abhängt.

Die Bedingung (b) kann für ein hinreichend großes n_j erfüllt werden, denn die Simulation von $M_i(1^i 0^{n_j-i})$ gelingt im Raum

$$c_i \cdot \text{space}_{M_i}(n_j) \leq c_i \cdot s_1(n_j) < s_2(n_j),$$

wobei c_i die obige Konstante ist, die dadurch verursacht wird, dass N die Zustände und Symbole von M_i codieren muss, und wobei die letzte Ungleichung aus (3.3) folgt.

Die Bedingung (c) kann für ein hinreichend großes n_j erfüllt werden, da aus Satz 3.6 für $s_1 \geq \log$ folgt:

$$\begin{aligned} \text{time}_{M_i}(n_j) &\leq 2^{d \cdot \text{space}_{M_i}(n_j)} \quad \text{für eine geeignete Konstante } d \\ &\leq 2^{d \cdot s_1(n_j)} \\ &< 2^{s_2(n_j)}, \end{aligned}$$

wobei sich die letzte Ungleichung wieder aus (3.3) ergibt. Folglich gelingt die Simulation von $M_i(1^i 0^{n_j-i})$, bevor der Binärzähler der Länge $s_2(n_j)$ auf dem dritten Arbeitsband von N überläuft.

Aus den Bedingungen (a), (b) und (c) und der Konstruktion von N folgt für das Wort $x = 1^i 0^{n_j-i}$:

$$\begin{aligned} x \in A &\iff N \text{ akzeptiert } x \\ &\iff M_i \text{ lehnt } x \text{ ab.} \end{aligned}$$

Somit ist $A \neq L(M_i)$, im Widerspruch zu unserer Annahme $A = L(M_i)$. Es folgt $A \notin \text{DSPACE}(s_1)$. \square

Der Beweis von Satz 3.15 liefert eigentlich sogar ein stärkeres Resultat als die im Satz angegebene Aussage.

Korollar 3.16. Für jede raumkonstruierbare Funktion s_2 gilt:

$$\text{DSPACE}(s_2) \not\subseteq \bigcup_{s_1 \prec_{\text{io}} s_2} \text{DSPACE}(s_1).$$

Weiter folgt Korollar 3.17 unten unmittelbar aus Satz 3.15. Dabei bezeichnet $A \subset B$ die *echte Inklusion* zweier Mengen A und B , d.h., $A \subseteq B$ und $A \neq B$.

Korollar 3.17. Gilt $s_1 \leq s_2$ und $s_1 \prec_{\text{io}} s_2$ und ist s_2 raumkonstruierbar, so gilt:

$$\text{DSPACE}(s_1) \subset \text{DSPACE}(s_2).$$

Definiere $\text{POLYLOGSPACE} = \bigcup_{k \geq 1} \text{DSPACE}((\log n)^k)$. Korollar 3.17 liefert eine echte Hierarchie deterministischer Raumklassen. Der Beweis von Korollar 3.18 wird dem Leser als Übung 3.7(a) überlassen.

Korollar 3.18. $L \subset \text{POLYLOGSPACE} \subset \text{LINSPACE} \subset \text{PSPACE} \subset \text{EXPSPACE}$.

Nun wenden wir uns dem Zeithierarchiesatz zu.

Satz 3.19 (Zeithierarchiesatz). *Gilt $t_2 \geq \text{id}$ und $t_1 \prec_{\text{io}} t_2$ und ist t_2 in der Zeit $t_2 \log t_2$ konstruierbar, so gilt:*

$$\text{DTIME}(t_2 \log t_2) \not\subseteq \text{DTIME}(t_1).$$

Hier ist eine Skizze der Beweisidee von Satz 3.19. Der Beweis dieses Satzes beruht wieder auf einer Diagonalisierung, ähnlich der, die im Beweis von Satz 3.15 angegeben wurde. Ausgehend von einer festen Gödelisierung M_0, M_1, M_2, \dots aller DTM's muss die diagonalisierende DTM N , die in der Zeit $t_2 \log t_2$ arbeitet, jede Maschine M_i „schlagen“, d.h., die Konstruktion muss sicherstellen, dass wenn M_i in der Zeit t_1 arbeitet, dann haben N und M_i unterschiedliche Sprachen. Da N in der Lage sein muss, eine jede Mehrband-DTM M_i zu simulieren, stellt sich die Frage, wie N mit ihrer festen Anzahl von Bändern das schaffen kann. Während die Anzahl von Bändern für den Raum, den N in ihrer Simulation von M_i benötigt, keine Rolle spielt, ist diese Anzahl für die Ressource Zeit wichtig. Jede k -bändrige DTM, die im Raum s arbeitet, kann von einer einbändrigen DTM simuliert werden, die dieselbe Anzahl s von Bandzellen braucht, siehe Übung 3.8. Deshalb konnten im Beweis von Satz 3.15 problemlos nur einbändrige DTM's M_i betrachtet werden. Im Gegensatz dazu fordert die Simulation einer k -bändrigen DTM M , die in der Zeit t arbeitet, ihren Preis: Zur Simulation von M braucht eine einbändrige DTM die Zeit t^2 und eine zweibändrige DTM die Zeit $t \log t$. Das letztere Resultat ist Hennie und Stearns [HS66] zu verdanken, und es erklärt die Zeitschranke „ $t_2 \log t_2$ “ in Satz 3.19. Mit diesem Ergebnis bewaffnet, genügt es zu zeigen, dass N jede zweibändrige DTM M_i „schlägt“, und dies kann von einer vierbändrigen DTM N geleistet werden, deren Zeitressource einen zusätzlichen logarithmischen Faktor enthält.

Korollar 3.20. *Gilt $t_1 \leq t_2 \log t_2$ und $t_2 \geq \text{id}$ und $t_1 \prec_{\text{io}} t_2$ und ist t_2 in der Zeit $t_2 \log t_2$ konstruierbar, so gilt $\text{DTIME}(t_1) \subset \text{DTIME}(t_2 \log t_2)$.*

Korollar 3.21. *Für jede Konstante $k > 0$ gilt $\text{DTIME}(n^k) \subset \text{DTIME}(n^{k+1})$ und $\text{DTIME}(2^{k \cdot n}) \subset \text{DTIME}(2^{(k+1)n})$.*

Korollar 3.22. $P \subset E \subset \text{EXP}$.

Story 3.23 Zum Abschluss von Abschnitt 3.3 ist eine kleine traurige Geschichte über zwei Schwestern zu erzählen, Paula und Ella, und über ihre besten Freundinnen, Ann Paula und Ann Ella, die ebenfalls Geschwister sind. Die vierjährige Paula liebt es, mit Ann P. zu spielen, die gerade fünf geworden ist. Ihre großen Schwestern Ella und Ann E., acht bzw. neun Jahre alt, sind ebenso eng befreundet. Doch auch auf die dickste Freundschaft kann mal ein Schatten fallen: Eines Tages streiten sich Paula und Ann P. um Paulas Lieblingsspielzeug, einen tanzenden Hamster, der wie ein alter Hippie aussieht, von einem eingebauten Tonband coole Funkmusik spielt wie etwa den Song „Jungle Boogie“ von 1974 und dazu wie verrückt tanzt. Ann P. ist größer und stärker als Paula, also nimmt sie ihr den tanzenden Hamster weg. Von diesem Tag an gehen die beiden kleinen Mädchen getrennte Wege.

Paulas große Schwester, Ella, liebt dieselbe Musik wie Paula, nicht auf Band in einem Hamster allerdings, sondern hübsch auf CD gebrannt. Sie beobachtet den Streit der Kleinen, wird wütend und überträgt ihren Ärger über Ann P., die ja zum Verhauen doch noch zu klein ist, auf deren große Schwester, Ann E.: Voll Wut schmeißt Ella ihre „Jungle Boogie“-CD nach Ann E. und schreit sie an. Von diesem Tag an gehen auch die beiden großen Mädchen getrennte Wege.

Was hat diese Geschichte mit Komplexitätstheorie zu tun? Nun, was den Kindern passiert ist, kann auch bei Komplexitätsklassen vorkommen, und zwar als das „Upward-Separation“-Resultat von Book. Dieses sagt, dass sich eine Separation zweier kleiner Komplexitätsklassen mittels einer „*tally*“ (d.h. unär codierten) Menge „nach oben“ (englisch „*upward*“) überträgt, um zwei mit ihnen verwandte, viel größere Klassen zu separieren. „Upward Separation“ wird manchmal auch als „Downward Collapse“ bezeichnet, was bedeutet, dass wenn zwei große Komplexitätsklassen übereinstimmen, dann sind auch ihre „kleinen Schwestern“ auf den *tally* Mengen gleich.

Story 3.23 (Fortsetzung) *Mit dieser alternativen Sichtweise kann der glückliche Ausgang der traurigen Geschichte oben noch rasch erzählt werden: Am nächsten Tag, nachdem Ann E. sich mit Ella wieder versöhnt und ihr die CD zurückgebracht hatte und die beiden wieder ein Herz und eine Seele waren, gab auch Ann P. ihrer Freundin Paula den tanzenden Hamster zurück und vertrug sich wieder mit ihr. Viel mehr Vergnügen bereitete es ihnen nun, ihr Spielzeug zu teilen, und die vier waren wieder unzertrennliche Freundinnen.*

Die beiden Geschwisterpaare in der obigen Geschichte entsprechen den folgenden beiden Paaren von „exponentiell verwandten“ Komplexitätsklassen: die deterministischen Zeitklassen P und E und ihre nichtdeterministischen Pendants, NP und NE. Ellas Musik, in obiger Geschichte „hübsch auf CD gebrannt“, entspricht einer gegebenen Sprache L , die kurz und knapp binär dargestellt ist, und Paulas tanzender Hamster entspricht der unären Codierung von L als *tally* Menge.

Definition 3.24 (Unärcodierung („Tally Encoding“) einer Sprache).

- Eine *tally* Sprache ist eine beliebige Teilmenge von $\{1\}^*$, der Menge der Wörter über dem einelementigen Alphabet $\{1\}$. TALLY bezeichne die Menge aller *tally* Sprachen.
- Sei $\Sigma = \{0, 1\}$ ein binäres Alphabet, und sei $L \subseteq \Sigma^*$ eine beliebige Sprache von Wörtern über Σ . Versieht man jedes Wort $x \in L$ mit dem Präfix 1 und interpretiert dann $1x$ als eine natürliche Zahl $\text{bin}(1x)$ in Binärdarstellung, so kann die Unärcodierung („*tally Encoding*“) von L definiert werden als $\text{Tally}(L) = \{1^{\text{bin}(1x)} \mid x \in L\}$.
- Umgekehrt kann jede *tally* Sprache $T \subseteq \{1\}^+$ in eine Menge über Σ transformiert werden, die definiert ist als $\text{Bin}(T) = \{x \in \Sigma^* \mid 1^{\text{bin}(1x)} \in T\}$, wobei das leere Wort (aus technischen Gründen) in T nicht vorkommen soll.

Natürlich gilt $\text{Bin}(\text{Tally}(L)) = L$ für jede Sprache $L \subseteq \Sigma^*$, und für jede *tally* Sprache $T \subseteq \{1\}^+$ ist $\text{Tally}(\text{Bin}(T)) = T$. Die Sprache $\text{Tally}(L)$ ist eine „exponentiell aufgeblähte“ Darstellung von L , denn $\text{Bin}(T)$ enthält dieselbe Information wie $T \in \text{TALLY}$ in „logarithmisch komprimierter“ Form. Diese Beobachtung wird im folgenden Lemma festgehalten, dessen Beweis dem Leser als Übung 3.9 überlassen wird.

Lemma 3.25. *Für jede Menge $L \subseteq \Sigma^*$ gilt:*

1. $L \in E \iff \text{Tally}(L) \in P$ und
2. $L \in NE \iff \text{Tally}(L) \in NP$.

Mittels Lemma 3.25 kann das folgende „Upward-Separation“-Resultat bewiesen werden, das die Separation von NP und P durch eine *tally* Sprache mit der Separation ihrer Exponentialzeit-Pendants verknüpft.

Satz 3.26. $NE = E$ gilt genau dann, wenn jede *tally* Sprache aus NP in P ist.

Beweis. Um die Implikation von links nach rechts zu zeigen, sei $NE = E$ angenommen. Sei $T \subseteq \{1\}^+$ eine beliebige *tally* Sprache in NP. Nach Aussage 2 von Lemma 3.25 ist $\text{Bin}(T)$ in NE. Aus $NE = E$ folgt dann, dass $\text{Bin}(T)$ in E ist. Nach Aussage 1 von Lemma 3.25 schließlich ist T in P.

Um umgekehrt die Implikation von rechts nach links zu beweisen, sei L eine beliebige Menge in NE. Nach Aussage 2 von Lemma 3.25 ist $\text{Tally}(L)$ in NP. Da nach Annahme jede *tally* Sprache aus NP in P liegt, muss insbesondere $\text{Tally}(L)$ in P sein. Aussage 1 von Lemma 3.25 impliziert dann, dass L in E liegt. Also ist $NE \subseteq E$, und somit gilt $NE = E$. \square

3.4 Zwischen Logarithmischem und Polynomialem Raum

Satz 3.27 unten erkundet die Beziehungen zwischen den Komplexitätsklassen aus Tabelle 3.1, die zwischen logarithmischem und polynomialem Raum liegen. Von keiner der angegebenen Inklusionen weiß man, ob sie echt ist, auch wenn man allgemein annimmt, dass sie es alle sind. Die Komplexitätstheorie hat viele wichtige und schöne Resultate hervorgebracht, und ebenso viele wichtige und interessante Fragen offen gelassen. Eine der berühmtesten offenen Fragen in der Komplexitätstheorie ist die Frage, ob P gleich NP ist oder nicht.

Satz 3.27. $L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE$.

Beweis. Die Inklusionen $L \subseteq NL$ und $P \subseteq NP$ sind unmittelbar klar, weil jede DTM nach Definition eine spezielle NTM ist.

Um zu beweisen, dass $NP \subseteq PSPACE$ gilt, sei M eine beliebige NTM, die in der Zeit $p(n)$ für ein $p \in \text{IPol}$ arbeitet. Bei der Konstruktion einer DTM N , die $L(M)$

in polynomialem Raum entscheidet, nutzen wir eine wichtige Eigenschaft der Komplexitätsressource Raum aus: Anders als die Zeitressource ist *Raum wiederverwendbar*. Indem sie denselben Raum auf ihrem Arbeitsband wieder und wieder verwendet, führt die DTM N eine Tiefensuche durch den Berechnungsbaum von $M(x)$ aus, durchläuft also Pfad für Pfad in diesem Baum auf der Suche nach einer akzeptierenden Konfiguration.

Konstruiere die DTM N , die bei einer Eingabe x der Länge n arbeitet, folgendermaßen. Zusätzlich zu ihrem Eingabeband hat N ein in drei Spuren unterteiltes Arbeitsband. Da alle Polynome raumkonstruierbar sind, kann N zunächst den Raum $p(n)$ auf ihrem Arbeitsband auslegen, d.h., N markiert genau $p(n)$ Zellen. Dann durchläuft N systematisch den Berechnungsbaum von $M(x)$ mit Tiefensuche. Um die Übersicht über die aktuelle Position in der Suche zu behalten, schreibt N den Namen des Pfades von $M(x)$, der gegenwärtig durchlaufen wird, auf die Spur 1 ihres Arbeitsbands. Pfadnamen werden dabei binär⁵ durch ein Wort $s \in \{0, 1\}^*$ der Länge $p(n)$ codiert, wobei das i -te Bit von s die i -te Verzweigung dieses Pfades in $M(x)$ darstellt. Die Bits von s dürfen nach Bedarf mit einem Dach markiert werden, um anzugeben, dass die markierten Bits von s dem Anfangsteil des Pfades entsprechen, der bereits abgearbeitet ist.

Zu Beginn schreibt N den Pfadnamen $0^{p(n)}$ auf die Spur 1 und die Startkonfiguration von $M(x)$ auf die Spur 2. Dann beginnt N mit ihrer Suche durch Simulation der Berechnung von $M(x)$ entlang des Pfades, dessen Name s derzeit auf Spur 1 steht. N schreibt je zwei aufeinander folgende Konfigurationen, die entlang dieses Pfades vorkommen, abwechselnd auf die Spuren 2 und 3. Sei $C_0 \vdash_M C_1 \vdash_M \dots \vdash_M C_{p(n)}$ die s entsprechende Konfigurationenfolge. Betrachte für i mit $1 \leq i < p(n)$ die drei aufeinander folgenden Konfigurationen C_{i-1}, C_i und C_{i+1} , wobei gilt: (a) $C_{i-1} \vdash_M C_i \vdash_M C_{i+1}$, (b) C_i ist die erste bislang unbesuchte Konfiguration entlang s , (c) die ersten $i - 1$ Bits von s sind derzeit mit einem Dach markiert, und (d) aktuell enthält z.B. die Spur 2 die Konfiguration C_{i-1} . Dann schreibt N die Konfiguration C_i auf die Spur 3, wobei sie den vorherigen Inhalt von Spur 3 löscht und das i -te Bit von s auf Spur 1 mit einem Dach versieht. N fährt in dieser Weise fort und vertauscht abwechselnd die Rollen von Spur 2 und Spur 3, bis der aktuelle Pfad vollständig abgearbeitet ist und alle Bits von s mit einem Dach gekennzeichnet sind. Wird eine akzeptierende Konfiguration erreicht, so hält N und akzeptiert x . Ist s abgearbeitet, ohne dass eine akzeptierende Konfiguration erreicht worden ist, so erhöht N den Pfadnamen in Spur 1 (aufgefasst als eine Binärzahl) um eins und sucht weiter nach noch unbesuchten Konfigurationen auf diesem neuen Pfad. Sind sämtliche Pfade von $M(x)$ ohne Erfolg abgearbeitet worden, so lehnt N die Eingabe x ab. Es folgt $L(N) = L(M)$. Da N deterministisch in polynomialem Raum arbeitet, ergibt sich $\text{NP} \subseteq \text{PSPACE}$.

Nun wird die letzte noch fehlende Inklusion gezeigt, $\text{NL} \subseteq \text{P}$. Sei \hat{M} eine beliebige NTM, die in logarithmischem Raum arbeitet, und sei x irgendeine Eingabe der Länge n . Zunächst ist festzustellen, dass die obige systematische Suche durch den ganzen Berechnungsbaum hier nicht funktioniert. Denn eine in logarithm-

⁵ Ohne Beschränkung der Allgemeinheit dürfen wir annehmen, dass $M(x)$ ein vollständiger Binärbaum ist, d.h., alle Pfade von $M(x)$ haben dieselbe Länge $p(n)$.

mischem Raum arbeitende NTM kann bis zu $q(n)$ Schritte auf jedem Pfad machen, $q \in \text{IPol}$, was insgesamt $2^{q(n)}$ mögliche Berechnungspfade ergibt. Daher hat eine polynomialzeit-beschränkte DTM keine Möglichkeit, jeden Pfad von $\widehat{M}(x)$ bis zu seiner vollen Länge von $q(n)$ Schritten zu überprüfen. Glücklicherweise jedoch zeigt ein Argument, das analog zum Beweis von Satz 3.6 ist, dass es eine Konstante c gibt, die nur von \widehat{M} abhängt, so dass die Anzahl der verschiedenen Konfigurationen von $\widehat{M}(x)$ nach oben beschränkt ist durch

$$2^{c \cdot \log n} = 2^{\log n^c} = n^c. \quad (3.4)$$

Das heißt, viele der Konfigurationen im Berechnungsbaum von $\widehat{M}(x)$ der vollen Tiefe $q(n)$ müssen mehrfach auftreten. Folglich genügt es, eine DTM \widehat{N} zu konstruieren, die bei Eingabe x lediglich einen polynomiell großen Teil des Berechnungsbaums von $\widehat{M}(x)$ durchsuchen muss, wobei jeder Pfad abgeschnitten wird, sobald man einer Konfiguration das zweite Mal begegnet. Zu diesem Zweck verwendet \widehat{N} drei Arbeitsbänder. Band 1 wird wieder in drei Spuren unterteilt und in derselben Weise wie das Arbeitsband der DTM N im obigen Beweis von $\text{NP} \subseteq \text{PSPACE}$ verwendet. Das heißt, in Spur 1 dieses Bandes merkt sich \widehat{N} wieder den aktuellen Pfadnamen und welcher Teil davon bereits abgearbeitet wurde, um die Übersicht über die aktuelle Position in der Suche zu behalten. Die Spuren 2 und 3 auf diesem Band speichern wieder je eine von zwei aufeinander folgenden Konfigurationen von $\widehat{M}(x)$. Dabei erzeugt \widehat{N} , zwischen diesen beiden Spuren alternierend, die nächste Konfiguration entlang des Pfades, dessen Name gerade in Spur 1 steht.

Außerdem hat die DTM \widehat{N} zwei weitere Bänder. Band 2 enthält eine Liste aller neuen Konfigurationen von $\widehat{M}(x)$, die bisher gefunden wurden. Sobald eine neue Konfiguration, nennen wir sie C , entweder auf Spur 2 oder Spur 3 von Band 1 erzeugt wurde, wird sie mit jeder Konfiguration verglichen, die zu diesem Zeitpunkt auf Band 2 steht, um zu überprüfen, ob C wirklich neu ist oder nicht. Wenn ja, dann wird C zur Liste auf Band 2 hinzugefügt; andernfalls kann der aktuelle Pfad von $\widehat{M}(x)$ gefahrlos abgeschnitten werden. Band 3 enthält einen Binärzähler der Länge $c \cdot \log n$, wobei c die Konstante aus (3.4) ist. Dieser Zähler wird jedes Mal um eins erhöht, wenn eine neue Konfiguration auf Band 2 hinzukommt. Wird im Verlauf dieses Prozesses eine akzeptierende Konfiguration von $\widehat{M}(x)$ gefunden, so hält N und akzeptiert x . Wird die Suche durch $\widehat{M}(x)$ jedoch erfolglos abgeschlossen oder läuft der Zähler auf Band 3 über, weil die größtmögliche Anzahl von n^c verschiedenen Konfigurationen gefunden wurde und keine von ihnen akzeptierend war, so lehnt N x ab. Es folgt $L(\widehat{N}) = L(\widehat{M})$.

Um den Zeitaufwand abzuschätzen, erinnern wir uns, dass $\widehat{M}(x)$ höchstens n^c Konfigurationen der Länge jeweils $\mathcal{O}(\log n)$ hat. Somit ist die Schrittanzahl, die zum Vergleich der aktuellen Konfiguration auf Band 1 mit dem gesamten Inhalt von Band 2 nötig ist, in $\mathcal{O}(n^c \cdot \log n)$, also in $\mathcal{O}(n^{c+1})$. Folglich erfordert der Vergleich einer jeden der n^c möglichen Konfigurationen auf Band 1 mit dem Inhalt von Band 2 insgesamt höchstens $\mathcal{O}(n^{2c+1})$ Schritte. Ähnlich schätzt man den Zeitaufwand für die Erzeugung neuer Konfigurationen auf Band 1 und für das Kopieren auf Band 2 mit höchstens $\mathcal{O}(n^{c+1})$ Schritten ab. Parallel dazu wird der Zähler auf Band 3 höchstens

n^c -mal erhöht. In der Summe ergibt sich, dass der Zeitaufwand von $\hat{N}(x)$ höchstens polynomiell in n ist. Somit gilt $\text{NL} \subseteq \text{P}$. \square

Dass die Zeitressource im obigen Beweis von $\text{NP} \subseteq \text{PSPACE}$ polynomiell beschränkt ist, ist keine unbedingt nötige Forderung. Ebenso ist nichts Besonderes daran, dass die Raumressource im obigen Beweis von $\text{NL} \subseteq \text{P}$ logarithmisch beschränkt ist. Man kann also die obigen Argumente verallgemeinern und erhält eine stärkere Version von Satz 3.6.

Korollar 3.28. 1. Ist t raumkonstruierbar, so ist $\text{NTIME}(t) \subseteq \text{DSPACE}(t)$.
2. Ist $s \geq \log$ in der Zeit $2^{\mathbb{L}\text{in}(s)}$ konstruierbar, so gilt:

$$\text{NSPACE}(s) \subseteq \text{DTIME}(2^{\mathbb{L}\text{in}(s)}).$$

Aus Korollar 3.28 folgt sofort:

$$\text{NTIME}(t) \subseteq \text{DTIME}(2^{\mathbb{L}\text{in}(t)}) \text{ und } \text{NSPACE}(s) \subseteq \text{DSPACE}(2^{\mathbb{L}\text{in}(s)}),$$

wodurch für Zeit und Raum die Kosten dafür, Nichtdeterminismus gegen Determinismus auszutauschen, nach oben beschränkt werden. Was ist mit den unteren Schranken? Ist ein exponentieller Zuwachs der Ressource wirklich notwendig, um Nichtdeterminismus deterministisch zu simulieren? Für die Zeitressource ist die Antwort auf diese Frage nicht bekannt. Für die Raumressource jedoch ist die Antwort nein! Satz 3.29 zeigt, dass für die Ressource Raum bereits ein quadratischer Zuwachs genügt, damit die deterministische Simulation von Nichtdeterminismus gelingt.

Satz 3.29 (Savitch). Ist $s \geq \log$ raumkonstruierbar, so gilt:

$$\text{NSPACE}(s) \subseteq \text{DSPACE}(s^2).$$

Beweis. Sei A eine beliebige Menge in $\text{NSPACE}(s)$, und sei M eine NTM, die A akzeptiert und bei Eingaben der Länge n im Raum $s(n)$ arbeitet. Das Ziel ist es, eine DTM N zu konstruieren, die A im Raum $\mathcal{O}(s^2)$ entscheidet; nach Satz 3.10 kann die Konstante, die implizit in der \mathcal{O} -Notation erlaubt ist, stillschweigend ignoriert werden.

Da $s \geq \log$ angenommen ist, folgt aus Satz 3.6, dass M die Menge A in der Zeit $t(n) \leq 2^{c \cdot s(n)}$ akzeptiert, für eine Konstante c , die nur vom Programm von M abhängt und gemäß dem Beweis von Satz 3.6 leicht bestimmt werden kann. Weiter treffen wir die folgenden vereinfachenden Annahmen:

- ACCEPT_M ist die eindeutig bestimmte akzeptierende Konfiguration von M bei irgendeiner Eingabe,⁶ und $\text{START}_M(x)$ ist die eindeutig bestimmte Startkonfiguration von M bei Eingabe x .

⁶ Man kann sich leicht davon überzeugen, dass dies ohne Beschränkung der Allgemeinheit angenommen werden darf. Beispielsweise kann man verlangen, dass M genau einen akzeptierenden Zustand hat und dass sie, bevor sie akzeptiert, stets erst den Inhalt ihrer Arbeitsbänder löscht und den Eingabekopf zurück zum am weitesten links stehenden Eingabesymbol bewegt.

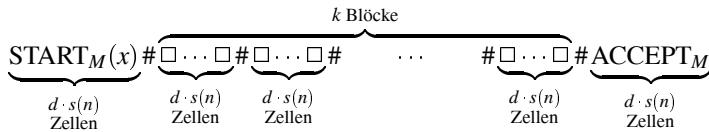
- Sei $k = c \cdot s(n)$. Dabei nehmen wir an, dass $2^{k-1} < t(n) \leq 2^k$ gilt, was $\lceil \log t(n) \rceil = k$ impliziert.
 - Die Konfigurationen von $M(x)$ seien geeignet als Wörter über einem festen Alphabet codiert, und alle solche Wörter mögen genau dieselbe Länge $d \cdot s(n)$ für eine Konstante d haben. Nummeriere alle Wörter der Länge $d \cdot s(n)$ als C_1, C_2, \dots, C_m in der lexikographischen Ordnung. Natürlich codieren nicht alle Wörter C_i syntaktisch korrekte Konfigurationen von $M(x)$.

Die Grundidee des Beweises ist eine clevere Teile-und-Herrsche-Strategie, die auf der einfachen Beobachtung beruht, dass für jedes Wort $x \in \Sigma^*$ gilt:

$$\begin{aligned} x \in A &\iff \text{START}_M(x) \vdash_M^{2^k} \text{ACCEPT}_M \\ &\iff (\exists i) [\text{START}_M(x) \vdash_M^{2^{k-1}} C_i \text{ und } C_i \vdash_M^{2^{k-1}} \text{ACCEPT}_M]. \end{aligned} \quad (3.5)$$

Es bleibt zu zeigen, dass sich diese Idee im Raum $\mathcal{O}(s^2)$ verwirklichen lässt. Bei einer Eingabe x der Länge n arbeitet die DTM N wie folgt:

1. N legt den Raum $s(n)$ aus und berechnet den Wert $k = c \cdot s(n)$, der gleich $\lceil \log t(n) \rceil$ ist.
 2. N erzeugt auf ihrem Arbeitsband das folgende Muster:



wobei $\#$ ein spezielles Symbol ist, das diese $k + 2$ Blöcke der Größe jeweils $d \cdot s(n)$ voneinander trennt.

3. $N(x)$ simuliert die Berechnung von $M(x)$ deterministisch in diesen $k+2$ Blöcken, wobei sie denselben Bereich ihres Bandes wieder und immer wieder verwendet. Diese Simulation wird unten im Detail beschrieben; siehe den Beweis von Lemma 3.30.
 4. N akzeptiert x genau dann, wenn sie die ACCEPT_M -Konfiguration im Laufe dieser Simulation erreicht.

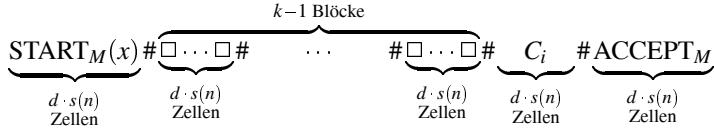
Um zu beweisen, dass die Simulation von $M(x)$ durch N für jedes $x \in A$ gelingt, benötigen wir das folgende Lemma.

Lemma 3.30. Ist $x \in A$, so gelingt die Simulation von $\text{START}_M(x) \vdash_M^{2^k} \text{ACCEPT}_M$ durch N im ausgelegten Raum $s(n)$. Ist jedoch $x \notin A$, so lehnt N die Eingabe x ab.

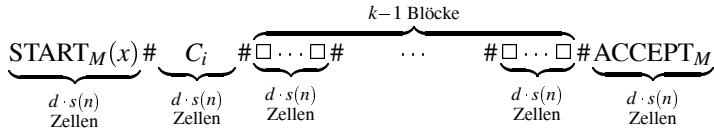
Beweis von Lemma 3.30. Der Beweis wird durch Induktion über k geführt.

$k = 0$: N kann überprüfen, ob $\text{START}_M(x) \vdash_M^{2^0} \text{ACCEPT}_M$ gilt oder nicht, indem sie einen Schritt von $M(x)$ simuliert.

$(k-1) \mapsto k$: N geht systematisch durch alle Wörter C_1, C_2, \dots, C_m der Länge $d \cdot s(n)$, die möglicherweise Konfigurationen von $M(x)$ codieren, und sucht nach einer Konfiguration C_i , die (3.5) erfüllt. Ist C_i das gerade überprüfte Wort, so prüft N zunächst, ob C_i eine syntaktisch korrekte Konfiguration von $M(x)$ codiert. Falls nein, so geht N zur Untersuchung des nächsten Wortes weiter, C_{i+1} . Andernfalls (d.h., wenn C_i eine syntaktisch korrekte Konfiguration von $M(x)$ ist) schreibt N das Wort C_i in den $(k+1)$ -ten Block ihres Arbeitsbands:



und überprüft, ob $\text{START}_M(x) \vdash_M^{2^{k-1}} C_i$ gilt oder nicht, was nach Induktionsannahme möglich ist. Schlägt dieser Test fehl, so löscht N das Wort C_i im $(k+1)$ -ten Block und setzt ihre Arbeit mit der Untersuchung des nächsten Wortes fort, C_{i+1} . Andernfalls (d.h., wenn C_i den Test $\text{START}_M(x) \vdash_M^{2^{k-1}} C_i$ bestanden hat) löscht N das Wort C_i im $(k+1)$ -ten Block, nachdem sie es in den zweiten Block ihres Arbeitsbands kopiert hat:



und überprüft nun, ob $C_i \vdash_M^{2^{k-1}} \text{ACCEPT}_M$ gilt oder nicht, was wieder nach Induktionsannahme möglich ist. Schlägt dieser Test fehl, so löscht N das Wort C_i im zweiten Block und setzt ihre Arbeit mit der Untersuchung des nächsten Wortes fort, C_{i+1} . Andernfalls (d.h., wenn C_i beide Tests gemäß (3.5) bestanden hat, $\text{START}_M(x) \vdash_M^{2^{k-1}} C_i$ und $C_i \vdash_M^{2^{k-1}} \text{ACCEPT}_M$) akzeptiert N die Eingabe x und hält. Wenn keine der potenziellen Konfigurationen von $M(x)$ beide Tests besteht, dann lehnt N die Eingabe x ab und hält.

Damit ist das Lemma bewiesen. □ Lemma 3.30

Nach Lemma 3.30 gilt $L(N) = A$. Da $k = c \cdot s(n)$ gilt und weil es $k+2$ Blöcke der Größe jeweils $d \cdot s(n)$ gibt, arbeitet $N(x)$ im Raum $\mathcal{O}((s(n))^2)$. Aus Satz 3.10 folgt nun $A \in \text{DSPACE}(s^2)$, womit der Satz bewiesen ist. □ Satz 3.29

Korollar 3.31. PSPACE = NPSPACE.

Im vorherigen Abschnitt wurde Satz 3.15 gezeigt, der Hierarchiesatz für deterministische Raumklassen, aus welchem insbesondere $L \subset \text{PSPACE}$ folgt; siehe Korollar 3.18. Eine noch stärkere echte Inklusion wird unten in Korollar 3.32 angegeben, die aus den Sätzen 3.15 und 3.29 mittels der folgenden Kette von Inklusionen folgt, von denen einige bekanntermaßen echt sind:

$$\begin{aligned} \text{NL} &\subseteq \text{DSPACE}((\log n)^2) \subset \text{DSPACE}((\log n)^3) \subset \dots \subset \text{DSPACE(id)} \\ &= \text{LINSPACE} \quad \subset \text{DSPACE}(n^2) \quad \subset \dots \subset \text{PSPACE}. \end{aligned} \quad (3.6)$$

Korollar 3.32. $\text{NL} \subset \text{PSPACE}$.

Alternativ dazu kann Korollar 3.32 mit dem obigen Korollar 3.31 und dem Hierarchiesatz für *nichtdeterministische* Raumklassen bewiesen werden, welcher hier nicht explizit angegeben wird, aber z.B. im Buch von Wagner und Wechsung gefunden werden kann [WW86]. Übung 3.7(b) macht eine noch stärkere Behauptung als Korollar 3.32, und Übung 3.7(c) erweitert die obige Inklusionskette (3.6).

Zwar wissen wir aus den Korollaren 3.18 und 3.32, dass L und sogar NL echt in PSPACE enthalten ist, aber es ist nicht bekannt, welche der Inklusionen aus Satz 3.27 echt ist. Das heißt, es ist nicht bekannt, welche der Relationen „ \subseteq “ in der Inklusionskette

$$L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE$$

tatsächlich sogar eine „ \subset “-Relation ist.

Korollar 3.31 legt nahe, dass Nichtdeterminismus der Komplexitätsressource Raum womöglich viel weniger zusätzliche Berechnungskraft verleiht als der Komplexitätsressource Zeit. Diese Intuition wird auch durch das folgende viel beachtete Resultat unterstützt, das unabhängig von Immerman [Imm88] und Szelepcsenyi [Sze88] entdeckt wurde. Für eine Komplexitätsklasse \mathcal{C} sei $\text{co}\mathcal{C} = \{L \mid \bar{L} \in \mathcal{C}\}$ die Klasse der Komplemente von Mengen in \mathcal{C} .

Satz 3.33. Ist $s \geq \log$ raumkonstruierbar, so ist $\text{NSPACE}(s) = \text{coNSPACE}(s)$.

Satz 3.33 hat für die Spezialfälle $s = \text{id}$ bzw. $s = \log$ wichtige Folgerungen. Insbesondere wird für den Fall $s = \text{id}$ eine offene Frage gelöst, die 1964 von Kuroda [Kur64] gestellt wurde: CS, die Klasse der kontextsensitiven Sprachen, ist komplementabgeschlossen. Bekanntlich ist CS gleich der Komplexitätsklasse NLINSPACE.

Korollar 3.34. $\text{NLINSPACE} = \text{coNLINSPACE}$ und $\text{NL} = \text{coNL}$.

3.5 Reduzierbarkeiten und Vollständigkeit

3.5.1 Many-One-Reduzierbarkeiten, Härte und Vollständigkeit

Angenommen, man hat zwei Probleme gegeben, A und B , und man möchte wissen, ob ihre Komplexität verschieden ist, und wenn ja, welches der beiden schwerer zu lösen ist als das andere. Vielleicht weiß man sogar, dass beide zur selben Komplexitätsklasse gehören, etwa NP. Dennoch könnte A sehr leicht zu lösen sein, zum Beispiel in logarithmischem Raum, wohingegen B womöglich viel härter ist. Denn die Zugehörigkeit eines gegebenen Problems A zu einer Komplexitätsklasse \mathcal{C} liefert lediglich eine obere Schranke für A . Wie kann man untere Schranken für ein Problem beweisen? Wie kann man insbesondere zeigen, dass ein gegebenes Problem

eines der härtesten Probleme einer Komplexitätsklasse ist? Um ein solches Resultat zu beweisen, müsste man zeigen, dass ein *jedes* Problem in der Klasse höchstens so schwer wie das gegebene Problem ist.

Komplexitätsbeschränkte Reduzierbarkeiten sind ein mächtiges Werkzeug zum Vergleich der Komplexität zweier Probleme. Lässt sich eine Menge A auf eine Menge B reduzieren, so bedeutet das intuitiv, dass B mindestens so hart wie A ist. Der Begriff der *Härte* für eine Komplexitätsklasse \mathcal{C} , bezüglich einer Reduzierbarkeit, formalisiert den intuitiven Begriff einer unteren Schranke: Ist B hart für \mathcal{C} , dann ist jede Menge A aus \mathcal{C} auf B reduzierbar. Ist B nicht nur hart für \mathcal{C} , sondern auch in \mathcal{C} enthalten, dann ist B eine \mathcal{C} -vollständige Menge. Der Begriff der *Vollständigkeit* fängt die härtesten Probleme einer Komplexitätsklasse bezüglich einer Reduzierbarkeit ein. Das heißt, B ist in dem Sinn für \mathcal{C} vollständig, dass die gesamte von der Klasse \mathcal{C} repräsentierte Berechnungskraft bzw. -komplexität bereits der Menge B innewohnt. Somit repräsentiert jedes für \mathcal{C} vollständige Problem die Klasse \mathcal{C} .

Nun werden eine der wichtigsten Reduzierbarkeiten, nämlich die polynomialzeit-beschränkte Many-one-Reduzierbarkeit, und die zugehörigen Begriffe der Härte und Vollständigkeit definiert.

Definition 3.35 (Many-one-Reduzierbarkeit und -Vollständigkeit). Für ein fest gewähltes Alphabet $\Sigma = \{0, 1\}$ seien A und B Mengen von Wörtern über Σ .

FP bezeichnet die Menge der polynomialzeit-berechenbaren totalen Funktionen von Σ^* in Σ^* . Sei \mathcal{C} eine beliebige Komplexitätsklasse.

1. Definiere die polynomialzeit-beschränkte Many-one-Reduzierbarkeit, bezeichnet mit \leq_m^p , wie folgt: $A \leq_m^p B$ genau dann, wenn es eine Funktion $f \in \text{FP}$ gibt, so dass für alle $x \in \Sigma^*$ gilt: $x \in A \iff f(x) \in B$.
2. Eine Menge B ist \leq_m^p -hart für \mathcal{C} , falls $A \leq_m^p B$ für jede Menge $A \in \mathcal{C}$ gilt.
3. Eine Menge B ist \leq_m^p -vollständig für \mathcal{C} , falls $B \leq_m^p$ -hart für \mathcal{C} ist und $B \in \mathcal{C}$.
4. \mathcal{C} heißt abgeschlossen unter der \leq_m^p -Reduzierbarkeit (kurz \leq_m^p -abgeschlossen), falls für je zwei Mengen A und B aus $A \leq_m^p B$ und $B \in \mathcal{C}$ folgt, dass A in \mathcal{C} ist.

Der Ausdruck „many-one“ oben bezieht sich auf die Tatsache, dass eine Reduktion $f \in \text{FP}$, die $A \leq_m^p B$ bezeugt, im Allgemeinen viele verschiedene Wörter auf ein und dasselbe Wort abbilden kann. Aus der obigen Definition ergeben sich sofort die im folgenden Lemma angegebenen Eigenschaften. Der Beweis von Lemma 3.36 wird dem Leser als Übung 3.11 überlassen.

- Lemma 3.36.**
1. Aus $A \leq_m^p B$ folgt $\overline{A} \leq_m^p \overline{B}$, doch im Allgemeinen gilt $A \leq_m^p \overline{A}$ nicht.
 2. Die Relation \leq_m^p ist sowohl reflexiv als auch transitiv, aber nicht antisymmetrisch.
 3. P, NP und PSPACE sind \leq_m^p -abgeschlossen.
 4. Gilt $A \leq_m^p B$ und ist $A \leq_m^p$ -hart für eine Komplexitätsklasse \mathcal{C} , so ist auch $B \leq_m^p$ -hart für \mathcal{C} .
 5. Seien \mathcal{C} und \mathcal{D} beliebige Komplexitätsklassen. Ist $\mathcal{C} \leq_m^p$ -abgeschlossen und ist $B \leq_m^p$ -vollständig für \mathcal{D} , so gilt $\mathcal{D} \subseteq \mathcal{C}$ genau dann, wenn B in \mathcal{C} ist. Ist B insbesondere NP-vollständig, so gilt $P = NP$ genau dann, wenn B in P ist.

Die dritte Aussage von Lemma 3.36 gilt natürlich nicht nur für die drei erwähnten Klassen. In der Tat sind die meisten „vernünftigen“ Komplexitätsklassen oberhalb von P unter \leq_m^p -Reduktionen abgeschlossen. Der \leq_m^p -Abschluss einer Klasse \mathcal{C} bedeutet, dass obere Schranken im Sinne von \mathcal{C} bezüglich der \leq_m^p -Reduzierbarkeit nach unten vererbt werden, wohingegen die vierte Aussage dieses Lemmas sagt, dass untere Schranken im Sinne von \mathcal{C} bezüglich \leq_m^p nach oben vererbt werden. Die letzte Aussage von Lemma 3.36 schließlich ist ganz entscheidend, da sie den Kollaps bzw. die Separation zweier Komplexitätsklassen mit der scheinbar viel einfacheren Frage verknüpft, ob ein einzelnes Problem aus der einen Klasse zur anderen gehört. Diese Eigenschaft, so einfach ihr Beweis auch sein mag, ist eine wesentliche Voraussetzung für die Nützlichkeit, Bedeutung und Schönheit der Theorie der NP-Vollständigkeit.

Der Begriff der \leq_m^p -Reduzierbarkeit ist für die Komplexitätsklassen L, NL und P nicht sinnvoll: Er ist einfach zu „grob“, um die Probleme in einer dieser Klassen hinsichtlich ihrer Berechnungskomplexität unterscheiden zu können. Insbesondere ist nämlich *jede* nichttriviale Menge in jeder dieser Klassen trivialerweise \leq_m^p -vollständig für die Klasse. Eine Menge B heißt nichttrivial, falls $\emptyset \neq B \neq \Sigma^*$ gilt. Lemma 3.37 illustriert diese Eigenschaft für den Fall der Klasse P.

Lemma 3.37. *Für jede nichttriviale Menge $B \in P$ (d.h., $\emptyset \neq B \neq \Sigma^*$) und für jede Menge $A \in P$ gilt $A \leq_m^p B$. Somit ist jede nichttriviale Menge in P \leq_m^p -vollständig für P.*

Beweis. Da B nichttrivial ist, können wir zwei Wörter wählen, für die $b_1 \in B$ und $b_2 \notin B$ gilt. Definiere die \leq_m^p -Reduktion f durch

$$f(x) = \begin{cases} b_1 & \text{falls } x \in A \\ b_2 & \text{falls } x \notin A. \end{cases}$$

Da A in P ist, ist die Funktion f in FP. Nach Definition von f gilt für jedes $x \in \Sigma^*$: $x \in A$ genau dann, wenn $f(x) \in B$. Folglich bezeugt f , dass $A \leq_m^p B$ gilt. Da A eine beliebige Menge in P ist, ist $B \leq_m^p$ -vollständig für P. \square

Angesichts von Lemma 3.37 ist eine feinere Reduzierbarkeit als \leq_m^p für P und kleinere Klassen erforderlich. Definition 3.38 unten führt die logarithmisch raumbeschränkte Many-one-Reduzierbarkeit ein, \leq_m^{\log} , und die zugehörigen Begriffe der \leq_m^{\log} -Härte und \leq_m^{\log} -Vollständigkeit.

Zu beachten ist, dass die \leq_m^{\log} -Reduzierbarkeit immer noch zu grob für die Klasse L ist, aus demselben Grund, weshalb \leq_m^p zu grob für P ist; siehe Lemma 3.37. Um Probleme in L und solche in noch kleineren Klassen sinnvoll voneinander unterscheiden zu können, müssen daher Reduzierbarkeiten verwendet werden, die noch feiner als \leq_m^{\log} sind. Solche Reduzierbarkeiten jedoch, die gewöhnlich mittels uniformer boolescher Schaltkreisfamilien definiert sind, werden in diesem Buch nicht betrachtet; der interessierte Leser wird stattdessen auf Vollmers Buch [Vol99] verwiesen. Für NL und P ist die \leq_m^{\log} -Reduzierbarkeit geeignet, und auch für andere Klassen wie NP stellt die \leq_m^{\log} -Reduzierbarkeit Härte- und Vollständigkeitsbegriffe bereit, die sich nur leicht von denen unterscheiden, die die \leq_m^p -Reduzierbarkeit liefert.

Definition 3.38 (Log-Space-Many-one-Reduzierbarkeit und -Vollständigkeit).

Sei $\Sigma = \{0, 1\}$ ein fest gewähltes Alphabet, und seien A und B Mengen von Wörtern über Σ . FL bezeichnet die Menge der in logarithmischem Raum berechenbaren totalen Funktionen von Σ^* in Σ^* . Definiere die Log-Space-Many-one-Reduzierbarkeit, bezeichnet mit \leq_m^{\log} , wie folgt: $A \leq_m^{\log} B$ genau dann, wenn es eine Funktion $f \in \text{FL}$ gibt, so dass für alle $x \in \Sigma^*$ gilt: $x \in A \iff f(x) \in B$.

Die Begriffe der \leq_m^{\log} -Härte, der \leq_m^{\log} -Vollständigkeit und des \leq_m^{\log} -Abschlusses für jede Komplexitätsklasse \mathcal{C} sind analog wie in Definition 3.35 definiert.

Genau wie die \leq_m^p -Reduzierbarkeit ist die \leq_m^{\log} -Reduzierbarkeit natürlich eine reflexive Relation. Satz 3.39 unten stellt eine andere grundlegende Eigenschaft der \leq_m^{\log} -Reduzierbarkeit heraus, die anders als die Reflexivität keineswegs trivialerweise gilt: die Transitivität.

Satz 3.39. Die \leq_m^{\log} -Reduzierbarkeit ist eine transitive Relation.

Beweis. Seien A , B und C Mengen, so dass $A \leq_m^{\log} B$ mittels einer Reduktion $f \in \text{FL}$ und $B \leq_m^{\log} C$ mittels einer Reduktion $g \in \text{FL}$ gilt. Sei F eine DTM, die f in logarithmischem Raum berechnet, und sei G eine DTM, die g in logarithmischem Raum berechnet. Um die Transitivität von \leq_m^{\log} zu beweisen, ist $A \leq_m^{\log} C$ mittels einer Reduktion $h \in \text{FL}$ zu zeigen. Das heißt, es ist eine DTM H anzugeben, die h in logarithmischem Raum berechnet, so dass für jedes $x \in \Sigma^*$ gilt: $x \in A$ genau dann, wenn $h(x) \in C$.

Wie oben erwähnt ist der Nachweis der Transitivität der \leq_m^{\log} -Reduzierbarkeit keineswegs eine triviale Angelegenheit. Um den Grund dafür einzusehen, ist es nützlich, zunächst den naiven Ansatz zur Definition einer DTM H zu betrachten, die eine Reduktion h für $A \leq_m^{\log} C$ berechnen soll, und dann zu sehen, weshalb dieser naive Ansatz misslingt. Der Einfachheit halber nehmen wir an, dass F und G jeweils ein Eingabeband (von dem nur gelesen werden kann), ein Arbeitsband (auf dem gelesen und geschrieben werden kann) sowie ein Ausgabeband (auf dem nur geschrieben werden kann) besitzen. Beim naiven Zugang würde man H so definieren, dass sie das Eingabeband von F als ihr eigenes Eingabeband und das Ausgabeband von G als ihr eigenes Ausgabeband und dass sie die folgenden drei Bänder als ihre eigenen Arbeitsbänder verwendet:

- AB 1: das Arbeitsband von F ,
- AB 2: das Ausgabeband von F , welches im Sinne von $h(x) = g(f(x))$ mit dem Eingabeband von G identifiziert wird, und
- AB 3: das Arbeitsband von G .

Definiere H bei Eingabe x als die „Komposition“ von F und G bei derselben Eingabe, um $h(x) = g(f(x))$ zu berechnen. Das heißt, $H(x)$ führt F und G hintereinander aus, beginnt also mit der Simulation von $F(x)$, wobei sie den auf AB 1 verfügbaren Platz $\log|x|$ verwendet und den Wert $f(x)$ auf AB 2 schreibt. Dann simuliert H die Berechnung von $G(f(x))$ auf AB 3 und schreibt den Wert $h(x) = g(f(x))$ auf ihr Ausgabeband. Das Problem mit diesem naiven Versuch – und der Grund für sein

Misslingen – liegt natürlich darin, dass die Länge des Ausgabewerts $f(x)$ auf AB 2 angesichts des bekannten Resultats $\text{DSPACE}(s) \subseteq \text{DTIME}(2^{\text{lin}(s)})$ (siehe Satz 3.6) nicht logarithmisch in $|x|$ beschränkt ist. Tatsächlich gibt es eine Konstante c , so dass $|f(x)| \leq 2^{c \cdot \log|x|} = |x|^c$ polynomiell in $|x|$ sein kann.

Um dieses Problem des naiven Zugangs zu überwinden, erinnern wir uns daran, dass man die Komplexitätsressource Raum wiederverwenden kann. Die Idee ist simpel: Anstatt den vollständigen Wert von $f(x)$ während der Simulation zu speichern, merkt sich H nur ein Bit von $f(x)$ auf AB 2, nämlich das aktuell von G in der Simulation gelesene Bit. Zu diesem Zweck benötigt H zwei weitere Arbeitsbänder, AB 4 und AB 5. Genauer gesagt enthält AB 4 den Binärwert der Position i , die aktuell vom Eingabekopf von G gelesen wird. Da $i \leq |x|^c$ gilt, ist dies im Raum $\lceil c \log|x| \rceil$ möglich. Der Zweck von AB 5 wird unten erläutert.

H arbeitet bei Eingabe x wie folgt. Anfangs enthält AB 4 den Binärwert eins, da der Eingabekopf von G zu Beginn das am weitesten links stehende Symbol des Eingabewortes von G liest. Alle anderen Arbeitsbänder sind leer. $H(x)$ beginnt nun mit der Simulation der Berechnung von $F(x)$ auf AB 1, bis das erste Ausgabebit von $f(x)$ auf AB 2 geschrieben wird. Die Simulation von $F(x)$ vorübergehend unterbrechend, simuliert H nun die Berechnung von $G(f(x))$ auf AB 3, bis G das nächste Eingabebit lesen muss. Die Simulation von $G(f(x))$ auf AB 3 vorübergehend unterbrechend, aktualisiert H nun den Inhalt von AB 4 entsprechend, löscht das erste Ausgabebit von $f(x)$ auf AB 2, setzt die Simulation von $F(x)$ auf AB 1 fort und wechselt in dieser Weise weiter zwischen beiden Simulationen hin und her.

Die allgemeine Situation ist die folgende: AB 2 enthält das i -te Bit von $f(x)$, und AB 4 enthält die Zahl i in Binärdarstellung. Angenommen, die Simulation von $G(f(x))$ ist gerade unterbrochen worden, weil H nun entweder das $(i-1)$ -te oder das $(i+1)$ -te Bit von $f(x)$ benötigt. Betrachte die folgenden beiden Fälle.

Fall 1: H benötigt das $(i+1)$ -te Bit von $f(x)$. Dann setzt H die Simulation von $F(x)$ auf AB 1 an der Stelle fort, an der sie zuvor unterbrochen wurde, bis das $(i+1)$ -te Bit von $f(x)$ auf AB 2 geschrieben wird. H erhöht den aktuellen Wert auf AB 4 um eins, so dass dieses nun $i+1$ in Binärdarstellung enthält, und setzt die Simulation von $G(f(x))$ fort.

Fall 2: H benötigt das $(i-1)$ -te Bit von $f(x)$. Da das $(i-1)$ -te Bit von $f(x)$ nicht mehr auf dem AB 2 vorliegt, muss die Berechnung von $F(x)$ ganz von Anfang an neu gestartet werden, wobei derselbe Raum auf AB 1 wiederverwendet wird. Ein zweites Arbeitsband, AB 5, wird hierfür gebraucht, um die Anzahl der Bits von $f(x)$ bis zum $(i-1)$ -ten Bit zu zählen. Anfangs enthält AB 5 den Wert null.

Schritt 1: H vermindert den Wert auf AB 4, welches zu diesem Zeitpunkt die Zahl $i > 1$ in Binärdarstellung enthält, um eins, so dass dort nun $i-1$ steht, und sie löscht auf AB 2 das i -te Ausgabebit von $f(x)$;

Schritt 2: H simuliert die Berechnung von $F(x)$ auf AB 1. Immer wenn F versucht, ein Ausgabebit von $f(x)$ auf AB 2 zu schreiben, geht H gemäß der folgenden zwei Unterfälle vor.

Unterfall 2.1: AB 4 enthält die Zahl $j \neq 0$ in Binärdarstellung. H vermindert den Wert auf AB 4 um eins, so dass dort nun $j-1$ in Binärdar-

stellung steht, und sie erhöht AB 5 um eins, so dass dort nun $i - j$ in Binärdarstellung steht. Dann setzt H die Simulation von $F(x)$ fort, ohne auf AB 2 zu schreiben.

Unterfall 2.2: AB 4 enthält 0. Da Unterfall 2.1 ($i - 1$)-mal aufgetreten ist, enthält AB 5 zurzeit $i - 1$ in Binärdarstellung und F versucht gerade, das $(i - 1)$ -te Ausgabebit von $f(x)$ auf AB 2 zu schreiben. H schreibt nun dieses Bit auf AB 2, kopiert den Inhalt von AB 5 auf AB 4, unterbricht die Simulation von $F(x)$ und setzt die Simulation von $G(f(x))$ fort.

Da die Berechnung von $G(f(x))$ auf AB 3 in logarithmischem Raum machbar ist, berechnet $H(x)$ den Wert $h(x) = g(f(x))$ in logarithmischem Raum, also ist x genau dann in A , wenn $h(x)$ in C ist. Somit gilt $A \leq_m^{\log} C$ mittels $h \in \text{FL}$, und die Transitivität von \leq_m^{\log} ist gezeigt. \square

Satz 3.40 kann analog zum obigen Beweis gezeigt werden; siehe Übung 3.12(a).

Satz 3.40. L und NL sind \leq_m^{\log} -abgeschlossen.

Stimmen die Reduzierbarkeiten \leq_m^{\log} und \leq_m^P überein? Stimmen sie wenigstens auf P überein? Betrachtet man die Relationen \leq_m^{\log} und \leq_m^P als Mengen von Mengenpaaren, d.h., $\leq_m^{\log} = \{(A, B) | A \leq_m^{\log} B\}$ und $\leq_m^P = \{(A, B) | A \leq_m^P B\}$, dann impliziert die Inklusion $\text{FL} \subseteq \text{FP}$ sofort $\leq_m^{\log} \subseteq \leq_m^P$. Ob die umgekehrte Inklusion ebenfalls gilt oder nicht, ist ein offenes Problem. Das folgende Resultat sagt jedoch, dass wenn L und P verschieden sind, dann unterscheiden sich \leq_m^{\log} und \leq_m^P auf P. Die Bezeichnung $A \not\leq_m^{\log} B$ bedeutet, dass $A \leq_m^{\log} B$ nicht gilt.

Satz 3.41. Ist $L \neq P$, so gibt es Mengen A und B in P mit $A \leq_m^P B$, aber $A \not\leq_m^{\log} B$.

Beweis. Um die Kontraposition der Aussage des Satzes zu zeigen, nehmen wir an, dass $\leq_m^{\log} = \leq_m^P$ auf P gilt. Sei B eine nichttriviale Menge in L, d.h., $\emptyset \neq B \neq \Sigma^*$. Sei A eine beliebige Menge in P. Nach Lemma 3.37 gilt $A \leq_m^P B$. Da B in L ist und da L nach Satz 3.40 \leq_m^{\log} -abgeschlossen ist, gilt $A \in L$. Weil A eine beliebige Menge in P ist, folgt $P = L$. \square

3.5.2 NL-Vollständigkeit

In diesem Abschnitt wird die \leq_m^{\log} -Vollständigkeit zweier Probleme in NL gezeigt, des Grapherreichbarkeitsproblems (kurz GAP für „Graph Accessibility Problem“) und des Erfüllbarkeitsproblems für boolesche Formeln mit (höchstens) zwei Literalen pro Klausel (kurz 2-SAT für „2-Satisfiability“).

Das Grapherreichbarkeitsproblem wird hier für gerichtete Graphen definiert. G ist ein *gerichteter Graph*, falls $E(G) \subseteq V(G) \times V(G)$ gilt, wobei $V(G)$ die *Knotenmenge* von G und $E(G)$ die *Kantenmenge* von G bezeichnet. Das heißt, eine gerichtete Kante $e = (u, v)$ von u zu v in G ist ein geordnetes Paar von Knoten. Für je zwei

Knoten $u, v \in V(G)$ in einem gerichteten Graphen G ist ein *Pfad (der Länge k) von u zu v* eine Folge x_1, x_2, \dots, x_{k+1} von Knoten in G , so dass $u = x_1$ und $v = x_{k+1}$ gilt und für jedes i mit $1 \leq i \leq k$ ist $e_i = (x_i, x_{i+1})$ eine Kante von G .

Definition 3.42 (Grapherreichbarkeitsproblem). Definiere die Entscheidungsversion des Grapherreichbarkeitsproblems durch

$$\text{GAP} = \left\{ \langle G, s, t \rangle \mid \begin{array}{l} G \text{ ist ein gerichteter Graph mit } s, t \in V(G), \\ \text{und es gibt einen gerichteten Pfad von } s \text{ zu } t \text{ in } G \end{array} \right\}.$$

Satz 3.43. GAP ist \leq_m^{\log} -vollständig für NL.

Beweis. Um zu zeigen, dass GAP in NL ist, definiere eine NTM M , die GAP in logarithmischem Raum wie folgt akzeptiert. Sei $\langle G, x_1, x_N \rangle$ eine gegebene Eingabe, wobei G ein gerichteter Graph ist, der durch die Liste seiner Kanten dargestellt wird, und $V(G) = \{x_1, x_2, \dots, x_N\}$. Ausgehend von x_1 rät M nichtdeterministisch einen Pfad in G zu x_N . Zu diesem Zweck speichert M die Indizes der Knoten auf einem solchen Pfad binär, und sie merkt sich stets nur zwei aufeinander folgende Knoten. Genauer: Sind i und j die Binärzahlen, die aktuell auf dem Arbeitsband von M stehen, so liest M ihre Eingabe, um zu überprüfen, ob es eine Kante (x_i, x_j) in $E(G)$ gibt oder nicht. Wenn ja, dann akzeptiert M die Eingabe im Falle $j = N$. Ist (x_i, x_j) eine Kante in $E(G)$ und ist $j \neq N$, so löscht M die Binärzahl i auf ihrem Arbeitsband und rät einen neuen Knoten, schreibt seinen Index k auf ihr Arbeitsband und setzt diese Prozedur so fort. Wenn (x_i, x_j) jedoch keine Kante in $E(G)$ ist, dann lehnt M ab und hält auf diesem Berechnungspfad erfolglos.

Einen Pfad α von x_1 zu x_N gibt es in G genau dann, wenn es einen Berechnungspfad von $M(\langle G, x_1, x_N \rangle)$ gibt, auf dem α geraten wurde. Somit akzeptiert M die Eingabe $\langle G, x_1, x_N \rangle$ genau dann, wenn $\langle G, x_1, x_N \rangle$ in GAP ist. Weil M niemals mehr als zwei Knotenindizes gleichzeitig speichert und weil die Binärdarstellung eines jeden Knotenindex die Länge höchstens $\lceil \log N \rceil$ hat, werden nicht mehr als $\mathcal{O}(\log n)$ Bandzellen auf dem Arbeitsband von M benutzt, wenn n die Größe der (vernünftig codierten) Eingabe ist; also arbeitet M in logarithmischem Raum. Somit ist GAP in NL.

Um die NL-Härte von GAP zu beweisen, sei A eine beliebige Menge in NL und M eine NTM, die A in logarithmischem Raum akzeptiert. Ohne Beschränkung der Allgemeinheit sei angenommen, dass M ein Eingabe- und ein Arbeitsband hat. Wie im Beweis von Satz 3.29 nehmen wir weiter an, dass $\text{START}_M(x)$ die eindeutig bestimmte Startkonfiguration von M bei Eingabe x und dass ACCEPT_M die eindeutig bestimmte akzeptierende Konfiguration von M bei irgendeiner Eingabe ist. Definiere für ein Eingabewort der Länge n den Graphen $G_{M,n}$ aller potenziellen $(\log n)$ -raumbeschränkten Konfigurationen von M durch:

$$V(G_{M,n}) = \left\{ C \mid \begin{array}{l} C \text{ ist eine potenzielle Konfiguration von } M, \text{ für die der} \\ \text{Inhalt des Arbeitsbands von } M \text{ die Länge } \leq \lceil \log n \rceil \text{ hat} \end{array} \right\};$$

$$E(G_{M,n}) = \{(C_1, C_2) \mid C_1 \vdash_M C_2\},$$

wobei \vdash_M die unmittelbare Folgerelation von M bezeichnet, d.h., die Konfiguration C_2 kann von der Konfiguration C_1 innerhalb eines Schrittes von M erreicht werden. Unter einer „potenziellen $(\log n)$ -raumbeschränkten Konfiguration“ von M verstehen wir dabei nicht nur eine Konfiguration im Berechnungsbaum von $M(x)$ für eine spezifische Eingabe x , sondern gemeint ist eine *jede* syntaktisch korrekte Konfiguration von M bei irgendeiner Eingabe der Länge n , die nicht mehr als $\lceil \log n \rceil$ Symbole auf dem Arbeitsband hat, selbst wenn eine solche Konfiguration von $\text{START}_M(x)$ für ein x nicht erreichbar sein sollte. Natürlich ist der Berechnungsbaum von $M(x)$ für jedes feste Eingabewort $x \in \Sigma^*$ der Länge n ein induzierter Teilgraph von $G_{M,n}$.

Man überzeugt sich leicht davon, dass jeder Knoten von $G_{M,n}$ durch $\mathcal{O}(\log n)$ Symbole codiert werden kann:

- die binär dargestellte aktuelle Position des Eingabekopfes von M erfordert $\lceil \log n \rceil$ Symbole, denen ein Trennzeichen folgt,
- die aktuelle Inschrift des Arbeitsbands von M benötigt höchstens $\lceil \log n \rceil$ Symbole und
- $c = \lceil \log |Z| \rceil$ Symbole (wobei Z die Zustandsmenge von M ist), die den aktuellen Zustand von M darstellen, können in das Wort, das die aktuelle Inschrift des Arbeitsbands von M beschreibt, an der Stelle eingefügt werden, an der gerade der Kopf des Arbeitsbands steht, um dessen aktuelle Position anzuzeigen.

Somit genügen $c + 1 + 2\lceil \log n \rceil$ Bandzellen, um jede potenzielle $(\log n)$ -raumbeschränkte Konfiguration von M darzustellen.

Definiere die gesuchte \leq_m^{\log} -Reduktion f von A auf GAP für jedes Wort $x \in \Sigma^*$ der Länge n durch

$$f(x) = \langle G_{M,n}, \text{START}_M(x), \text{ACCEPT}_M \rangle.$$

Somit gilt für jedes $x \in \Sigma^*$:

$$\begin{aligned} x \in A &\iff M(x) \text{ hat einen akzeptierenden Berechnungspfad} \\ &\iff f(x) \in \text{GAP}. \end{aligned}$$

Dass f in FL ist, bleibt noch zu zeigen. Dies leistet der folgende deterministische Algorithmus, der f bei Eingabe x , $|x| = n$, in logarithmischem Raum berechnet:

Schritt 1: Lege den Raum $c + 1 + 2\lceil \log n \rceil$ aus.

Schritt 2: Erzeuge systematisch, eine nach der anderen in lexikographischer Ordnung, sämtliche potenziellen $(\log n)$ -raumbeschränkten Konfigurationen von M , deren Darstellung nicht mehr als die markierten $c + 1 + 2\lceil \log n \rceil$ Bandzellen benötigt.

Schritt 3: Für jede solche erzeugte Konfiguration C wird

1. geprüft, ob C syntaktisch korrekt ist;
2. C zur Liste der Knoten von $G_{M,n}$ hinzugefügt;
3. ausgehend von C ein Schritt von M simuliert, um die potenziellen $(\log n)$ -raumbeschränkten Konfigurationen C_1 und C_2 von M mit $C \vdash_M C_1$ und $C \vdash_M C_2$ zu erzeugen;

4. für $i \in \{1, 2\}$ das Paar (C, C_i) zur Liste der Kanten von $G_{M,n}$ hinzugefügt.

Schritt 4: Wenn die Konstruktion des Graphen $G_{M,n}$ abgeschlossen ist, erfolgt die Ausgabe:

$$\langle G_{M,n}, \text{START}_M(x), \text{ACCEPT}_M \rangle.$$

Somit bezeugt $f \in \text{FL}$, dass $A \leq_m^{\log} \text{GAP}$ gilt. \square

Nun wenden wir uns einigen Varianten des Erfüllbarkeitsproblems zu. Boolesche Formeln, Wahrheitswertbelegungen und der Begriff der Erfüllbarkeit einer booleschen Formel wurden in den Definitionen 2.23 und 2.24 in Abschnitt 2.3 eingeführt.

Definition 3.44 (Erfüllbarkeitsproblem). Definiere die Entscheidungsversion des Erfüllbarkeitsproblems durch

$$\text{SAT} = \{\varphi \mid \varphi \text{ ist eine erfüllbare boolesche Formel in KNF}\}.$$

Definiere für jedes feste $k \geq 1$ die folgende Einschränkung des Erfüllbarkeitsproblems:

$$k\text{-SAT} = \{\varphi \mid \varphi \text{ ist eine erfüllbare boolesche Formel in } k\text{-KNF}\}.$$

Um zu verstehen, weshalb das Erfüllbarkeitsproblem oben nur für boolesche Formeln in KNF definiert wurde, siehe Übung 2.10.

Satz 3.45. 2-SAT ist \leq_m^{\log} -vollständig für NL.

Beweis. Um zu zeigen, dass 2-SAT in NL ist, sei eine beliebige boolesche Formel $\varphi(x_1, x_2, \dots, x_n)$ in 2-KNF gegeben; ohne Beschränkung der Allgemeinheit darf angenommen werden, dass φ genau zwei Literale pro Klausel hat. Konstruiere einen gerichteten Graphen G_φ aus φ wie folgt:

$$V(G_\varphi) = \{x_1, x_2, \dots, x_n\} \cup \{\neg x_1, \neg x_2, \dots, \neg x_n\};$$

$$E(G_\varphi) = \{(\alpha, \beta) \mid (\neg \alpha \vee \beta) \text{ oder } (\beta \vee \neg \alpha) \text{ ist eine Klausel in } \varphi\}.$$

Die Kanten in G_φ stellen die Implikationen in φ und ihre Kontrapositionen dar. Das heißt, wenn φ eine Klausel der Form $(\neg \alpha \vee \beta)$ für Literale α und β enthält, dann ist diese Klausel semantisch äquivalent zu der Implikation $(\alpha \implies \beta)$, welche wiederum semantisch äquivalent zu ihrer Kontraposition $(\neg \beta \implies \neg \alpha)$ ist, welche wiederum semantisch äquivalent zu $(\beta \vee \neg \alpha)$ ist. Nach Definition besitzt jede Kante in G_φ die folgende Symmetrie: Ist (α, β) eine Kante von G_φ , dann ebenso $(\neg \beta, \neg \alpha)$.

Betrachte beispielsweise die boolesche Formel φ mit vier Klauseln:

$$\varphi(x_1, x_2, x_3) = (\neg x_1 \vee x_3) \wedge (\neg x_3 \vee \neg x_1) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee x_1).$$

Abbildung 3.1 zeigt den entsprechenden Graphen G_φ mit acht Kanten, wobei zwei Kanten jeweils einer Klausel in φ entsprechen. Wie man leicht sieht, ist φ in diesem Beispiel nicht erfüllbar. Andererseits, wenn wir die Knoten von G_φ mit den Literalen in φ identifizieren, dann gibt es einen Knoten im Graphen G_φ aus Abbildung 3.1, für den ein Pfad von diesem Knoten zu seiner Negation und zurück zum Knoten existiert. Betrachte zum Beispiel den Zyklus $(x_1, x_3, \neg x_1, x_2, x_1)$. Wie das folgende Lemma zeigt, ist diese Eigenschaft kein Zufall.

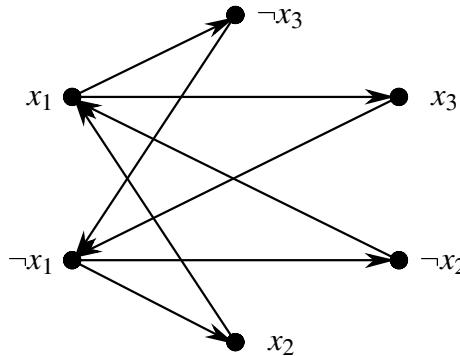


Abb. 3.1. 2-SAT ist NL-vollständig: Graph G_φ konstruiert aus der booleschen Formel φ

Lemma 3.46. Sei φ eine boolesche Formel in 2-KNF. Dann ist φ genau dann nicht erfüllbar, wenn es einen Knoten x in G_φ gibt, für den ein Pfad von x zu $\neg x$ und ein Pfad von $\neg x$ zurück zu x in G_φ existiert.

Beweis von Lemma 3.46. Um die Richtung von rechts nach links zu beweisen, sei für einen Widerspruch angenommen, dass es in G_φ einen Pfad von x zu $\neg x$ und einen Pfad von $\neg x$ zurück zu x gibt, für ein $x \in V(G_\varphi)$, und dennoch gibt es eine Wahrheitswertbelegung t , die φ erfüllt. Betrachte die folgenden beiden Fälle.

Fall 1: $t(x) = 1$. Dann ist $t(\neg x) = 0$. Also existiert eine Kante (α, β) auf dem Pfad von x zu $\neg x$, so dass $t(\alpha) = 1$ und $t(\beta) = 0$ ist. Nach Konstruktion von G_φ und weil (α, β) eine Kante ist, gibt es eine Klausel der Form $(\neg \alpha \vee \beta)$ oder $(\beta \vee \neg \alpha)$ in φ , die von t nicht erfüllt wird. Folglich erfüllt t die Formel φ nicht, ein Widerspruch.

Fall 2: $t(x) = 0$. Dann ist $t(\neg x) = 1$. Mit einem zu Fall 1 analogen Argument gibt es eine Kante (α, β) auf dem Pfad von $\neg x$ zu x , der einer Klausel der Form $(\neg \alpha \vee \beta)$ oder $(\beta \vee \neg \alpha)$ in φ entspricht, die von t nicht erfüllt wird. Folglich erfüllt t die Formel φ nicht, was wieder ein Widerspruch ist.

Für die Richtung von links nach rechts wird die Kontraposition gezeigt: Wenn es für keinen Knoten x in G_φ einen Pfad der Form $(x, \dots, \neg x, \dots, x)$ gibt, dann ist φ erfüllbar. Tatsächlich kann nämlich unter der eben genannten Voraussetzung eine erfüllende Belegung t für φ wie folgt konstruiert werden:

while (es gibt Variablen in φ , die unter t noch keinen Wahrheitswert haben) {

Schritt 1: Wähle die erste solche Variable x in φ und betrachte den entsprechenden Knoten x in G_φ . Nach Voraussetzung gibt es keinen Pfad von x zu $\neg x$ und zurück von $\neg x$ zu x .

Schritt 2: Für jeden Knoten γ in G_φ , der von x erreichbar ist (inklusive x selbst), setze $t(\gamma) = 1$ und $t(\neg \gamma) = 0$, wobei wieder Knoten von G_φ mit den Literalen in φ identifiziert werden.

}

Zu zeigen ist, dass Schritt 2 in der obigen `while`-Schleife wohldefiniert ist. Dazu sehen wir uns die folgenden beiden Möglichkeiten an, was schief gehen könnte, und überzeugen uns davon, dass keiner dieser schlechten Fälle wirklich auftreten kann.

Fall 1: Angenommen, es gibt Pfade von x zu sowohl γ als auch $\neg\gamma$ in Schritt 2 der obigen `while`-Schleife. In diesem Fall wären wir in Schwierigkeiten, denn t müsste sowohl γ als auch $\neg\gamma$ den Wahrheitswert 1 zuweisen, und sowohl $\neg\gamma$ als auch γ den Wahrheitswert 0.

Dieser Fall kann jedoch wegen der Symmetrie in der Konstruktion von G_φ nicht auftreten. Denn unter der obigen Annahme muss es auch einen Pfad sowohl von γ als auch von $\neg\gamma$ zu $\neg x$ geben, da wegen der Symmetrie ein Pfad von x zu γ einen solchen von $\neg\gamma$ zu $\neg x$ impliziert, und somit gibt es einen Pfad von x zu $\neg\gamma$ und von $\neg\gamma$ zu $\neg x$. Mit einem ähnlichen Argument muss es einen Pfad von x zu $\neg x$ über γ geben. Wieder impliziert wegen der Symmetrie ein Pfad von x zu $\neg x$ einen solchen von $\neg x$ zu x , was ein Widerspruch zur Wahl von x ist.

Fall 2: Angenommen, ein Knoten γ , der von x erreichbar ist, erhält den Wahrheitswert 1, obwohl ihm von t bereits der Wahrheitswert 0 in einem früheren Durchlauf der `while`-Schleife zugewiesen wurde. Diese Annahme bedeutet also, dass es einen Pfad von x zu γ gibt und $t(\gamma) = 0$ gilt. Dann wären wir wieder in Schwierigkeiten.

Dieser Fall kann jedoch auch nicht auftreten, da in diesem früheren Durchlauf der `while`-Schleife x bereits den Wert 0 erhalten hätte: $t(\gamma) = 0$ impliziert $t(\neg\gamma) = 1$, was wiederum $t(\neg x) = 1$ zur Folge hat, wegen des Pfades von $\neg\gamma$ zu $\neg x$. Somit ist $t(x) = 0$, was wieder ein Widerspruch zur Wahl von x ist.

Nach Annahme gibt es für keinen Knoten x einen Pfad von x zu $\neg x$ und von $\neg x$ zurück zu x . Folglich terminiert die obige Prozedur zur Konstruktion einer Belegung t , denn in jedem Durchlauf der `while`-Schleife wird mindestens einer Variablen ein Wahrheitswert zugewiesen.

Es bleibt zu zeigen, dass t tatsächlich φ erfüllt. Jedes Mal, wenn einem Literal der Wahrheitswert 1 zugewiesen wurde, erhält auch der Nachfolger des Knoten, der diesem Literal entspricht, den Wert 1. Analog sieht man, dass jedem Vorgänger eines Knoten, dessen entsprechendes Literal den Wahrheitswert 0 erhält, ebenfalls der Wert 0 zugewiesen wird. Somit ergibt keine Klausel von φ eine Implikation der Form $(1 \implies 0)$ unter der Belegung t , also erfüllt t die Formel φ . \square Lemma 3.46

Um den Beweis, dass 2-SAT in NL ist, abzuschließen, wird mittels Lemma 3.46 nun gezeigt, dass das *Komplement* von 2-SAT in NL liegt. Weil nach Korollar 3.34 NL = coNL gilt, folgt so die Zugehörigkeit von 2-SAT zu NL.

Bei Eingabe φ geht eine NL-Maschine für das Komplement von 2-SAT folgendermaßen vor: Rate eine Variable x und einen Pfad in G_φ von x zu $\neg x$ und von $\neg x$ zurück zu x , und akzeptiere die Eingabe φ genau dann, wenn ein solcher Pfad existiert. Nach Lemma 3.46 akzeptiert dieser nichtdeterministische Algorithmus φ genau dann, wenn φ nicht erfüllbar ist.

Genau wie im Beweis von Satz 3.43 werden beim Raten eines solchen Pfades in G_φ stets nur zwei aufeinander folgende Knotenindizes gleichzeitig gespeichert.

Daher arbeitet dieser nichtdeterministische Algorithmus in logarithmischem Raum. Es folgt, dass 2-SAT in NL liegt.

Um nun zu zeigen, dass 2-SAT NL-hart ist, wird eine \leq_m^{\log} -Reduktion vom Komplement einer eingeschränkten Version des Grapherreichbarkeitsproblems auf 2-SAT angegeben. Zunächst wird gezeigt, dass diese eingeschränkte Version von GAP ebenfalls NL-vollständig ist.

Ein *Zyklus* (oder *Kreis*) in einem Graphen G ist ein Pfad der Form $x_1, x_2, \dots, x_n, x_1$. Ein Graph heißt *azyklisch*, falls er keinen Zyklus enthält. Definiere das auf azyklische Graphen eingeschränkte Grapherreichbarkeitsproblem wie folgt: Gegeben ein azyklischer Graph G und zwei Knoten s und t in G , gibt es einen Pfad von s zu t ? Bezeichne mit $\text{GAP}_{\text{acyclic}}$ diese Einschränkung des Grapherreichbarkeitsproblems. Es ist nicht schwer zu zeigen, dass $\text{GAP}_{\text{acyclic}} \leq_m^{\log}$ -vollständig für NL ist. Dazu modifiziert man den Beweis von Satz 3.43 geeignet, wie es im Beweis des folgenden Lemmas erklärt wird.

Lemma 3.47. $\text{GAP}_{\text{acyclic}}$ ist \leq_m^{\log} -vollständig für NL.

Beweis von Lemma 3.47. In Abwandlung des Beweises von Satz 3.43, in dem der Graph $G_{M,n}$ aller potenziellen $(\log n)$ -raumbeschränkten Konfigurationen der gegebenen NTM M bei Eingaben der Länge n betrachtet wurde, verwenden wir nun den induzierten Teilgraphen $M(x)$ von $G_{M,n}$ für eine spezifische Eingabe x der Länge n . Außerdem modifizieren wir die Definition des Graphen $G_{M,n}$ in diesem Beweis dadurch, dass wir uns nun für jede Konfiguration C von $M(x)$ auch die Anzahl t der Schritte von $\text{START}_M(x)$ bis C merken. Definiere also den Graphen $\hat{G}_{M,x}$ für eine gegebene NL-Machine M und ein festes Eingabewort x durch:

$$V(\hat{G}_{M,x}) = \left\{ \langle C, t \rangle \middle| \begin{array}{l} C \text{ ist eine Konfiguration von } M(x), \text{ für die der Inhalt des} \\ \text{Arbeitsbands von } M \text{ die Länge } \leq \lceil \log n \rceil \text{ hat, und genau} \\ t \geq 0 \text{ Takte von } M \text{ überführen } \text{START}_M(x) \text{ in } C \end{array} \right\};$$

$$E(\hat{G}_{M,x}) = \{(\langle C_1, t_1 \rangle, \langle C_2, t_2 \rangle) \mid C_1 \vdash_M C_2 \text{ und } t_2 = t_1 + 1\}.$$

Wie man leicht sieht, hat dieser Graph $\hat{G}_{M,x}$ keine (gerichteten) Zyklen. Ausgehend von $\langle \text{START}_M(x), 0 \rangle$ kann $\hat{G}_{M,x}$ deterministisch durch Breitensuche in logarithmischem Raum konstruiert werden. Sei p eine Polynomialeitzeituhr, so dass $M(x)$ genau $p(|x|)$ Takte auf jedem Berechnungspfad und für jede Eingabe x ausführt; vergleiche Korollar 3.28. Dann akzeptiert M ihre Eingabe x genau dann, wenn $\langle \hat{G}_{M,x}, \langle \text{START}_M(x), 0 \rangle, \langle \text{ACCEPT}_M, p(|x|) \rangle \rangle$ in $\text{GAP}_{\text{acyclic}}$ ist. Dies ist eine \leq_m^{\log} -Reduktion von einem beliebigen NL-Problem auf $\text{GAP}_{\text{acyclic}}$, was die NL-Härte von $\text{GAP}_{\text{acyclic}}$ beweist. Es ist nicht schwer zu zeigen, dass $\text{GAP}_{\text{acyclic}}$ in NL liegt (siehe Übung 3.12(b)); somit ist $\text{GAP}_{\text{acyclic}}$ NL-vollständig. \square Lemma 3.47

Der Beweis von Satz 3.45 kann nun mittels Satz 3.39 abgeschlossen werden, nach welchem die \leq_m^{\log} -Reduzierbarkeit transitiv ist. Genauer zeigt das folgende Lemma die Reduktion $\text{GAP}_{\text{acyclic}} \leq_m^{\log}$ 2-SAT. Da nach Korollar 3.34 $\text{NL} = \text{coNL}$ gilt, ist $\text{GAP}_{\text{acyclic}}$ auch für coNL \leq_m^{\log} -vollständig. Wie schon im obigen Beweis von

$\text{2-SAT} \in \text{NL}$ genügt es daher für den Beweis von $\text{GAP}_{\text{acyclic}} \leq_m^{\log} \text{2-SAT}$ zu zeigen, dass das Komplement von $\text{GAP}_{\text{acyclic}}$, welches nach Korollar 3.34 und Lemma 3.47 NL-vollständig ist, auf $\text{2-SAT} \leq_m^{\log}$ -reduziert werden kann.

Lemma 3.48. $\text{GAP}_{\text{acyclic}} \leq_m^{\log} \text{2-SAT}$.

Beweis von Lemma 3.48. Nach den obigen Bemerkungen genügt es, eine Reduktion $f \in \text{FL}$ zu definieren, so dass für jede Eingabe x gilt:

$$x \notin \text{GAP}_{\text{acyclic}} \iff f(x) \in \text{2-SAT}. \quad (3.7)$$

Sei $x = \langle G, v_1, v_n \rangle$ eine beliebige Eingabeinstanz von $\text{GAP}_{\text{acyclic}}$, wobei G ein zyklischer gerichteter Graph mit $V(G) = \{v_1, v_2, \dots, v_n\}$ ist. Definiere die Reduktion f durch

$$\begin{aligned} f(x) &= \varphi_x(v_1, v_2, \dots, v_n) \\ &= v_1 \wedge \neg v_n \wedge \bigwedge_{(v_i, v_j) \in E(G)} (\neg v_i \vee v_j). \end{aligned}$$

Dass f in FL ist, kann man sich leicht überlegen. Die Intuition hinter dieser Reduktion ist, dass ein Literal v_i in φ_x genau dann wahr sein soll, wenn der Knoten v_i in G von v_1 aus erreichbar ist. Insbesondere bedeutet die Erfüllung der ersten Klausel in φ_x , dass v_1 von v_1 aus erreichbar ist. Ebenso bedeutet die Erfüllung der zweiten Klausel in φ_x , dass v_n nicht von v_1 aus erreichbar ist. Schließlich bedeutet die Erfüllung aller übrigen Klauseln in φ_x , dass wenn (v_i, v_j) in $E(G)$ ist und v_i von v_1 aus erreichbar ist, dann ist auch v_j von v_1 aus erreichbar. Folglich gilt:

$$\begin{aligned} x \notin \text{GAP}_{\text{acyclic}} &\iff \text{es gibt keinen Pfad von } v_1 \text{ zu } v_n \text{ in } G \\ &\iff f(x) = \varphi_x(v_1, v_2, \dots, v_n) \in \text{2-SAT}. \end{aligned}$$

Somit ist (3.7) erfüllt und das Lemma bewiesen. \square Lemma 3.48

Da 2-SAT nach dem im Beweis von Lemma 3.46 präsentierten Algorithmus in NL liegt und da 2-SAT nach den Lemmata 3.47 und 3.48 NL -hart ist, ist der Beweis des Satzes abgeschlossen. \square Satz 3.45

3.5.3 NP-Vollständigkeit

Historisch war das in Definition 3.44 eingeführte Erfüllbarkeitsproblem das erste natürliche Problem, dessen NP-Vollständigkeit bewiesen wurde. Dieses bahnbrechende Resultat geht auf Cook [Coo71] und, unabhängig, auf Levin [Lev73] zurück. Tausende weitere NP-Vollständigkeitsresultate folgten; siehe Garey und Johnsons Einführung in die Theorie der NP-Vollständigkeit [GJ79], die Hunderte solcher Probleme auflistet.

In diesem Abschnitt werden der Einfachheit halber sämtliche NP-Vollständigkeitsresultate nur bezüglich der \leq_m^p -Reduzierbarkeit gezeigt. Mit ein wenig mehr Anstrengung können diese NP-Vollständigkeitsresultate ebenso bezüglich der etwas stärkeren \leq_m^{\log} -Reduzierbarkeit bewiesen werden.

Satz 3.49 (Cook). SAT ist NP-vollständig.

Beweis. Um zu sehen, dass SAT in NP liegt, betrachte die folgende NTM M , die SAT wie folgt in Polynomialzeit akzeptiert. Rate für eine gegebene boolesche Formel $\varphi(x_1, x_2, \dots, x_n)$ nichtdeterministisch eine Wahrheitswertbelegung t der Variablen x_1, x_2, \dots, x_n , bewerte für jede geratene Belegung t die Formel φ gemäß t und akzeptiere genau dann, wenn $\varphi(t) = 1$.

Zum Beweis der NP-Härte sei A eine beliebige Menge in NP, und sei $M = (\Sigma, \Gamma, Z, \delta, \square, s_0, F)$ eine NTM, die A in Polynomialzeit akzeptiert; siehe Kapitel 2 für die Bedeutung der einzelnen Komponenten in dem Tupel, das M beschreibt. Ohne Beschränkung der Allgemeinheit darf angenommen werden, dass M nur ein Band hat, das zugleich als Eingabe- und als Arbeitsband dient, und dass M bei jeder Eingabe der Länge n genau $p(n) \geq n$ Schritte für ein $p \in \text{IPol}$ ausführt. Das Ziel ist es, eine Reduktion $f \in \text{FP}$ zu definieren, so dass für jedes x gilt:

$$x \in A \iff f(x) = F_x \in \text{SAT}, \quad (3.8)$$

wobei F_x eine boolesche Formel ist, deren Struktur und deren Variablen unten beschrieben werden.

...	□	...	□	x_1	x_2	...	x_n	□	...	□	...
...	$-p(n)$...	-1	0	1	...	$n-1$	n	...	$p(n)$...

Abb. 3.2. Nummerierung der Bandzellen der NTM M bei Eingabe x in der Cook-Reduktion

Sei $x = x_1 x_2 \dots x_n$ das gegebene Eingabewort, mit $x_i \in \Sigma$ für jedes i . Da M in der Zeit $p(n)$ arbeitet, kann sich der Kopf nicht weiter als um $p(n)$ Bandzellen nach links oder nach rechts bewegen. Nummeriere die relevanten Bandzellen von $-p(n)$ bis $p(n)$. Abbildung 3.2 zeigt das Band von M bei der Startkonfiguration: Die Eingabesymbole $x_1 x_2 \dots x_n$ stehen auf den Bandzellen 0 bis $n-1$, der Kopf liest gerade die Bandzelle mit Nummer 0 und der Startzustand ist s_0 .

Nun wird die boolesche Formel F_x so konstruiert, dass (3.8) erfüllt ist. Die booleschen Variablen von F_x , der Bereich ihrer Indizes und ihre beabsichtigte Bedeutung sind in Tabelle 3.4 angegeben. Intuitiv gibt es drei Typen von Variablen:

- $\text{state}_{t,s}$ stellt den Zustand s von M in Takt t dar;
- $\text{head}_{t,i}$ stellt die Nummer i der Bandzelle dar, die der Kopf von M in Takt t liest;
- $\text{tape}_{t,i,a}$ stellt das Symbol $a \in \Gamma$ dar, das in Takt t in der Bandzelle Nummer i von M steht.

F_x hat die Form

$$F_x = S \wedge \ddot{U}_1 \wedge \ddot{U}_2 \wedge E \wedge K,$$

wobei diese Teilformeln von F_x die folgende Bedeutung haben:

- S wird erfüllt \iff die Rechnung von $M(x)$ startet korrekt;

Tabelle 3.4. Die booleschen Variablen von F_x und ihre Bedeutung in der Cook-Reduktion

Variable	Indexbereich	Bedeutung
$\text{state}_{t,s}$	$t \in \{0, 1, \dots, p(n)\}$ $s \in Z$	$\text{state}_{t,s}$ ist wahr \iff M ist im Takt t in Zustand s
$\text{head}_{t,i}$	$t \in \{0, 1, \dots, p(n)\}$ $i \in \{-p(n), \dots, p(n)\}$	$\text{head}_{t,i}$ ist wahr \iff M liest im Takt t die Bandzelle Nr. i
$\text{tape}_{t,i,a}$	$t \in \{0, 1, \dots, p(n)\}$ $i \in \{-p(n), \dots, p(n)\}$ $a \in \Gamma$	$\text{tape}_{t,i,a}$ ist wahr \iff Symbol a steht im Takt t in Bandzelle Nr. i

- \ddot{U}_1 wird erfüllt \iff die *Übergänge* der Rechnung von $M(x)$ von jedem Takt t zum Takt $t + 1$ sind in den Bandzellen korrekt, auf denen der Kopf steht und deren Inhalt daher geändert werden kann;
- \ddot{U}_2 wird erfüllt \iff die *Übergänge* der Rechnung von $M(x)$ von jedem Takt t zum Takt $t + 1$ sind in den Bandzellen korrekt, auf denen der Kopf nicht steht und deren Inhalt daher unverändert bleibt;
- E wird erfüllt \iff die Rechnung von $M(x)$ *endet* korrekt (d.h., M akzeptiert x);
- K wird erfüllt \iff die allgemeine *Korrektheit* der Rechnung von $M(x)$ ist gegeben, d.h.:
 - in jedem Takt t der Rechnung von $M(x)$ steht der Kopf von M auf *genau* einem Feld und ist M in *genau* einem Zustand, und
 - in jedem Takt t und für jede Bandposition i gibt es *genau* ein Symbol $a \in \Gamma$ in diesem Feld des Bandes.

Um diese Teilformeln von F_x formal zu beschreiben, sei $Z = \{s_0, s_1, \dots, s_k\}$ die Zustandsmenge und $\Gamma = \{\square, a_1, a_2, \dots, a_\ell\}$ das Arbeitsalphabet von M . Das Eingabealphabet Σ ist dabei in Γ enthalten. Definiere zunächst die Teilformel K von F_x durch

$$K = \bigwedge_{0 \leq t \leq p(n)} [D_1(\text{state}_{t,s_0}, \text{state}_{t,s_1}, \dots, \text{state}_{t,s_k}) \wedge \\ D_2(\text{head}_{t,-p(n)}, \text{head}_{t,-p(n)+1}, \dots, \text{head}_{t,p(n)}) \wedge \quad (3.9) \\ \bigwedge_{-p(n) \leq i \leq p(n)} D_3(\text{tape}_{t,i,\square}, \text{tape}_{t,i,a_1}, \dots, \text{tape}_{t,i,a_\ell})],$$

wobei Lemma 3.50 unten die Struktur der drei Teilformeln D_i von K in (3.9) genauer spezifiziert. Insbesondere gilt für $i \in \{1, 2, 3\}$: (a) D_i ist genau dann wahr, wenn *genau* eine der Variablen von D_i wahr ist, und (b) die Größe von D_i – und folglich auch die Größe von K – ist polynomiell in n .

Lemma 3.50. Für jedes m gibt es eine boolesche Formel D mit m Variablen, so dass gilt:

1. $D(v_1, v_2, \dots, v_m)$ ist wahr \iff genau eine Variable v_i ist wahr;
2. die Größe von D (d.h., die Zahl der Vorkommen von Variablen in D) ist in $\mathcal{O}(m^2)$.

Beweis von Lemma 3.50. Definiere für ein festes $m \geq 1$ die Formel

$$D(v_1, v_2, \dots, v_m) = \underbrace{\left(\bigvee_{i=1}^m v_i \right)}_{D_{\geq}} \wedge \underbrace{\left(\bigwedge_{j=1}^{m-1} \bigwedge_{k=j+1}^m \neg(v_j \wedge v_k) \right)}_{D_{\leq}}.$$

Die Teilformeln D_{\geq} und D_{\leq} von D erfüllen dabei die folgenden Äquivalenzen:

$$D_{\geq}(v_1, v_2, \dots, v_m) \text{ ist wahr} \iff \text{mindestens eine Variable } v_i \text{ ist wahr}; \quad (3.10)$$

$$D_{\leq}(v_1, v_2, \dots, v_m) \text{ ist wahr} \iff \text{höchstens eine Variable } v_i \text{ ist wahr}. \quad (3.11)$$

Die Äquivalenz (3.10) ist offensichtlich. Dass auch die Äquivalenz (3.11) gilt, sieht man an der folgenden Struktur der Teilformel D_{\leq} :

$$\begin{aligned} D_{\leq}(v_1, v_2, \dots, v_m) &= (\neg v_1 \vee \neg v_2) \wedge (\neg v_1 \vee \neg v_3) \wedge \dots \wedge (\neg v_1 \vee \neg v_m) \\ &\quad \wedge (\neg v_2 \vee \neg v_3) \wedge \dots \wedge (\neg v_2 \vee \neg v_m) \\ &\quad \vdots \\ &\quad \wedge (\neg v_{m-1} \vee \neg v_m). \end{aligned}$$

Die Äquivalenzen (3.10) und (3.11) zusammen ergeben, dass $D(v_1, v_2, \dots, v_m)$ genau dann wahr ist, wenn genau eine Variable v_i erfüllt ist. Offenbar ist die Größe von D in $\mathcal{O}(m^2)$. \square Lemma 3.50

Der Beweis von Satz 3.49 wird mit der Definition der Teilformel S von F_x fortgesetzt:

$$S = \text{state}_{0,s_0} \wedge \text{head}_{0,0} \wedge \bigwedge_{i=-p(n)}^{-1} \text{tape}_{0,i,\square} \wedge \bigwedge_{i=0}^{n-1} \text{tape}_{0,i,x_{i+1}} \wedge \bigwedge_{i=n}^{p(n)} \text{tape}_{0,i,\square},$$

die für Takt $t = 0$ den korrekten Start der Berechnung $M(x)$ beschreibt (siehe Abbildung 3.2).

Als Nächstes wird die Teilformel \ddot{U}_1 von F_x definiert, die den korrekten Übergang von Takt t zu Takt $t + 1$ für diejenigen Bandzellen beschreibt, deren Inhalt vom Kopf von M geändert werden kann:

$$\begin{aligned} \ddot{U}_1 = \bigwedge_{t,s,i,a} & ((\text{state}_{t,s} \wedge \text{head}_{t,i} \wedge \text{tape}_{t,i,a}) \implies \\ & \bigvee_{\substack{\hat{s} \in Z, \hat{a} \in \Gamma, y \in \{-1, 0, 1\} \\ \text{mit } (\hat{s}, \hat{a}, y) \in \delta(s, a)}} (\text{state}_{t+1,\hat{s}} \wedge \text{head}_{t+1,i+y} \wedge \text{tape}_{t+1,i,\hat{a}})), \end{aligned}$$

wobei δ die Überführungsfunktion von M ist und $y \in \{-1, 0, 1\}$ die Kopfbewegung repräsentiert (nach links bzw. keine Bewegung bzw. nach rechts).

Nun wird die Teilformel \ddot{U}_2 von F_x definiert, die den korrekten Übergang von Takt t zu Takt $t + 1$ für die aktuell von M nicht gelesenen (also auch nicht modifizierbaren) Bandzellen beschreibt:

$$\ddot{U}_2 = \bigwedge_{t,i,a} ((\neg \text{head}_{t,i} \wedge \text{tape}_{t,i,a}) \implies \text{tape}_{t+1,i,a}).$$

Schließlich wird die Teilformel E von F_x definiert, die das korrekte Ende der Berechnung von $M(x)$ beschreibt und überprüft, ob M die Eingabe x akzeptiert oder nicht:

$$E = \bigvee_{s \in F} \text{state}_{p(n),s},$$

wobei F die Menge der akzeptierenden Endzustände von M ist.

Damit ist die Konstruktion der Reduktion f abgeschlossen. Analysiert man die Struktur der Formel $f(x) = F_x$ und wendet man Lemma 3.50 an, so kann man zeigen, dass f in FP ist; siehe Übung 3.13. Zu zeigen bleibt also die Äquivalenz (3.8): $x \in A$ genau dann, wenn die Formel $f(x) = F_x$ erfüllbar ist.

Sei $x \in A$. Dann gibt es einen akzeptierenden Berechnungspfad α von $M(x)$. Weist man nun jeder Variablen von F_x gemäß ihrer beabsichtigten Bedeutung aus Tabelle 3.4 einen α entsprechenden Wahrheitswert zu, dann erfüllt diese Belegung jede Teilformel von F_x , also auch F_x selbst. Somit ist $F_x \in \text{SAT}$.

Sei nun $F_x \in \text{SAT}$. Dann gibt es eine Belegung τ , die F_x erfüllt. Die Variablen $\text{state}_{t,s}$, $\text{head}_{t,i}$ und $\text{tape}_{t,i,a}$ von F_x können gemäß τ (und der beabsichtigten Bedeutung von Variablen aus Tabelle 3.4) als eine Folge $K_0, K_1, \dots, K_{p(n)}$ von Konfigurationen von $M(x)$ entlang eines Berechnungspfades interpretiert werden. Da $F_x = S \wedge \ddot{U}_1 \wedge \ddot{U}_2 \wedge E \wedge K$ unter τ wahr ist, muss τ jede dieser Teilformeln von F_x erfüllen. Somit gilt:

- K_0 ist die Startkonfiguration von $M(x)$, weil τ die Teilformel S erfüllt,
- $K_{t-1} \vdash_M K_t$ für jedes t mit $1 \leq t \leq p(n)$, weil τ die Teilformeln \ddot{U}_1 , \ddot{U}_2 und K erfüllt, und
- $K_{p(n)}$ ist eine akzeptierende Endkonfiguration von $M(x)$, weil τ die Teilformel E erfüllt.

Folglich ist $x \in A$, und die Äquivalenz (3.8) ist gezeigt. □ Satz 3.49

Nach Satz 3.45 ist die Einschränkung 2-SAT des Erfüllbarkeitsproblems NL-vollständig. Im Gegensatz dazu zeigt Satz 3.51 unten, dass 3-SAT genau wie das allgemeine Erfüllbarkeitsproblem NP-vollständig ist. Die Bedeutung dieses Resultats liegt u.a. in der Tatsache, dass 3-SAT ein sehr geeigneter Ausgangspunkt für weitere NP-Vollständigkeitsresultate ist.

Satz 3.51. 3-SAT ist NP-vollständig.

Beweis. Dass 3-SAT in NP ist, folgt unmittelbar daraus, dass das allgemeinere Problem SAT in NP liegt. Um nun $\text{SAT} \leq_m^P 3\text{-SAT}$ zu zeigen, wird eine Reduktion f definiert, die jede gegebene boolesche Formel φ in eine boolesche Formel ψ in 3-KNF transformiert, so dass gilt:

$$\varphi \text{ ist erfüllbar} \iff \psi \text{ ist erfüllbar}. \quad (3.12)$$

Sei $\varphi(x_1, x_2, \dots, x_n) = C_1 \wedge C_2 \wedge \dots \wedge C_m$, wobei C_j die j -te Klausel von φ ist. Die Formel ψ wird wie folgt aus φ konstruiert. Die Variablen von ψ sind die Variablen x_1, x_2, \dots, x_n von φ und zusätzlich die Variablen $y_1^j, y_2^j, \dots, y_{h_j}^j$, deren Anzahl h_j von der Form der Klausel C_j abhängt. Definiere $\psi = D_1 \wedge D_2 \wedge \dots \wedge D_m$, wobei jede Klausel D_j von ψ wie folgt aus der Klausel C_j von φ konstruiert wird. Betrachte die j -te Klausel von φ , die die Form $C_j = (z_1 \vee z_2 \vee \dots \vee z_k)$ habe, wobei jedes z_i ein Literal über $\{x_1, x_2, \dots, x_n\}$ ist. Unterscheide die folgenden vier Fälle.

Fall 1: $k = 1$. Dann ist

$$D_j = (z_1 \vee y_1^j \vee y_2^j) \wedge (z_1 \vee \neg y_1^j \vee y_2^j) \wedge (z_1 \vee y_1^j \vee \neg y_2^j) \wedge (z_1 \vee \neg y_1^j \vee \neg y_2^j).$$

Fall 2: $k = 2$. Dann ist $D_j = (z_1 \vee z_2 \vee y_1^j) \wedge (z_1 \vee z_2 \vee \neg y_1^j)$.

Fall 3: $k = 3$. Dann ist $D_j = C_j = (z_1 \vee z_2 \vee z_3)$.

Fall 4: $k \geq 4$. Dann ist

$$\begin{aligned} D_j = & (z_1 \vee z_2 \vee y_1^j) \wedge (\neg y_1^j \vee z_3 \vee y_2^j) \wedge (\neg y_2^j \vee z_4 \vee y_3^j) \wedge \dots \wedge \\ & (\neg y_{k-4}^j \vee z_{k-2} \vee y_{k-3}^j) \wedge (\neg y_{k-3}^j \vee z_{k-1} \vee z_k). \end{aligned}$$

Offenbar kann die Reduktion f in Polynomialzeit berechnet werden. Es bleibt zu zeigen, dass (3.12) gilt. Für die Implikation von links nach rechts sei t eine Wahrheitswertbelegung der Variablen x_1, x_2, \dots, x_n von φ , die φ erfüllt. Erweitere t zu einer Belegung t' der Variablen von ψ wie folgt. Da für $i \neq j$ die Klauseln D_i und D_j bezüglich der y -Variablen disjunkt sind, genügt es, alle Klauseln von ψ separat zu betrachten.

Betrachte also die Klausel D_j für ein beliebiges j . In den Fällen 1 bis 3 oben erfüllt t bereits D_j , so dass t beliebig zu t' erweitert werden kann. Betrachte nun den Fall 4 oben. Sei z_i , $1 \leq i \leq k$, das erste Literal in C_j , für das $t(z_i) = 1$ gilt. Solch ein i muss existieren, weil t die Klausel C_j erfüllt. Ist $i \in \{1, 2\}$, so setze $t'(y_\ell^j) = 0$ für jedes ℓ mit $1 \leq \ell \leq k-3$. Ist $i \in \{k-1, k\}$, so setze $t'(y_\ell^j) = 1$ für jedes ℓ mit $1 \leq \ell \leq k-3$. Andernfalls setze

$$t'(y_\ell^j) = \begin{cases} 1 & \text{falls } 1 \leq \ell \leq i-2 \\ 0 & \text{falls } i-1 \leq \ell \leq k-3. \end{cases}$$

In jedem Fall erfüllt t' die Klausel D_j , für jedes j . Folglich erfüllt t' die Formel ψ .

Um umgekehrt die Implikation von rechts nach links in (3.12) zu zeigen, sei t' eine erfüllende Belegung für ψ . Sei t die Einschränkung von t' auf die Variablen x_1, x_2, \dots, x_n von φ . Folglich erfüllt t die Formel φ , womit (3.12) und somit der Satz bewiesen ist. \square

Der obige Beweis liefert nicht nur eine Reduktion vom NP-vollständigen Problem SAT auf dessen Einschränkung 3-SAT, welche nach Definition *höchstens* drei Literale pro Klausel erlaubt, sondern sogar auf die etwas stärkere Einschränkung von SAT, bei der pro Klausel *genau* drei Literale verlangt werden. Diese Eigenschaft wird beim Beweis weiterer NP-Vollständigkeitsresultate nützlich sein.

Nun werden NP-Vollständigkeitsresultate für einige wichtige Graphenprobleme vorgestellt. Alle diese Probleme sind für ungerichtete Graphen definiert, Graphen also, deren Kanten ungeordnete Knotenpaare sind. Dabei werden nur einfache Graphen betrachtet, welche keine Schleifen oder Mehrfachkanten haben. Zur Erinnerung: $V(G)$ bzw. $E(G)$ bezeichnet die Knoten- bzw. Kantenmenge eines Graphen G .

Zunächst werden einige graphentheoretische Begriffe und drei wohlbekannte Graphenprobleme definiert. Definition 2.49 in Abschnitt 2.4.3 stellte einige Grundlagen der Graphentheorie bereit.

Definition 3.52 (Clique, Unabhängige Menge und Knotenüberdeckung). Sei G ein ungerichteter Graph.

- Eine Clique von G ist eine Teilmenge $C \subseteq V(G)$, so dass für je zwei Knoten $x, y \in C$, $x \neq y$, $\{x, y\}$ eine Kante in $E(G)$ ist.
- Eine unabhängige Menge (englisch „independent set“) von G ist eine Teilmenge $I \subseteq V(G)$, so dass für je zwei Knoten $x, y \in I$, $x \neq y$, $\{x, y\}$ keine Kante in $E(G)$ ist.
- Eine Knotenüberdeckung (englisch „vertex cover“) von G ist eine Teilmenge $C \subseteq V(G)$, so dass $\{x, y\} \cap C \neq \emptyset$ für jede Kante $\{x, y\}$ in $E(G)$ gilt.

Definiere nun die Entscheidungsversionen des Clique-Problems, des Unabhängigen-Menge-Problems sowie des Knotenüberdeckungsproblems durch:

$$\text{Clique} = \{\langle G, k \rangle \mid \text{Graph } G \text{ hat eine Clique der Größe mindestens } k\};$$

$$\text{IS} = \{\langle G, k \rangle \mid \text{Graph } G \text{ hat eine unabhängige Menge der Größe mindestens } k\};$$

$$\text{VC} = \{\langle G, k \rangle \mid \text{Graph } G \text{ hat eine Knotenüberdeckung der Größe höchstens } k\}.$$

Lemma 3.53. Für jeden Graphen G und für jede Teilmenge $U \subseteq V(G)$ sind die folgenden Eigenschaften paarweise äquivalent:

1. U ist eine Knotenüberdeckung von G .
2. $\overline{U} = V(G) - U$ ist eine unabhängige Menge von G .
3. $\overline{U} = V(G) - U$ ist eine Clique des co-Graphen von G , welcher definiert ist als der Graph mit der Knotenmenge $V(G)$ und der Kantenmenge

$$\{\{u, v\} \mid u, v \in V(G) \text{ und } \{u, v\} \notin E(G)\}.$$

Der Beweis von Lemma 3.53 wird dem Leser als Übung 3.14(a) überlassen.

Satz 3.54. Clique, IS und VC sind NP-vollständig.

Beweis. Man sieht leicht, das jedes der Probleme Clique, IS und VC zu NP gehört; siehe Übung 3.14(c). Aus Lemma 3.53 folgt, dass diese drei Probleme paarweise \leq_m^p -äquivalent sind, d.h., für je zwei Probleme A und B , gewählt unter Clique, IS und VC, gilt $A \leq_m^p B$ und $B \leq_m^p A$; siehe Übung 3.14(b). Folglich genügt es, 3-SAT \leq_m^p IS zu beweisen. Sei $\varphi(x_1, x_2, \dots, x_n) = C_1 \wedge C_2 \wedge \dots \wedge C_m$ eine gegebene boolesche Formel in KNF mit genau drei Literalen pro Klausel. Für jedes i

mit $1 \leq i \leq m$ sei die i -te Klausel gegeben durch $C_i = (z_{i,1} \vee z_{i,2} \vee z_{i,3})$, wobei jedes $z_{i,j} \in \{x_1, x_2, \dots, x_n\} \cup \{\neg x_1, \neg x_2, \dots, \neg x_n\}$ ein Literal ist.

Die Reduktion f bildet φ auf das Paar $\langle G, m \rangle$ ab, wobei G der Graph mit der Knotenmenge $V(G) = \{z_{i,j} \mid 1 \leq i \leq m \text{ und } 1 \leq j \leq 3\}$ und der Kantenmenge

$$\begin{aligned} E(G) = & \{\{z_{i,j}, z_{i,k}\} \mid 1 \leq i \leq m \text{ und } 1 \leq j, k \leq 3 \text{ und } j \neq k\} \cup \\ & \{\{z_{i,j}, z_{r,s}\} \mid i \neq r \text{ und } z_{i,j} = \neg z_{r,s}\} \end{aligned}$$

ist. Das heißt, jedes Vorkommen eines Literals in einer Klausel von φ wird durch einen Knoten von G repräsentiert, die Klauseln von φ entsprechen den Dreiecken in G , und je zwei Knoten in verschiedenen Dreiecken sind genau dann durch eine Kante verbunden, wenn der eine Knoten ein Literal und der andere dessen Negation repräsentiert.

Betrachte beispielsweise den Graphen G in Abbildung 3.3, der aus der Formel

$$\varphi(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3)$$

konstruiert wurde.

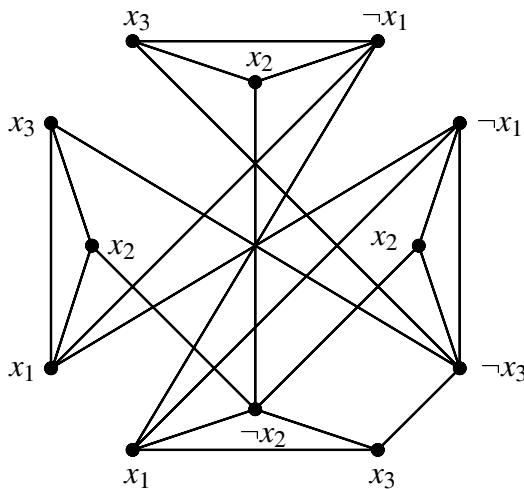


Abb. 3.3. Beispielgraph G in der Reduktion $3\text{-SAT} \leq_m^p \text{IS}$

Offensichtlich ist f in FP. Aus der Konstruktion folgt:

- $$\begin{aligned} \varphi \in 3\text{-SAT} &\iff \text{es gibt eine Belegung } t, \text{ die } \varphi \text{ erfüllt} \\ &\iff \text{jede Klausel } C_i \text{ hat ein Literal } z_{i,j_i} \text{ mit } t(z_{i,j_i}) = 1 \\ &\iff \text{es gibt eine Folge von Literalen } z_{1,j_1}, z_{2,j_2}, \dots, z_{m,j_m}, \\ &\quad \text{so dass } z_{i,j_i} \neq \neg z_{k,j_k} \text{ für } i, k \in \{1, 2, \dots, m\} \text{ mit } i \neq k \\ &\iff \text{es gibt eine Folge von Literalen } z_{1,j_1}, z_{2,j_2}, \dots, z_{m,j_m}, \text{ so dass} \\ &\quad \{z_{1,j_1}, z_{2,j_2}, \dots, z_{m,j_m}\} \text{ eine unabhängige Menge der Größe } m \\ &\quad \text{in } G \text{ ist.} \end{aligned}$$

Da G eine unabhängige Menge der Größe mindestens m genau dann hat, wenn φ erfüllbar ist, wird $3\text{-SAT} \leq_m^P \text{IS}$ durch die Reduktion f bezeugt. \square

Als Nächstes beschäftigen wir uns mit dem Graphfärbbarkeitsproblem und dem Domatische-Zahl-Problem, die in einer Vielzahl von Anwendungen auftreten und eng mit Planungs- und Zerlegungsproblemen (englisch „*scheduling and partitioning problems*“) für Graphen verwandt sind. Das Graphfärbbarkeitsproblem zum Beispiel fragt nach der kleinsten Anzahl von Farben, die nötig ist, die Knoten eines gegebenen Graphen so zu färben, dass je zwei benachbarte Knoten verschieden gefärbt sind. Diese Zahl nennt man die chromatische Zahl des Graphen.

Definition 3.55 (Chromatische Zahl und das Färbbarkeitsproblem). Sei G ein ungerichteter Graph. Eine Färbung von G ist eine Abbildung von $V(G)$ in die positiven natürlichen Zahlen, welche die Farben repräsentieren. Eine Färbung ψ von G heißt legal, falls für je zwei Knoten x und y in $V(G)$ gilt: Aus $\{x, y\} \in E(G)$ folgt $\psi(x) \neq \psi(y)$.

Die chromatische Zahl von G , bezeichnet mit $\chi(G)$, ist die kleinste Zahl von Farben, die nötig ist, um G legal zu färben. Für eine jede feste Konstante $k \geq 1$ heißt G k -färbar, falls es eine legale Färbung von G mit nicht mehr als k Farben gibt, und die Entscheidungsversion des k -Färbbarkeitsproblems ist definiert durch

$$k\text{-Colorability} = \{G \mid G \text{ ist ein Graph mit } \chi(G) \leq k\}.$$

Es ist bekannt, dass 2-Colorability in Polynomialzeit gelöst werden kann; siehe Übung 3.16. Im Gegensatz dazu zeigt Satz 3.56 unten, dass 3-Colorability NP-vollständig ist.

Satz 3.56. 3-Colorability ist NP-vollständig.

Beweis. Die Zugehörigkeit von 3-Colorability zu NP ist leicht zu sehen: Rate nichtdeterministisch eine Zerlegung der Knotenmenge des gegebenen Graphen in drei Farbklassen und verifizierte für jede geratene Zerlegung deterministisch, dass sie eine legale Färbung darstellt.

Die folgende Reduktion von 3-SAT auf 3-Colorability wird die NP-Härte beweisen. Sei $\varphi(x_1, x_2, \dots, x_n) = C_1 \wedge C_2 \wedge \dots \wedge C_m$ eine gegebene 3-SAT -Instanz mit genau drei Literalen pro Klausel. Definiere eine Reduktion f , die φ auf den folgendermaßen konstruierten Graphen G abbildet. Die Knotenmenge von G ist definiert durch

$$V(G) = \{v_1, v_2, v_3\} \cup \{x_i, \bar{x}_i \mid 1 \leq i \leq n\} \cup \{y_{j,k} \mid 1 \leq j \leq m \text{ und } 1 \leq k \leq 6\},$$

wobei die x_i und \bar{x}_i Knoten sind, die die Literale x_i bzw. ihre Negationen $\neg x_i$ repräsentieren. Die Kantenmenge von G ist definiert durch

$$\begin{aligned} E(G) = & \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_1, v_3\}\} \cup \{\{x_i, \bar{x}_i\} \mid 1 \leq i \leq n\} \\ & \cup \{\{v_3, x_i\}, \{v_3, \bar{x}_i\} \mid 1 \leq i \leq n\} \\ & \cup \{\{a_j, y_{j,1}\}, \{b_j, y_{j,2}\}, \{c_j, y_{j,3}\} \mid 1 \leq j \leq m\} \\ & \cup \{\{v_2, y_{j,6}\} \mid 1 \leq j \leq m\} \\ & \cup \{\{y_{j,1}, y_{j,2}\}, \{y_{j,1}, y_{j,4}\}, \{y_{j,2}, y_{j,4}\} \mid 1 \leq j \leq m\} \\ & \cup \{\{y_{j,3}, y_{j,5}\}, \{y_{j,3}, y_{j,6}\}, \{y_{j,5}, y_{j,6}\} \mid 1 \leq j \leq m\} \\ & \cup \{\{y_{j,4}, y_{j,5}\} \mid 1 \leq j \leq m\}, \end{aligned}$$

wobei $a_j, b_j, c_j \in \bigcup_{1 \leq i \leq n} \{x_i, \bar{x}_i\}$ Knoten sind, die die Literale repräsentieren, die in der Klausel $C_j = (a_j \vee b_j \vee c_j)$ vorkommen.

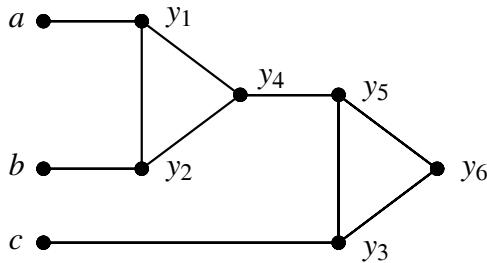


Abb. 3.4. Graph H in der Reduktion $3\text{-SAT} \leq_m^P 3\text{-Colorability}$

Der Graph H in Abbildung 3.4 ist der entscheidende Baustein dieser Reduktion, welche m disjunkte Kopien von H (mit entsprechenden Indizes) benutzt, eine für jede Klausel C_j von φ . Die Korrektheit der Reduktion folgt aus den folgenden beiden Eigenschaften des Graphen H :

Jede Färbung der Knoten a, b und c , die mindestens einen von a, b und c mit der Farbe 1 färbt, kann zu einer legalen 3-Färbung von H erweitert werden, die den Knoten y_6 mit der Farbe 1 färbt. (3.13)

Ist ψ eine legale 3-Färbung von H mit $\psi(a) = \psi(b) = \psi(c) = i$, dann gilt $\psi(y_6) = i$. (3.14)

Der Beweis von (3.13) und (3.14) wird dem Leser als Übung 3.15 überlassen. Aus (3.13) und (3.14) folgt, dass φ genau dann erfüllbar ist, wenn G 3-färbbar ist. Somit ist 3-Colorability NP-vollständig. \square

Nun werden der Begriff einer dominierenden Menge in einem Graphen und das Domatische-Zahl-Problem definiert. Dieses Problem, das z.B. im Gebiet der

Computer-Netzwerke auftritt, besteht darin, einen gegebenen Graphen in eine maximale Anzahl disjunkter dominierender Mengen zu zerlegen.

Definition 3.57 (Domatische-Zahl-Problem). Sei G ein ungerichteter Graph. Eine dominierende Menge von G ist eine Teilmenge $D \subseteq V(G)$, so dass es für jeden Knoten $u \in V(G) - D$ einen Knoten $v \in D$ mit $\{u, v\} \in E(G)$ gibt. Die domatische Zahl von G , bezeichnet mit $\delta(G)$, ist die maximale Anzahl disjunkter dominierender Mengen von G . Definiere die Entscheidungsversion des Domatische-Zahl-Problems durch

$$\text{DNP} = \{\langle G, k \rangle \mid G \text{ ist ein Graph und } k \text{ ist eine positive ganze Zahl mit } \delta(G) \geq k\}.$$

Bezeichnet $\min-deg(G)$ den Minimalgrad des Graphen G (also die kleinste Zahl von Kanten, die aus einem der Knoten von G auslaufen), so gilt stets $\delta(G) \leq \min-deg(G) + 1$, wie man sich leicht überlegt.

Satz 3.58. DNP ist NP-vollständig.

Beweis. Um $\text{3-Colorability} \leq_m^p \text{DNP}$ zu zeigen, wird nun eine Reduktion f konstruiert, die einen gegebenen Graphen G auf ein Paar $\langle H, 3 \rangle = f(G)$ abbildet, wobei H ein Graph ist, der die Implikationen (3.15) und (3.16) erfüllt:

$$G \in \text{3-Colorability} \implies \delta(H) = 3; \quad (3.15)$$

$$G \notin \text{3-Colorability} \implies \delta(H) = 2. \quad (3.16)$$

Da in Polynomialzeit getestet werden kann, ob ein gegebener Graph zweifärbbar ist (siehe Übung 3.16), dürfen wir ohne Beschränkung der Allgemeinheit annehmen, dass G nicht zweifärbbar ist. Auch nehmen wir an, dass G keine isolierten Knoten hat. Die domatische Zahl eines jeden Graphen ohne isolierte Knoten ist stets mindestens zwei; siehe [GJ79]. Der Graph H wird aus G konstruiert, indem $\|E(G)\|$ neue Knoten geschaffen werden, die jeweils auf jede Kante von G gesetzt werden (so dass aus jeder alten Kante zwei neue werden), und indem neue Kanten hinzugefügt werden, so dass die ursprünglichen Knoten von G eine Clique bilden. In dieser Weise induziert jede Kante von G ein Dreieck in H , und jedes Paar von nicht benachbarten Knoten in G ist in H durch eine Kante verbunden.

Sei $V(G) = \{v_1, v_2, \dots, v_n\}$. Definiere die Knoten- und Kantenmenge von H durch:

$$\begin{aligned} V(H) &= V(G) \cup \{u_{i,j} \mid \{v_i, v_j\} \in E(G)\}; \\ E(H) &= \{\{v_i, u_{i,j}\} \mid \{v_i, v_j\} \in E(G)\} \cup \{\{u_{i,j}, v_j\} \mid \{v_i, v_j\} \in E(G)\} \\ &\quad \cup \{\{v_i, v_j\} \mid 1 \leq i, j \leq n \text{ und } i \neq j\}. \end{aligned}$$

Abbildung 3.5 zeigt an einem Beispiel, wie der Graph H aus dem gegebenen Graphen G konstruiert wird. G ist in diesem Beispiel dreifärbbar; man könnte etwa die Knoten v_1 und v_4 rot, den Knoten v_2 blau und die Knoten v_3 und v_5 grün färben. Dann hat H die domatische Zahl 3, und die Knotenmenge $V(H)$ kann etwa in die folgenden drei dominierenden Mengen zerlegt werden:

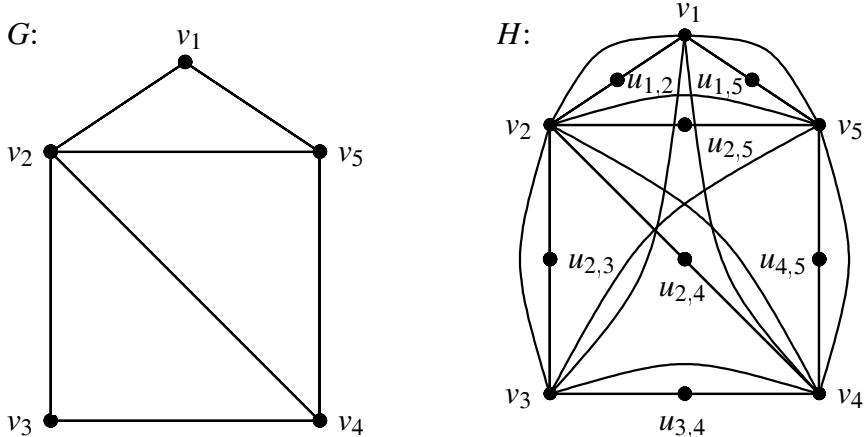


Abb. 3.5. Beispielgraphen G und H in der Reduktion $3\text{-Colorability} \leq_m^p \text{DNP}$

$$\begin{aligned} R &= \{v_1, v_4, u_{2,3}, u_{2,5}\}, \\ B &= \{v_2, u_{1,5}, u_{3,4}, u_{4,5}\}, \\ G &= \{v_3, v_5, u_{1,2}, u_{2,4}\}, \end{aligned}$$

was (3.15) für dieses Beispiel zeigt.

Nun beweisen wir die Implikationen (3.15) und (3.16) im Allgemeinen. Da nach Konstruktion $\min-deg(H) = 2$ gilt und da H keine isolierten Knoten hat, folgt aus der Ungleichung $\delta(H) \leq \min-deg(H) + 1$, dass $2 \leq \delta(H) \leq 3$ gilt.

Angenommen, G ist dreifärbbar. Seien C_1, C_2 und C_3 drei fest gewählte Farbklassen von G , d.h., $C_k = \{v_i \in V(G) \mid v_i \text{ hat die Farbe } k\}$ für $k \in \{1, 2, 3\}$. Zerlege $V(H)$ in $\hat{C}_k = C_k \cup \{u_{i,j} \mid \{v_i, v_j\} \in E(G) \text{ und } v_i \notin C_k \text{ und } v_j \notin C_k\}$ für $k \in \{1, 2, 3\}$. Da $\hat{C}_k \cap V(G) \neq \emptyset$ für jedes k gilt und weil $V(G)$ eine Clique in H induziert, wird $V(G)$ von jeder Menge \hat{C}_k in H dominiert. Außerdem enthält jedes Dreieck $\{v_i, u_{i,j}, v_j\}$ in H ein Element aus jedem \hat{C}_k , so dass eine jede Menge \hat{C}_k ebenso die Menge $\{u_{i,j} \mid \{v_i, v_j\} \in E(G)\}$ in H dominiert. Folglich gilt $\delta(H) = 3$, womit (3.15) bewiesen ist.

Umgekehrt sei angenommen, dass $\delta(H) = 3$ gilt. Seien \hat{C}_1, \hat{C}_2 und \hat{C}_3 drei dominierende Mengen, die $V(H)$ zerlegen. Färbe die Knoten in \hat{C}_k mit der Farbe k . Jedes Dreieck $\{v_i, u_{i,j}, v_j\}$ in H ist dreigefärbt, woraus folgt, dass dieselbe Färbung eingeschränkt auf die Knoten in $V(G)$ eine legale 3-Färbung von G ist; also ist G in 3-Colorability. Folglich gilt $\chi(G) = 3$ genau dann, wenn $\delta(H) = 3$ gilt. Da $2 \leq \delta(H) \leq 3$ gilt, folgt die Implikation (3.16). \square

Schließlich betrachten wir einige Matching-, Mengenüberdeckungs- und Rucksack-Probleme. Wir beginnen mit dem Matching-Problem.

Definition 3.59 (Zweidimensionales Matching-Problem). Ein vollständiger bipartiter Graph ist ein Graph mit $2n$ Knoten, dessen Knotenmenge in zwei disjunkte

Teilmengen der Größe n zerlegt werden kann, V_1 und V_2 , die beide unabhängige Mengen sind. Das heißt, weder die Knoten in V_1 noch die Knoten in V_2 sind durch Kanten verbunden; es kann lediglich Kanten geben, die Knoten von V_1 mit Knoten von V_2 verbinden.

Sei G ein vollständiger bipartiter Graph mit $V(G) = V_1 \cup V_2$ und $V_1 \cap V_2 = \emptyset$. Ein (perfektes) bipartites Matching von G ist eine Teilmenge $M \subseteq E(G)$ von n Kanten, so dass für je zwei verschiedene Kanten $\{v, w\}$ und $\{x, y\}$ in M gilt: $v \neq x$ und $w \neq y$. Das bipartite Matching-Problem (alias das zweidimensionale Matching-Problem) fragt, ob es ein bipartites Matching in einem gegebenen bipartiten Graphen gibt.

Beispiel 3.60 (Zweidimensionales Matching-Problem). Betrachte eine Menge V_{bride} von n Bräuten und eine Menge V_{groom} von n Bräutigamen, die die Knoten eines bipartiten Graphen G bilden. Die Knotenmenge von G ist also in V_{bride} und V_{groom} zerlegt, so dass $V(G) = V_{\text{bride}} \cup V_{\text{groom}}$ und $V_{\text{bride}} \cap V_{\text{groom}} = \emptyset$ gilt. Eine Kante zwischen zwei Knoten zeigt an, dass die entsprechenden Partner gewillt sind, einander zu heiraten. Dass es keine Kanten innerhalb von V_{bride} und keine Kanten innerhalb von V_{groom} gibt, lässt sich konventionell erklären.

Ein Matching kann man sich nun als eine Methode vorstellen, n Hochzeiten zwischen der Gruppe der n Bräute und der Gruppe der n Bräutigame so zu arrangieren, dass – in den Worten von Garey und Johnson [GJ79] – „Polygamie vermieden wird und alle einen akzeptablen Gatten bzw. eine akzeptable Gattin bekommen“.

Die linke Seite von Abbildung 3.6 gibt ein konkretes Beispiel mit vier Paaren (p_i, m_i) an, die sich gemäß der fettgedruckten Kanten paaren.

Die obige „Hochzeitsinterpretation“ bipartiter Matchings erklärt, weshalb dieses Problem manchmal als das „Heiratsproblem“ bezeichnet wird. Es ist bekannt, dass das Heiratsproblem in Polynomialzeit gelöst werden kann; siehe Hopcroft und Karp [HK73]. Im wirklichen Leben ist ihr Resultat ebenso gut anwendbar: *Heiraten ist leicht!*

Nun werden bipartite Graphen und Matchings auf drei Dimensionen verallgemeinert. Das entsprechende allgemeinere Problem wird als das *dreidimensionale Matching-Problem* (alias das *tripartite Matching-Problem*) bezeichnet.

Definition 3.61 (Dreidimensionales Matching-Problem). Sei $n > 0$ eine natürliche Zahl, seien U , V und W drei paarweise disjunkte Mengen der Größe n , und sei $R \subseteq U \times V \times W$ eine ternäre Relation, d.h., eine Menge von Tripeln (u, v, w) mit $u \in U$, $v \in V$ und $w \in W$.

Ein tripartites Matching von R ist eine Teilmenge $M \subseteq R$ der Größe n , so dass für je zwei verschiedene Tripel (u, v, w) und $(\hat{u}, \hat{v}, \hat{w})$ in M gilt: $u \neq \hat{u}$, $v \neq \hat{v}$ und $w \neq \hat{w}$. Das heißt, keine zwei Elemente eines tripartiten Matchings stimmen in irgendeiner Koordinate überein.

Definiere die Entscheidungsversion des dreidimensionalen Matching-Problems durch

$$3\text{-DM} = \left\{ \langle R, U, V, W \rangle \middle| \begin{array}{l} U, V \text{ und } W \text{ sind paarweise disjunkte, nichtleere} \\ \text{Mengen gleicher Größe, und } R \subseteq U \times V \times W \\ \text{ist eine ternäre Relation, die ein tripartites Matching} \\ \text{der Größe } \|U\| \text{ enthält} \end{array} \right\}.$$

Die folgende kleine Geschichte setzt Beispiel 3.60 fort und erweitert es.

Story 3.62 *Neun Monate sind vergangen. Eines Morgens sind unsere n glücklich verheirateten Paare auf ihrem Weg in den Kreißsaal. Ein paar Stunden harter Arbeit später sind n Babies auf der Welt, die sofort mit Schreien anfangen und die Komplexität im Leben ihrer Eltern beträchtlich erhöhen. Das fängt damit an, dass sie sich alle die Namensschilder abstrampeln, auf denen steht, zu welchem Elternpaar sie gehören, was ein gewaltiges Durcheinander im Kreißsaal verursacht. Schlimmer noch ist, dass jeder der frisch gebackenen Väter – vielleicht von der Aufregung des Augenblicks verwirrt oder von der Schönheit der anderen Frauen verführt – behauptet, er hätte diese junge Dame nie gesehen, die da störrisch darauf besteht, sie habe gerade sein Kind zur Welt gebracht. Stattdessen behauptet der treulose Geselle mit der anderen jungen Frau gleich rechts daneben liiert zu sein. Das Chaos ist perfekt!*

Die diensthabende Oberschwester steht einem schwierigen Problem gegenüber: Welches Baby gehört zu welchem Elternpaar? Anders gesagt, um die n glücklichen, harmonischen und paarweise disjunkten Familien wiederherzustellen, muss sie ein dreidimensionales Matching zwischen den n Vätern, n Müttern und n Babies finden. Kein Wunder, dass sich im Gegensatz zum effizient lösbarer Heiratsproblem das Problem 3-DM als NP-vollständig erweist. Schließlich muss die Oberschwester, um das dreidimensionale Matching-Problem zu lösen, 3n Blutproben nehmen, die sie dann für gewisse wirklich raffinierte DNA-Tests verwendet (deren Beschreibung über den Rahmen dieses Buches hinausgeht).

Und wieder entspricht das Resultat, dass 3-DM NP-vollständig ist, dem gesunden Menschenverstand und der Lebensweisheit im Alltag: Wenn Kinder kommen, kann die Aufgabe, eine glückliche und harmonische Familie zu bleiben – disjunkt zu allen anderen Familien! – gelegentlich ganz schön schwer werden!

Satz 3.63. 3-DM ist NP-vollständig.

Beweis. Leicht sieht man, dass 3-DM in NP liegt: Für eine gegebene 3-DM-Instanz $\langle R, U, V, W \rangle$, wobei $R \subseteq U \times V \times W$ eine ternäre Relation über paarweise disjunkten Mengen U , V und W der Größe jeweils n ist, wird nichtdeterministisch eine Teilmenge $M \subseteq R$ der Größe n geraten, und für jede geratene Teilmenge M wird deterministisch überprüft, ob sie ein tripartites Matching von R ist oder nicht.

Die Intuition hinter dem Beweis der NP-Härte wird am besten dadurch verständlich, dass man sich die Lösungsmethode der diensthabenden Oberschwester im Kreißsaal aus der obigen Story 3.62 ansieht. Wie löst sie das tripartite Matching-Problem? Zunächst versieht sie jedermann im Saal mit einem Namensschildchen, wobei sie dafür sorgt, dass es nicht entfernt werden kann. Angenommen, die Mütter sind mit m_1, m_2, \dots, m_n , die Väter sind mit p_1, p_2, \dots, p_n und die Babies sind mit b_1, b_2, \dots, b_n gekennzeichnet. Sei $\bar{b}_1, \bar{b}_2, \dots, \bar{b}_n$ eine weitere Menge von n Babies, wobei jedes \bar{b}_i ein Klon⁷ von b_i ist. Dann arrangiert die Oberschwester alle Personen in zwei Kreisen, so dass die $2n$ Eltern den inneren Kreis bilden, in dem sich Väter und

⁷ Wieder würden die technischen Details des Klonens von Babies – und die Diskussion damit zusammenhängender ethischer Fragen – den Rahmen dieses Buches sprengen.

Mütter abwechseln. Im äußeren Kreis wechseln sich die n Babies und ihre n Klonen ab. Für $n = 4$ zeigt Abbildung 3.6 (rechts) diese beiden Kreise, in denen nebeneinander sitzende (bzw. liegende) Personen so miteinander verbunden sind, dass sie $2n$ Dreiecke bilden. Jeder Vater ist also mit zwei Müttern und zwei Babies verbunden, und jede Mutter ist mit zwei Vätern und zwei Babies verbunden.

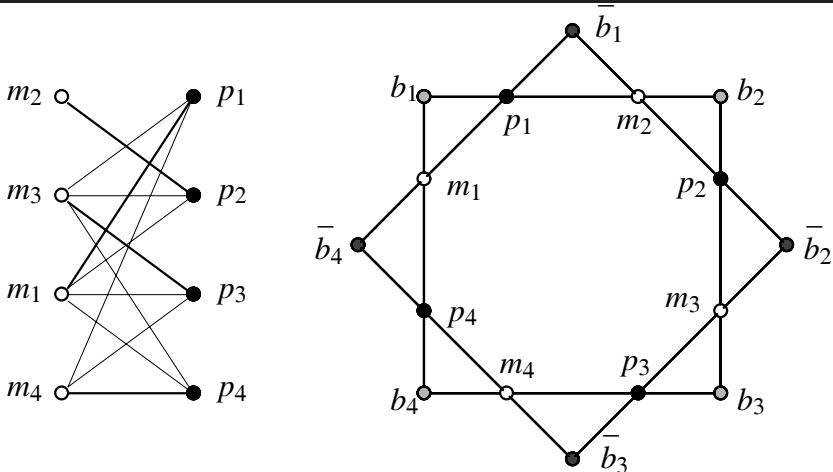


Abb. 3.6. Links: Heiratsproblem Rechts: Wahrheitswert-Komponente für $3\text{-SAT} \leq_m^p 3\text{-DM}$

Für jedes i (modulo $n = 4$) behauptet Vater p_i , mit Mutter m_{i+1} verheiratet zu sein und das i -te Kind gemeinsam mit ihr zu haben. Mutter m_i besteht jedoch darauf, dass sie die Frau von p_i ist und das i -te Baby gemeinsam mit ihm hat. Diese beiden widersprüchlichen Aussagen sind in Abbildung 3.6 (rechts) dargestellt. Die Behauptung von Vater p_i wird durch das Dreieck mit den Ecken p_i, m_{i+1} und \bar{b}_i und die Behauptung von Mutter m_i wird durch das Dreieck mit den Ecken m_i, p_i und b_i gezeigt.

Jedes der $2n$ Dreiecke repräsentiert eine potenzielle Familie, und die Oberschwester muss ermitteln, welches der Dreiecke zu den ursprünglichen n Familien gehört und welches nicht. Die einzige Möglichkeit, n disjunkte Familien zu erhalten, ist, entweder jedes Dreieck mit einem Baby b_i oder jedes Dreieck mit einem geklonten Baby \bar{b}_i zu wählen. Indem sie $3n$ Blutproben nimmt und die Resultate ihrer DNA-Tests verwendet, kann die Oberschwester die richtige Wahl treffen und jeden Vater seiner wirklichen Gattin und seinem leiblichen Kind zuweisen und somit die n ursprünglichen Familien wiederherstellen. Die übrigen n Babies (und das ist die traurige Seite der Lösungsmethode der Oberschwester für dieses Problem – und des Klonens von Babies überhaupt) werden an Waisenhäuser übergeben.

Komplexitätstheoretiker wissen nicht viel über DNA-Tests. Glücklicherweise kennen sie jedoch das Erfüllbarkeitsproblem. Um die NP-Härte von 3-DM zu zeigen, wird nun eine formale Reduktion von 3-SAT auf 3-DM angegeben.

Gegeben eine boolesche Formel $\varphi(x_1, x_2, \dots, x_\ell) = C_1 \wedge C_2 \wedge \dots \wedge C_n$, wobei die Klauseln C_j von φ genau drei Literale haben, besteht das Ziel darin, eine Instanz $\langle R, U, V, W \rangle$ von 3-DM so zu konstruieren, dass $R \subseteq U \times V \times W$ gilt, U, V und W paarweise disjunkte, nichtleere Mengen gleicher Größe sind und die folgende Äquivalenz gilt:

$$\varphi \text{ ist erfüllbar} \iff R \text{ enthält ein tripartites Matching der Größe } \|U\|. \quad (3.17)$$

R besteht aus verschiedenen Typen von Tripeln, die jeweils einer bestimmten Absicht entsprechen. Alle Tripel desselben Typs werden zu einer Komponente gebündelt. Die erste Komponente, nennen wir sie X , enthält die Tripel in R , deren Form eine bestimmte Wahrheitswertbelegung der Variablen der Formel φ erzwingt, wobei sicherzustellen ist, dass diese Belegung für alle Klauseln von φ konsistent ist. Das heißt, kommt eine Variable in verschiedenen Klauseln vor, dann ist bei jedem Vorkommen dieser Variablen derselbe Wahrheitswert zuzuweisen. Deshalb wird diese Komponente X die *Wahrheitswert-Komponente* von R genannt.

Nun werden die Mengen U, V und W und die Komponenten der Relation R definiert, beginnend mit X . Für jede Variable x_i in φ enthält U genau $2n$ Elemente, $b_1^i, b_2^i, \dots, b_n^i$ und $\bar{b}_1^i, \bar{b}_2^i, \dots, \bar{b}_n^i$, wobei n die Anzahl der Klauseln von φ ist. Hier repräsentiert b_j^i das Vorkommen von x_i in C_j , der j -ten Klausel von φ , und \bar{b}_j^i repräsentiert das Vorkommen von $\neg x_i$ in C_j . Da die Literale nicht in allen Klauseln vorkommen, entsprechen manche b_j^i oder \bar{b}_j^i keinem Literalvorkommen in φ . Die Elemente b_j^i und \bar{b}_j^i bilden den äußeren Kreis des in Abbildung 3.6 (rechts) gezeigten Graphen, für $n = 4$ und ohne obere Indizes.

Außerdem enthält, für jede Variable x_i in φ , die Menge V die n Elemente $m_1^i, m_2^i, \dots, m_n^i$ und die Menge W die n Elemente $p_1^i, p_2^i, \dots, p_n^i$, welche den inneren Kreis des in Abbildung 3.6 (rechts) gezeigten Graphen bilden. Definiere nun die in dieser Abbildung dargestellte Wahrheitswert-Komponente: Die Elemente m_j^i, p_j^i und b_j^i und ebenso die Elemente p_j^i, m_{j+1}^i und \bar{b}_j^i bilden jeweils ein Dreieck. Diese Dreiecke entsprechen Tripeln in R . Die m_j^i und p_j^i aus dem inneren Kreis tauchen nur dem Vorkommen der Variable x_i entsprechend auf, wohingegen die b_j^i und \bar{b}_j^i aus dem äußeren Kreis auch sonst auftreten können. Formal hat die Wahrheitswertkomponente X die Form $X = \bigcup_{i=1}^\ell X_i$, wobei für jede Variable x_i in φ die Menge $X_i = F_i \cup T_i$ durch die folgenden beiden Mengen von Tripeln definiert ist:

$$\begin{aligned} F_i &= \{(b_j^i, m_j^i, p_j^i) \mid 1 \leq j \leq n\}; \\ T_i &= \{(\bar{b}_j^i, m_{j+1}^i, p_j^i) \mid 1 \leq j < n\} \cup \{(\bar{b}_n^i, m_1^i, p_n^i)\}. \end{aligned}$$

Da keines der Elemente m_j^i und p_j^i , mit $1 \leq i \leq \ell$ und $1 \leq j \leq n$, aus dem inneren Kreis in irgendeinem Tripel von R außerhalb von $X_i = F_i \cup T_i$ vorkommen wird, muss jedes Matching M von R genau n Tripel aus X_i enthalten, entweder alle Tripel aus F_i oder alle Tripel aus T_i . Intuitiv zwingt uns diese Wahl zwischen Tripeln aus F_i und Tripeln aus T_i für das Matching dazu, eine Entscheidung zu treffen, ob die Variable x_i falsch oder wahr zu setzen ist. Da alle Vorkommen von x_i in der Formel φ in X_i

repräsentiert sind, ist diese Wahl für die ganze Formel konsistent. Folglich spezifiziert jedes Matching M von R eine Wahrheitswertbelegung von φ , so dass für jedes $i \in \{1, 2, \dots, \ell\}$ die Variable x_i genau dann wahr ist, wenn $M \cap X_i = T_i$ gilt.

Als Nächstes fügen wir zu R eine Menge $Y = \bigcup_{j=1}^n Y_j$ von Tripeln hinzu, so dass jedes Y_j die Erfüllbarkeit der entsprechenden Klausel C_j in φ überprüft. Deshalb nennt man Y die *Erfüllbarkeitskomponente* von R . Für jede Klausel C_j gibt es zwei Elemente, $v_j \in V$ und $w_j \in W$, die nur in Y_j vorkommen. Außerdem enthält Y_j drei weitere Elemente aus $\bigcup_{i=1}^\ell \{b_j^i\} \cup \{\bar{b}_j^i\}$, die den drei Literalen in C_j entsprechen und womöglich auch in anderen Komponenten von R vorkommen. Formal ist die Erfüllbarkeitskomponente von R für jede Klausel C_j von φ als die folgende Menge von Tripeln definiert:

$$Y_j = \{(b_j^i, v_j, w_j) \mid x_i \text{ kommt in } C_j \text{ vor}\} \cup \{(\bar{b}_j^i, v_j, w_j) \mid \neg x_i \text{ kommt in } C_j \text{ vor}\}.$$

Da keines der Elemente v_j und w_j , $1 \leq j \leq n$, in irgendeinem Tripel von R außerhalb von Y_j vorkommen wird, muss jedes Matching M von R genau ein Tripel aus Y_j enthalten, entweder (b_j^i, v_j, w_j) oder (\bar{b}_j^i, v_j, w_j) . Jedoch enthält M ein Tripel aus Y_j mit entweder b_j^i (falls x_i in C_j vorkommt) oder \bar{b}_j^i (falls $\neg x_i$ in C_j vorkommt) genau dann, wenn dieses Element nicht zu den Tripeln in $M \cap X_i$ gehört, was genau dann der Fall ist, wenn die von M mittels der Wahrheitswert-Komponente gewählte Belegung die Klausel C_j erfüllt.

Bisher enthält U genau $2n\ell$ Elemente, aber sowohl V als auch W enthalten jeweils nur $n\ell + n$ Elemente. Fügt man $n(\ell - 1)$ weitere Elemente sowohl zu V als auch zu W hinzu, so stellt man sicher, dass diese drei Mengen dieselbe Größe haben. So werden also die Elemente $v_{n+1}, v_{n+2}, \dots, v_{n\ell}$ zu V und die Elemente $w_{n+1}, w_{n+2}, \dots, w_{n\ell}$ zu W und die folgenden Mengen von Tripeln zu R hinzugefügt:

$$\begin{aligned} Z = & \{(b_j^i, v_k, w_k) \mid 1 \leq i \leq \ell \text{ und } 1 \leq j \leq n \text{ und } n+1 \leq k \leq n\ell\} \\ & \cup \{(\bar{b}_j^i, v_k, w_k) \mid 1 \leq i \leq \ell \text{ und } 1 \leq j \leq n \text{ und } n+1 \leq k \leq n\ell\}. \end{aligned}$$

Die Idee ist, dass wenn ein Matching von $R - Z$ alle Bedingungen der Wahrheitswert- und der Erfüllbarkeitskomponente von R erfüllt, dann lässt dieses Matching genau $n(\ell - 1)$ Elemente von U frei, welche nun eindeutig einem Paar (v_k, w_k) in Z zugeordnet werden können. Diese Erweiterung des Matchings von $R - Z$ stellt ein Matching von R dar.

Zusammengefasst werden die Mengen U , V und W wie folgt definiert:

$$\begin{aligned} U &= \{b_j^i \mid 1 \leq i \leq \ell \text{ und } 1 \leq j \leq n\} \cup \{\bar{b}_j^i \mid 1 \leq i \leq \ell \text{ und } 1 \leq j \leq n\}; \\ V &= \{m_j^i \mid 1 \leq i \leq \ell \text{ und } 1 \leq j \leq n\} \cup \{v_k \mid 1 \leq k \leq n\ell\}; \\ W &= \{p_j^i \mid 1 \leq i \leq \ell \text{ und } 1 \leq j \leq n\} \cup \{w_k \mid 1 \leq k \leq n\ell\}, \end{aligned}$$

und die Relation $R \subseteq U \times V \times W$ ist definiert durch

$$R = X \cup Y \cup Z.$$

R enthält genau $2n\ell + 3n + 2n^2\ell(\ell - 1)$ Tripel, und die Struktur von R kann leicht aus der Struktur der gegebenen Formel φ bestimmt werden. Deshalb ist die Reduktion in Polynomialzeit berechenbar. Die Äquivalenz (3.17) folgt aus den während der Konstruktion von R gemachten Bemerkungen; ein formaler Beweis von (3.17) wird dem Leser als Übung 3.17 überlassen. \square

Definition 3.64 (Mengenüberdeckungs- und Mengenpackungsprobleme). Für eine Menge U bezeichne $\mathfrak{P}(U)$ die Potenzmenge von U , d.h., die Menge der Teilmengen von U .

Definiere die Entscheidungsversion des Mengenüberdeckungsproblems durch

$$\text{SetCovering} = \left\{ \langle \mathcal{S}, U, k \rangle \middle| \begin{array}{l} k \in \mathbb{N}, U \text{ ist eine endliche Menge, } \mathcal{S} \subseteq \mathfrak{P}(U) \\ \text{und es gibt } k \text{ Mengen } S_1, S_2, \dots, S_k \text{ in } \mathcal{S}, \text{ die} \\ U \text{ überdecken, d.h., } U = \bigcup_{i=1}^k S_i \end{array} \right\}.$$

Für eine gegebene Menge U und eine Familie $\mathcal{S} \subseteq \mathfrak{P}(U)$ von Teilmengen von U bezeichne $\kappa(\mathcal{S})$ die maximale Anzahl von paarweise disjunkten Mengen in \mathcal{S} .

Definiere die Entscheidungsversion des Mengenpackungsproblems durch

$$\text{SetPacking} = \left\{ \langle \mathcal{S}, U, k \rangle \middle| \begin{array}{l} k \in \mathbb{N}, U \text{ ist eine endliche Menge, } \\ \mathcal{S} \subseteq \mathfrak{P}(U) \text{ und } \kappa(\mathcal{S}) \geq k \end{array} \right\}.$$

Definiere die Entscheidungsversion des exakten Überdeckungsproblems durch Dreiermengen (englisch „exact cover by 3-sets“) durch

$$\text{X-3-Cover} = \left\{ \langle \mathcal{S}, U \rangle \middle| \begin{array}{l} U \text{ ist eine Menge, } \|U\| = 3m \text{ für } m \in \mathbb{N}, \mathcal{S} \subseteq \mathfrak{P}(U), \\ \|S\| = 3 \text{ für alle } S \in \mathcal{S} \text{ und es gibt } m \text{ paarweise dis-} \\ \text{junkte Mengen } S_1, S_2, \dots, S_m \in \mathcal{S} \text{ mit } U = \bigcup_{i=1}^m S_i \end{array} \right\}.$$

Satz 3.65. SetCovering, SetPacking und X-3-Cover sind NP-vollständig.

Beweis. Die Zugehörigkeit zu NP ist für jedes dieser drei Probleme wieder leicht zu sehen. Um die NP-Härte zu zeigen, genügt es festzustellen, dass jedes dieser Probleme das 3-DM-Problem verallgemeinert. Insbesondere ist 3-DM der Spezialfall des Problems X-3-Cover, bei dem das gegebene Universum U in drei paarweise disjunkte Mengen gleicher Größe zerlegt werden kann, etwa A, B und C , so dass jede Menge S in der gegebenen Familie \mathcal{S} genau ein Element von jeweils A, B und C enthält. Das Problem X-3-Cover wiederum ist der Spezialfall des Problems SetCovering, bei dem das gegebene Universum U genau $3m$ Elemente hat, jede Menge S in der gegebenen Familie \mathcal{S} genau drei Elemente enthält und die gegebene Konstante k gleich m ist. Ein ähnliches Argument funktioniert für das Problem SetPacking; siehe Übung 3.18. \square

Zum Schluss des Abschnitts über NP-Vollständigkeit wenden wir uns bestimmten Rucksack-Problemen zu. Diese können als einzeilige ganzzahlige lineare Programme der folgenden Form dargestellt werden:

$$\begin{aligned} \text{Maximiere } & \sum_{i=1}^n s_i x_i \\ \text{unter der Bedingung } & \sum_{i=1}^n s_i x_i \leq T, \quad \text{wobei } x_i \in \mathbb{Z}. \end{aligned}$$

Allgemein wird die *ganzzahlige lineare Programmierung* (englisch „integer linear programming“) zum Finden ganzzahliger Lösungen eines gegebenen Systems von linearen Ungleichungen in n Variablen und mit ganzzahligen Koeffizienten verwendet. Man kann sich ein Rucksack-Problem also als ein Optimierungsproblem vorstellen, dessen Ziel es ist, einen Rucksack der Kapazität T mit Dingen unterschiedlicher Größe ($s_i, 1 \leq i \leq n$) so zu füllen, dass der größtmögliche Nutzwert in den Rucksack passt. Satz 3.67 unten zeigt die NP-Vollständigkeit der ganz speziellen Variante des Rucksack-Problems, bei der $x_i \in \{0, 1\}$ für jedes i ist und bei der verlangt wird, dass die Zielkapazität T genau erreicht wird. Daraus folgt, dass das allgemeine Rucksack-Problem (mit ganzen Zahlen x_i und wobei T lediglich eine obere Schranke ist) und ebenso das noch allgemeinere Problem der ganzzahligen linearen Programmierung ebenso NP-hart sind. Diese beiden Probleme sind auch NP-vollständig; der schwierige Teil des Beweises besteht bei dem letzteren Problem darin, die Zugehörigkeit zu NP zu zeigen. Im Gegensatz dazu kann das Problem der *linearen Programmierung*, welches sich von dem der ganzzahligen linearen Programmierung nur durch den Verzicht auf Ganzzahligkeit unterscheidet, mittels des Algorithmus von H. Lenstra [Len83] in Polynomialzeit gelöst werden, siehe auch Hačijan [Hač79] für eine noch effizientere Variante.

Die oben erwähnte 0-1-Einschränkung des Rucksack-Problems, die auch als das „Subset-of-Sums“-Problem (kurz SOS) bekannt ist, wird nun definiert. Dieses Problem ist für bestimmte kryptographische Anwendungen benutzt worden. Es wurden nämlich Kryptosysteme vorgeschlagen, deren Sicherheit auf der Härte des SOS-Problems und seiner Varianten beruht; siehe Abschnitt 8.5. Einige dieser Systeme wurden gebrochen, wohingegen andere noch als sicher gelten und in Gebrauch sind. Außerdem ist das SOS-Problem eng mit Problemen verwandt, die in der Gitter-Kryptographie verwendet werden, siehe die Literaturhinweise in Abschnitt 8.8.

Definition 3.66 (Subset-of-Sums-Problem). Das Subset-of-Sums-Problem, SOS, ist folgendermaßen definiert: Gegeben eine Folge s_1, s_2, \dots, s_n, T von positiven ganzen Zahlen (wie üblich binär codiert), gibt es in $\{0, 1\}^n$ einen booleschen Vektor $\mathbf{x} = (x_1, x_2, \dots, x_n)$, so dass gilt:

$$\sum_{i=1}^n x_i s_i = T ?$$

Die Zahlen s_i nennt man die Größen, und T ist die Zielsumme. Formalisiert als eine Menge von Ja-Instanzen, hat dieses Entscheidungsproblem die folgende Form:

$$\text{SOS} = \left\{ \langle s_1, s_2, \dots, s_n, T \rangle \mid \begin{array}{l} s_1, s_2, \dots, s_n, T \in \mathbb{N} - \{0\} \text{ und es gibt} \\ \text{ein } \mathbf{x} \in \{0, 1\}^n, \text{ so dass } \sum_{i=1}^n x_i s_i = T \end{array} \right\}.$$

Satz 3.67. SOS ist NP-vollständig.

Beweis. Der Beweis, dass SOS zu NP gehört, wird dem Leser als Übung 3.19 überlassen. Um die NP-Härte zu beweisen, wird das NP-vollständige Problem X-3-Cover auf SOS reduziert. Gegeben sind also eine Menge U mit $3m$ Elementen und eine Familie $\mathcal{S} \subseteq \mathfrak{P}(U)$ von Teilmengen von U , so dass jede Menge in \mathcal{S} genau drei Elemente hat. Unser Ziel ist es, eine Instanz $\langle s_1, s_2, \dots, s_n, T \rangle$ des SOS-Problems zu konstruieren, so dass sich einige der Größen s_i genau dann zu genau T aufaddieren, wenn U in m paarweise disjunkte Mengen aus \mathcal{S} zerlegt werden kann.

Es ist zweckmäßig, die Elemente des Universums U als positive ganze Zahlen aufzufassen, d.h., $U = \{1, 2, \dots, 3m\}$. Sei n die Anzahl der Mengen in der gegebenen Familie \mathcal{S} , d.h., $\mathcal{S} = \{S_1, S_2, \dots, S_n\}$. Jede Menge S_i in \mathcal{S} kann als ein Bitvektor s_i der Dimension $3m$ dargestellt werden. Beispielsweise könnten $U = \{1, 2, \dots, 6\}$ und eine Familie $\mathcal{S} = \{S_1, S_2, S_3\}$ mit drei Mengen gegeben sein:

$$\begin{aligned} S_1 &= \{1, 3, 6\} \text{ entspricht } s_1 = (1, 0, 1, 0, 0, 1); \\ S_2 &= \{3, 4, 6\} \text{ entspricht } s_2 = (0, 0, 1, 1, 0, 1); \\ S_3 &= \{2, 4, 5\} \text{ entspricht } s_3 = (0, 1, 0, 1, 1, 0). \end{aligned}$$

Interpretiere die Bitvektoren s_i als positive ganze Zahlen s_i in $(n+1)$ -ärer Darstellung. Die Basis $n+1$ wurde gewählt, um Probleme mit dem Übertrag bei der Addition von Zahlen zu vermeiden, die durch die s_i repräsentiert werden. Das heißt, für jedes i mit $1 \leq i \leq n$ entspricht die Zahl s_i der Menge S_i , die definiert ist durch

$$s_i = \sum_{j \in S_i} (n+1)^{3m-j}.$$

Im obigen Beispiel ergibt sich:

$$\begin{aligned} s_1 &= 4^5 + 4^3 + 4^0 = 1089; \\ s_2 &= 4^3 + 4^2 + 4^0 = 81; \\ s_3 &= 4^4 + 4^2 + 4^1 = 276. \end{aligned}$$

Das Universum U , das alle ganzen Zahlen j mit $1 \leq j \leq 3m$ enthält, entspricht demnach dem Vektor $1 = (1, 1, \dots, 1)$ der Dimension $3m$. Definiert man die Zielsumme T als diejenige Zahl, die bezüglich der Basis $n+1$ diesem Vektor entspricht:

$$T = \sum_{j=0}^{3m-1} (n+1)^j,$$

so folgt, dass U genau dann in m paarweise disjunkte Mengen aus \mathcal{S} zerlegt werden kann, wenn $\sum_{i=1}^n x_i s_i = T$ für geeignet gewählte Koeffizienten $x_i \in \{0, 1\}$ gilt. Insbesondere ist $x_i = 1$ genau dann, wenn S_i eine der zu dieser Zerlegung von U gehörigen Mengen ist. Im obigen Beispiel ist $U = S_1 \cup S_3$ mit den disjunkten Mengen S_1 und S_3 aus \mathcal{S} , und für den booleschen Koeffizientenvektor $\mathbf{x} = (1, 0, 1)$ ergibt sich $s_1 + s_3 = 1365 = T$. \square

3.6 Innerhalb von NP

3.6.1 P versus NP und das Graphisomorphie-Problem

Ist P gleich NP ? Niemand weiß es. Diese berühmte Frage hat die Komplexitätstheoretiker seit den Anfängen dieses Gebiets geärgert, und noch immer scheint keine Lösung dieses zentralen Problems in Sicht zu sein. Angesichts der Bedeutung der Klassen P und NP ist es verständlich, dass man spekuliert. In der überwiegenden Mehrheit glauben die Komplexitätstheoretiker, dass $P \neq NP$ gilt; siehe die „ $P \neq NP$ “-Abstimmung, die Gasarch im Jahre 2002 durchführte [Gas02]. Sollte sich jemals herausstellen, dass tatsächlich jedoch $P = NP$ gilt, dann wäre die Welt der Komplexitätstheorie um eine herausragende Einsicht reicher, doch gleichzeitig ärmer durch den Verlust eines Großteils ihrer Struktur und zahlreicher Komplexitätsklassen. Beispielsweise würde $P = NP$ Kapitel 5 vollständig hinfällig machen (bis auf Abschnitt 5.6, der sich mit alternierenden Turingmaschinen beschäftigt; der Rest des Kapitels wäre Geschichte). Kapitel 5 studiert einige interessante Hierarchien von Komplexitätsklassen, die auf NP beruhen, und der Kollaps von NP auf P würde alle diese Hierarchien ebenfalls zum Einsturz bringen und auf P kollabieren lassen.

Die P -versus- NP -Frage ist auch in der Kryptographie von zentraler Bedeutung. Insbesondere würde aus $P = NP$ folgen, dass die meisten der derzeit verwendeten Kryptosysteme tatsächlich nutzlos wären, da ihre Sicherheit auf der Annahme beruht, dass gewisse Probleme, wie etwa das Faktorisierungsproblem, schwer zu lösen sind. Das Faktorisierungsproblem fragt nach der Primzahlzerlegung $p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ einer gegebenen Zahl n , zusammen mit Zertifikaten der Primalität ihrer Primfaktoren p_i . Wie wir in Kapitel 7 sehen werden, können Zahlen in nichtdeterministischer Polynomialzeit faktorisiert werden, doch kein effizienter deterministischer Algorithmus ist für dieses Problem bekannt. Falls jedoch $P = NP$ gelten würde, so könnte man sogar in deterministischer Polynomialzeit faktorisieren. Andererseits folgt aus der (hypothetischen) Existenz eines effizienten Faktorisierungsalgorithmus in keiner offensichtlichen Weise die Gleichheit $P = NP$. In dieser Hinsicht unterscheidet sich das Faktorisierungsproblem von den NP -vollständigen Problemen.

Aus Lemma 3.36 ist bekannt, dass $P = NP$ genau dann gilt, wenn auch nur ein NP -vollständiges Problem in P liegt, was genau dann gilt, wenn alle NP -vollständigen Probleme in P sind. Daher kann, falls $P \neq NP$ gilt, kein NP -vollständiges Problem in P sein. Eine natürliche Frage tritt da auf: Angenommen, es gilt $P \neq NP$, kann es dann NP -Probleme geben, die weder in P noch NP -vollständig sind? Ladner bewies, dass die Antwort auf diese Frage positiv ist. Auf den Beweis seines Resultats wird hier verzichtet, weil es implizit später bewiesen wird: Satz 3.68 ist ein Spezialfall eines Ergebnisses von Schöning, das in Abschnitt 5.7 als Satz 5.88 erscheint.

Satz 3.68 (Ladner).

$P \neq NP$ genau dann, wenn es Mengen in NP gibt, die weder in P liegen noch NP -vollständig sind.

Falls $P \neq NP$ gilt, gibt es nach Satz 3.68 zwar Probleme in NP , die weder in P noch NP -vollständig sind. Aber die im Beweis dieses Resultats konstruierten Probleme sind nicht übermäßig natürlich. Gibt es irgendwelche *natürlichen* NP -Probleme,

die weder in P noch NP-vollständig sind? Das Primzahl-Problem, welches fragt, ob eine gegebene Zahl eine Primzahl ist, galt lange Zeit als ein guter Kandidat für ein solches Problem. Bis Agrawal, Kayal und Saxena [AKS02] zeigten, dass dieses Problem doch in P liegt. Effiziente Primzahltests sind für viele Kryptosysteme wichtig, einschließlich des RSA-Systems; siehe Kapitel 7.

Ein anderer guter Kandidat ist das Graphisomorphie-Problem, siehe Definition 2.49 in Abschnitt 2.4. Der folgende Fakt kann leicht bewiesen werden; siehe Übung 3.20.

Fakt 3.69 GI ist in NP.

Kapitel 6 wird Indizien dafür liefern, dass GI vermutlich tatsächlich ein NP-Problem sein könnte, das weder in P noch NP-vollständig ist. Andererseits teilt es viele Eigenschaften der NP-vollständigen Mengen. Beispielsweise ist GI, genau wie alle „natürlichen“ NP-vollständigen Probleme, selbstreduzierbar. Selbstreduzierbarkeit ist eine wichtige Eigenschaft und ist in den unterschiedlichsten Zusammenhängen untersucht worden. Zum Beispiel machen die in Abschnitt 3.6.2 präsentierten Ergebnisse, und unter diesen besonders Satz 3.75, intensiven Gebrauch von der Selbstreduzierbarkeit NP-vollständiger Probleme wie etwa SAT; siehe auch die Bemerkungen in Abschnitt 3.8 dazu.

Intuitiv nennt man eine Menge A *selbstreduzierbar*, falls es einen effizienten Algorithmus zur Lösung von A gibt, der die Menge A selbst als ein Orakel benutzt. Wenn dieser Algorithmus einfach sein Orakel über sein Eingabewort befragen könnte, dann wäre natürlich jede Menge selbstreduzierbar und dieser Begriff somit trivial. Um dies zu verhindern, sind in einer Selbstreduktion nur Fragen über Wörter erlaubt, die „kleiner“ als das Eingabewort sind. Würde sich „kleiner“ hier lediglich auf die Wortlänge beziehen, dann wären Selbstreduktionen von der verwendeten Codierung abhängig, was nicht wünschenswert ist. Eine Vielzahl von Definitionen ist in der Literatur vorgeschlagen worden, um den Begriff der Selbstreduzierbarkeit formal einzufangen. Unter diesen hat sich der von A. Meyer und Paterson [MP79] eingeführte Begriff als besonders nützlich erwiesen. Ihr formaler Zugang hat den Vorteil, „volle Allgemeinheit zu erzielen und den Begriff unter polynomiell berechenbaren Isomorphismen zu erhalten“ [JY90, p. 84], siehe auch Abschnitt 3.6.2.

Definition 3.70 (Selbstreduzierbarkeit).

1. Eine Halbordnung $<_{\text{pwl}}$ auf Σ^* ist polynomiell wohlfundiert und längenbeschränkt, falls die folgenden beiden Eigenschaften gelten:
 - (a) Jede echt abfallende Kette ist endlich, und es gibt ein Polynom p , so dass jede endliche $<_{\text{pwl}}$ -abfallende Kette kürzer als p von der Länge ihres maximalen Elements ist.
 - (b) Es gibt ein Polynom q , so dass für alle $x, y \in \Sigma^*$ aus $x <_{\text{pwl}} y$ folgt, dass $|x| \leq q(|y|)$ gilt.
2. Eine Menge A ist selbstreduzierbar, falls es eine polynomiell wohlfundierte und längenbeschränkte Halbordnung $<_{\text{pwl}}$ auf Σ^* und eine DPOTM M gibt, so dass $A = L(M^A)$ gilt und M bei jeder Eingabe $x \in \Sigma^*$ nur Fragen y mit $y <_{\text{pwl}} x$ stellt.

Satz 3.71. *Sowohl SAT als auch GI sind selbstreduzierbar.*

Satz 3.71 wird hier nicht formal bewiesen. Stattdessen wird lediglich die Intuition des Beweises für das Erfüllbarkeitsproblem skizziert. Für eine gegebene boolesche Formel φ seien φ_0 und φ_1 die Formeln, die man erhält, wenn man die erste Variable von φ falsch bzw. wahr setzt. Intuitiv ist klar, dass φ_0 und φ_1 kleiner als φ sind, da sie eine Variable weniger als φ haben.

Konstruiere nun den Selbstreduzierbarkeitsbaum für eine gegebene Instanz ψ von SAT wie folgt: Die Wurzel wird mit ψ markiert, und jeder Knoten des Baums, der mit einer Formel φ markiert ist, erhält zwei Kinder, die mit φ_0 bzw. φ_1 zu markieren sind. Die Tiefe des Baumes ist durch die Anzahl der Variablen, etwa n , beschränkt, und die Anzahl seiner Blätter ist höchstens 2^n . Die Blätter enthalten Formeln, die unmittelbar ausgewertet werden können, da sie keine Variablen mehr haben. Da außerdem für jede Formel φ

$$\varphi \in \text{SAT} \iff \varphi_0 \in \text{SAT} \vee \varphi_1 \in \text{SAT}$$

gilt, ist die Formel ψ an der Wurzel genau dann erfüllbar, wenn es einen Pfad von der Wurzel zu einem Blatt gibt, so dass jede Formel auf diesem Pfad erfüllbar ist. Deshalb nennt man die gerade beschriebene Prozedur eine disjunktive Selbstreduktion.

3.6.2 Die Berman–Hartmanis-Isomorphievermutung und Einwegfunktionen

Ein Isomorphismus zwischen zwei Mengen, A und B , ist eine Bijektion (d.h. eine injektive und surjektive Abbildung) von A auf B . Das Cantor–Bernstein-Theorem der Mengenlehre sagt, dass A und B genau dann zueinander isomorph sind, wenn es eine Injektion von A in B und eine Injektion von B in A gibt.

Aus der Berechenbarkeitstheorie ist bekannt, dass alle in RE many-one-vollständigen Mengen paarweise isomorph sind; siehe das Buch [Rog67] von H. Rogers. Zu beachten ist, dass „many-one-vollständig“ hier im Sinne der Berechenbarkeitstheorie gemeint ist, d.h., die Reduktion muss zwar berechenbar, aber nicht unbedingt in Polynomialzeit berechenbar sein. Angesichts dieses Resultats ergibt sich natürlich die Frage, ob eine analoge Aussage auch für die NP-vollständigen Mengen und für in Polynomialzeit berechenbare Isomorphismen gilt.

Definition 3.72 (P-Isomorphismus). *Eine Funktion $\varphi : \Sigma^* \rightarrow \Sigma^*$ ist ein P-Isomorphismus, falls*

1. φ eine Bijektion auf Σ^* ist (falls φ also eine totale, injektive und surjektive Funktion auf Σ^* ist), und
2. sowohl φ als auch φ^{-1} in FP sind.

Je zwei Mengen A und B sind p-isomorph, falls $A \leq_m^p B$ mittels einer Reduktion φ gilt, die ein P-Isomorphismus ist.

Anders gesagt ist A genau dann p-isomorph zu B , wenn es eine in Polynomialzeit berechenbare und in Polynomialzeit invertierbare Permutation φ auf Σ^* gibt, so dass $\varphi(A) = B$ und $\varphi(\overline{A}) = \overline{B}$ gilt, wobei $\varphi(X)$ für eine Menge X als $\varphi(X) = \{\varphi(x) \mid x \in X\}$ definiert ist.

1977 bewiesen Berman und Hartmanis [BH77], dass alle damals bekannten „natürlichen“ NP-vollständigen Probleme paarweise p-isomorph sind. Ihre Resultate führten sie zu der folgenden berühmten Vermutung.

Vermutung 3.73 (Isomorphievermutung von Berman und Hartmanis). Alle NP-vollständigen Mengen sind paarweise p-isomorph.

Die Berman–Hartmanis–Isomorphievermutung sagt also, dass die NP-vollständigen Probleme in Wirklichkeit nur ein und dasselbe Problem in vielen verschiedenen Verkleidungen repräsentieren. Die Bedeutung von Vermutung 3.73 liegt angesichts des folgenden Resultats auf der Hand.

Satz 3.74. *Wenn alle NP-vollständigen Mengen paarweise p-isomorph sind, dann ist $P \neq NP$.*

Beweis. Um die Kontraposition zu beweisen, sei $P = NP$ angenommen. Dann sind nach Lemma 3.37 alle nichtrivialen Probleme in P (also alle von \emptyset und Σ^* verschiedenen P-Probleme) \leq_m^p -vollständig für NP. Insbesondere sind alle nichtrivialen endlichen Mengen in P \leq_m^p -vollständig in NP. Zwei endliche Mengen mit unterschiedlich vielen Elementen können jedoch nicht isomorph sein. Somit können sie erst recht nicht p-isomorph sein. \square

Nach Satz 3.74 würde ein Beweis der Berman–Hartmanis–Isomorphievermutung auch die P-versus-NP-Frage beantworten. Daher scheint ein Beweis von Vermutung 3.73 gegenwärtig außer Reichweite. Selbst ein Beweis der schwächeren Vermutung, dass aus $P \neq NP$ die paarweise P-Isomorphie zwischen allen NP-vollständigen Mengen folgt, was gerade die Umkehrung der Implikation von Satz 3.74 ist, scheint derzeit nicht möglich zu sein. Berman und Hartmanis [BH77] boten eine noch schwächere Vermutung an: Ist $P \neq NP$, so ist keine dünne Menge \leq_m^p -vollständig in NP.

Dünne Mengen sind ausgesprochen intensiv untersucht worden und spielen eine zentrale Rolle in vielen Resultaten der Komplexitätstheorie. Für eine Sprache S und ein $n \in \mathbb{N}$, ist die *Menge der Wörter in S bis zur Länge n* definiert als $S^{\leq n} = \{x \mid x \in S \text{ und } |x| \leq n\}$. Eine Sprache S heißt *dünn*, falls es ein Polynom p gibt, so dass $\|S^{\leq n}\| \leq p(n)$ für alle $n \in \mathbb{N}$ gilt; siehe auch Definition 5.58. Festzustellen ist, dass keines der bekannten NP-vollständigen Probleme dünn ist. Auch kann keine nicht-dünnen Menge durch einen P-Isomorphismus auf eine dünnen Menge abgebildet werden. Als Konsequenz ergibt sich, dass wenn die Aussage von Vermutung 3.73 gilt, dann kann keine dünen Menge \leq_m^p -vollständig in NP sein.

Auf Vorarbeiten von Berman [Ber78] und Fortune [For79] aufbauend (siehe Abschnitt 3.8) konnte Mahaney [Mah82] schließlich diese schwächere Vermutung von Berman und Hartmanis beweisen. Auf den Beweis des Ergebnisses von Mahaney,

welcher insbesondere die Selbstreduzierbarkeit des Erfüllbarkeitsproblems ausnutzt, wird hier verzichtet. Mahaney's Resultat wurde später von Ogiwara und Watana-be [OW91] verschärft; siehe Abschnitt 5.9.

Satz 3.75 (Mahaney). Ist $P \neq NP$, so ist keine dünne Menge \leq_m^p -vollständig in NP .

Ein wichtiger Zwischenschritt auf dem Weg zum Beweis von Satz 3.75 wird unten angegeben. Satz 3.76 stellt ein verwandtes Resultat für dünne Mengen mit einer leichten Zensusfunktion bereit. Die Zensusfunktion einer Menge L bildet jede (unär dargestellte) Zahl n auf die Anzahl der Elemente in L bis zur Länge n ab; das heißt, $census_L(1^n) = \|L^{\leq n}\|$ für jedes n .

Satz 3.76. Für jede dünne Menge S mit $census_S \in FP$ gilt: Ist S in NP , so ist auch \overline{S} in NP .

Beweis. Sei M eine NPTM für S . Bei einer Eingabe x der Länge n arbeitet eine NPTM N für \overline{S} wie folgt:

Schritt 1: Berechne $k = census_S(1^n)$.

Schritt 2: Rate nichtdeterministisch eine Folge $s = (s_1, s_2, \dots, s_k)$ von paarweise verschiedenen Wörtern, deren Länge jeweils höchstens n ist.

Schritt 3: Simuliere für jede geratene Folge s und sukzessive für jedes Wort s_i in s die nichtdeterministische Berechnung von M bei Eingabe s_i , um zu verifizieren, dass jedes s_i in S ist. Lehne ab, wenn mindestens einer dieser Tests fehlschlägt.

Schritt 4: Akzeptiere genau dann, wenn x nicht in s ist.

Da M eine NPTM ist und weil k polynomiell in n ist und in einer in n polynomialen Zeit berechnet werden kann, ist N eine NPTM. Offenbar gilt $L(N) = \overline{S}$. Somit ist \overline{S} in NP . \square

Sei $coNP = \{L \mid \overline{L} \in NP\}$ die Klasse der Komplemente von NP -Mengen. Die offene Frage, ob NP gleich $coNP$ ist oder nicht, ist beinahe so berühmt wie die P -versus- NP -Frage. Auch wenn ein Beweis von $NP \neq coNP$ derzeit nicht im Anwendungsbereich vorhandener Beweistechniken zu liegen scheint, wird es für höchst unwahrscheinlich gehalten, dass NP unter Komplement abgeschlossen sein könnte, dass diese beiden Klassen also gleich sein könnten. Deshalb sagt Korollar 3.77, dass es sehr unwahrscheinlich ist, dass eine dünne Menge mit einer leichten Zensusfunktion \leq_m^p -vollständig in NP ist.

Korollar 3.77. Ist S eine dünne \leq_m^p -vollständige Menge in NP mit $census_S \in FP$, so gilt $NP = coNP$.

Einen mit Vermutung 3.73 rivalisierenden Standpunkt vertreten Joseph und P. Young [JY85]. Inspiriert vom Begriff der kreativen Menge in der Berechenbarkeitstheorie,⁸ definieren sie die *k-creativen Mengen für NP*, indem sie verlangen,

⁸ Eine rekursiv aufzählbare Menge A ist *kreativ*, falls es eine Funktion $f \in \mathbb{R}$ gibt, so dass für jede rekursiv aufzählbare Menge R_i gilt: $R_i \subseteq \overline{A}$ impliziert $f(i) \in \overline{A} - R_i$. Die Idee ist, dass

dass die produktiven Funktionen für NP-Mengen injektiv und in Polynomialzeit berechenbar (jedoch nicht notwendig in Polynomialzeit invertierbar) seien und dass die Mengen R_i (gemäß der Definition der Kreativität in Fußnote 8) nicht ganz NP, sondern lediglich solche NP-Mengen umfassen müssen, die von NTMs mit einer Laufzeit p für ein Polynom p mit festem Grad k akzeptiert werden.

Mit dieser Definition bewaffnet, bewiesen Joseph und P. Young [JY85], dass jede k -kreative Menge für $\text{NP} \leq_m^p$ -vollständig in NP ist. Außerdem ist jede injektive, in Polynomialzeit berechenbare, „ehrliche“ Funktion eine produktive Funktion für eine k -kreative Menge. Hier heißt eine Funktion $f : \Sigma^* \rightarrow \Sigma^*$ *ehrlich*, falls es ein Polynom p gibt, so dass für jedes $y \in R_f$ ein $x \in D_f$ mit $y = f(x)$ und $|x| \leq p(|y|)$ existiert. Das heißt, ehrliche Funktionen schrumpfen ihre Eingaben nie mehr als polynomiell. Ehrlichkeit ist notwendig, damit der Begriff der Nicht-FP-Invertierbarkeit in Definition 3.78 unten nicht trivialisiert wird.

Der folgende Begriff einer Einwegfunktion beruht auf dem Worst-Case-Modell der Komplexitätstheorie. Kryptographische Einwegfunktionen erfordern einen stärkeren Begriff der Nichtinvertierbarkeit, der auf der Average-Case-Komplexität und auf Invertern beruht, die randomisierte (und nicht nur deterministische) Algorithmen sind.

Definition 3.78 (Einwegfunktion). Sei $f : \Sigma^* \rightarrow \Sigma^*$ eine injektive Funktion. Mit R_f wird der Wertebereich von f bezeichnet.

1. Wir sagen, f ist FP-invertierbar, falls es eine Funktion $g \in \text{FP}$ gibt, so dass $f(g(y)) = y$ für jedes $y \in R_f$ gilt.
2. Wir sagen, f ist eine Einwegfunktion, falls f ehrlich, in FP und nicht FP-invertierbar ist.

Auf Grund ihrer oben erwähnten Ergebnisse [JY85] vermuteten Joseph und P. Young, dass nicht alle k -kreativen Mengen p-isomorph zu SAT sind, falls Einwegfunktionen existieren. Demnach wäre die Isomorphievermutung von Berman und Hartmanis falsch. Insbesondere stellten sie Vermutung 3.79 auf.

Vermutung 3.79 (Einweg/Isomorphie-Vermutung von Joseph und Young). Gibt es Einwegfunktionen, dann gibt es NP-vollständige Mengen, die nicht p-isomorph sind.

In [JY85, You83] wird die Frage aufgeworfen, ob die Umkehrung der Implikation in Vermutung 3.79 gilt. Kurtz, Mahaney und Royer [KMR87] formulierten ihrerseits diese Umkehrung der Implikation als eine Vermutung.

Vermutung 3.80 (Einweg-Vermutung von Kurtz, Mahaney und Royer). Gibt es NP-vollständige Mengen, die nicht p-isomorph sind, dann gibt es Einwegfunktionen.

Hartmanis und L. Hemaspaandra [HH91] fanden ein relativiertes Gegenbeispiel zu Vermutung 3.80: Es gibt ein Orakel, relativ zu dem es NP-vollständige Mengen

\overline{A} nicht rekursiv aufzählbar sein kann, da für jeden Kandidaten R_i , der womöglich gleich \overline{A} sein könnte, f ein Element $f(i)$ produziert, durch welches $\overline{A} \not\subseteq R_i$ gilt. Deshalb nennt man f eine *produktive Funktion* für \overline{A} .

gibt, die nicht p-isomorph sind, jedoch gibt es keine Einwegfunktionen in dieser relativierten Welt. In ähnlicher Weise gab J. Rogers [Rog97] ein relativiertes Gegenbeispiel zu Vermutung 3.79 an: In einer geeignet relativierten Welt gilt die Isomorphievermutung (d.h., alle NP-vollständigen Mengen sind paarweise p-isomorph) und doch gibt es Einwegfunktionen in ihr. Diese Forschungslinie wird hier nicht weiter verfolgt. Weitere Details und Literaturhinweise findet man in Abschnitt 3.8.

In enger Beziehung zur Existenz von Einwegfunktionen steht die Komplexitätsklasse UP, die von Valiant [Val76] eingeführt wurde.

Definition 3.81 (Unambiguous Polynomial Time). UP ist die Klasse der Mengen L , für die es eine NPTM M gibt, so dass gilt:

1. Für jede Eingabe x hat $M(x)$ höchstens einen akzeptierenden Berechnungspfad und
2. $L = \{x \in \Sigma^* \mid M(x) \text{ hat einen akzeptierenden Berechnungspfad}\}$.

Eine NTM, die die ersten beiden Bedingungen erfüllt, heißt nichtmehrdeutige Turingmaschine (englisch „unambiguous Turing machine“) oder auch kategorische Turingmaschine.

Aus der Definition ergeben sich sofort die Inklusionen $P \subseteq UP \subseteq NP$; von keiner weiß man, ob sie echt ist. Die Frage, ob P gleich UP ist oder nicht, kann mittels der Existenz von Einwegfunktionen charakterisiert werden.

Satz 3.82. Die folgenden drei Aussagen sind paarweise äquivalent:

1. $P \neq UP$.
2. Es gibt Einwegfunktionen.
3. Es gibt eine Menge $B \in P$ von booleschen Formeln, so dass jede Formel in B höchstens eine erfüllende Belegung hat und $B \cap SAT$ nicht in P liegt.

Beweis. 1. Zunächst wird die Äquivalenz der ersten beiden Aussagen gezeigt. Angenommen, es gilt $P \neq UP$. Sei L eine Menge in $UP - P$, und sei M eine kategorische NPTM, die L akzeptiert. Für jedes Wort $x \in L$ bezeichne $\alpha_M(x)$ den eindeutigen akzeptierenden Berechnungspfad von $M(x)$, codiert als ein Binärwort. Definiere für das Alphabet $\Sigma = \{0, 1\}$ die Funktion $f : \Sigma^* \rightarrow \Sigma^*$ durch

$$f(x) = \begin{cases} y0 & \text{falls } x = \alpha_M(y) \\ x1 & \text{sonst.} \end{cases}$$

Leicht sieht man, dass f total, injektiv, ehrlich und in Polynomialzeit berechenbar ist. Wäre f auch FP-invertierbar, dann könnte L deterministisch in Polynomialzeit entschieden werden, indem für ein gegebenes Wort y der Wert von $f^{-1}(y0)$ berechnet wird; ein Widerspruch zur Annahme $L \notin P$. Somit ist f eine Einwegfunktion.

Zum Beweis der Umkehrung wird die Kontraposition gezeigt: Angenommen, es gilt $P = UP$. Sei f eine totale, ehrliche, injektive Funktion in FP. Zu zeigen ist, dass f FP-invertierbar ist. Für eine jede Funktion g bezeichne

$$\Pi_g = \{\langle x, y \rangle \mid x \in D_g \text{ und } y \leq g(x)\}$$

die *Projektion von g*, wobei \leq die übliche lexikographische Ordnung auf Σ^* ist. Für jede polynomiell längenbeschränkte Funktion g (d.h., es gibt ein $p \in \mathbb{N}$, so dass $|g(x)| \leq p(|x|)$ für jedes x gilt) ist Π_g genau dann in P, wenn g in FP ist; siehe Übung 3.21.

Betrachte die inverse Funktion f^{-1} . Da f ehrlich und injektiv ist, ist $\Pi_{f^{-1}}$ in UP, und die entsprechende kategorische Maschine arbeitet so: Bei Eingabe $\langle x, y \rangle$ wird (nichtmehrdeutig) $f^{-1}(x)$ berechnet, und die Eingabe wird genau dann akzeptiert, wenn $y \leq f^{-1}(x)$ gilt. Laut Annahme ist $\Pi_{f^{-1}}$ in P. Somit ist f^{-1} in FP. Also gibt es keine Einwegfunktion.

2. Nun wird die Äquivalenz der zweiten und dritten Aussage gezeigt. Angenommen, es gibt eine Menge B boolescher Formeln, so dass B in P ist, jede Formel in B höchstens eine erfüllende Belegung hat und $B \cap \text{SAT}$ nicht in P ist. Definiere eine NPTM N wie folgt: Bei Eingabe φ überprüft N zunächst deterministisch, ob φ eine boolesche Formel in B ist, und falls das so ist, rät sie eine Belegung t von φ und akzeptiert genau dann, wenn t die Formel φ erfüllt. Offensichtlich ist N eine kategorische NPTM, die $B \cap \text{SAT}$ akzeptiert. Folglich ist $B \cap \text{SAT}$ eine Menge in UP – P.

Sei nun umgekehrt $P \neq \text{UP}$ angenommen. Sei L eine Menge in der Differenz UP – P, sei M eine kategorische NPTM, die L akzeptiert, und sei f_M die in Satz 3.49 konstruierte Cook-Reduktion. Also ist $f_M(x) = F_{M,x}$ für jede Eingabe x eine boolesche Formel mit:

$$x \in L \iff F_{M,x} \in \text{SAT}. \quad (3.18)$$

Da M eine kategorische NPTM ist, hat $F_{M,x}$ höchstens eine erfüllende Belegung für $x \in \Sigma^*$.

Eine sorgfältige Inspektion des Beweises von Satz 3.49 offenbart, dass die Cook-Reduktion „geizig“ (englisch „parsimonious“) ist, d.h., die Anzahl der verschiedenen akzeptierenden Berechnungspfade von $M(x)$ ist gleich der Anzahl der verschiedenen erfüllenden Belegungen von $F_{M,x}$. Außerdem sind sowohl das Maschinenprogramm von M als auch die Eingabe x in die Formel $F_{M,x}$ codiert. Daher kann man für eine gegebene Formel φ in Polynomialzeit entscheiden, ob φ gleich $F_{M,x}$ für ein Wort x ist oder nicht. Folglich ist die Menge $B = \{F_{M,x} \mid x \in \Sigma^*\}$ in P. Jedoch ist $B \cap \text{SAT}$ nicht in P, da sonst L in P wäre, im Widerspruch zur Annahme $L \notin P$. \square

Korollar 3.83. UP hat genau dann \leq_m^P -vollständige Mengen, wenn es in P eine Menge B boolescher Formeln gibt, so dass jede Formel in B höchstens eine erfüllende Belegung hat und $B \cap \text{SAT} \leq_m^P$ -vollständig in UP ist.

Beweis. Die Implikation von rechts nach links gilt unmittelbar. Um umgekehrt die Implikation von links nach rechts zu beweisen, seien L eine für UP vollständige Menge und M eine kategorische NPTM, die L akzeptiert. Wie im Beweis von Satz 3.82 ist die Menge $B = \{F_{M,x} \mid x \in \Sigma^*\}$ in P, wobei die Formel $F_{M,x}$ aus der Cook-Reduktion für $x \in \Sigma^*$ resultiert. Da M eine kategorische NPTM ist und da die Cook-Reduktion geizig ist, ist $B \cap \text{SAT}$ in UP. Des Weiteren ergibt sich aus der Äquivalenz (3.18) im Beweis von Satz 3.82, dass $L \leq_m^P B \cap \text{SAT}$ mittels der Cook-Reduktion gilt. Folglich ist $B \cap \text{SAT} \leq_m^P$ -vollständig in UP. \square

Satz 3.84. Die folgenden drei Aussagen sind paarweise äquivalent:

1. $P \neq UP \cap coUP$.
2. Es gibt surjektive Einwegfunktionen.
3. Es gibt eine Menge B in P , so dass gilt:
 - a) $B \subseteq SAT$,
 - b) jede Formel in B hat genau eine erfüllende Belegung und
 - c) keine FP-Funktion kann für jedes $\varphi \in B$ eine erfüllende Belegung finden, d.h., die durch

$$f(\varphi) = \begin{cases} \text{die eindeutige erfüllende Belegung von } \varphi & \text{falls } \varphi \in B \\ 0 & \text{falls } \varphi \notin B \end{cases} \quad (3.19)$$

definierte Funktion ist nicht in Polynomialzeit berechenbar.

Beweis. 1. Die Äquivalenz der ersten beiden Aussagen des Satzes kann analog zum Beweis von Satz 3.82 bewiesen werden; siehe Übung 3.23.

2. Nun wird die Äquivalenz der ersten und dritten Aussage des Satzes gezeigt. Sei L eine Menge in $(UP \cap coUP) - P$, und seien M bzw. \bar{M} kategorische NPTMs, die L bzw. \bar{L} akzeptieren. Definiere eine NPTM N wie folgt: Bei Eingabe x verzweigt N nichtdeterministisch in einem Schritt, wobei sie rät, ob $x \in L$ oder $x \notin L$ gilt. Auf dem einen Zweig simuliert N die Berechnung $M(x)$; auf dem anderen Zweig simuliert N die Berechnung $\bar{M}(x)$. Folglich gilt:

$$L(N) = L(M) \cup L(\bar{M}) = L \cup \bar{L} = \Sigma^*.$$

Sei $F_{N,x}$ die aus der Cook-Reduktion resultierende Formel für N und $x \in \Sigma^*$. Da sowohl M als auch \bar{M} kategorische NPTMs sind, ist auch N eine kategorische NPTM. Da die Cook-Reduktion geizig ist, hat $F_{N,x}$ höchstens eine erfüllende Belegung für jedes $x \in \Sigma^*$.

Definiere $B = \{F_{N,x} \mid x \in \Sigma^*\}$. Wie im Beweis von Satz 3.82 ist B in P . Da $L(N) = \Sigma^*$ ist, gilt $B \subseteq SAT$. Somit erfüllt B die ersten beiden Bedingungen, (3.a) und (3.b), der dritten Aussage. Um die dritte Bedingung, (3.c), zu zeigen, muss man sich nur überlegen, dass man mittels einer erfüllenden Belegung von $F_{N,x}$ leicht bestimmen kann, ob $x \in L$ oder $x \notin L$ in der nichtdeterministischen Anfangsverzweigung von $N(x)$ geraten wurde. Wäre also die in (3.19) definierte Funktion f polynomialzeit-berechenbar, so würde folgen, dass L in P ist, ein Widerspruch. Somit ist f nicht in FP, womit (3.c) bewiesen ist.

Sei nun umgekehrt angenommen, dass B eine Menge in P ist, die die drei Bedingungen (3.a) bis (3.c) in der dritten Aussage des Satzes erfüllt.

Für jede boolesche Formel $\varphi(x_1, x_2, \dots, x_n)$ und für jede (partielle) Belegung $t = (a_1, a_2, \dots, a_k)$ von φ , $k \leq n$, sei φ_t die Formel, die man erhält, wenn man die entsprechenden Variablen von φ mit den Werten a_i belegt; φ_t hängt also nur noch von den verbleibenden Variablen ab. Definiere die Menge \hat{B} durch

$$\hat{B} = \left\{ \langle \varphi, t \rangle \mid \begin{array}{l} \varphi \in B, t \text{ ist eine (partielle) Belegung von } \varphi \text{ und} \\ \varphi_t \text{ hat eine eindeutige erfüllende Belegung} \end{array} \right\}.$$

Offensichtlich ist \hat{B} in UP. Wäre \hat{B} ebenfalls in P, dann könnte für jede Formel in B unter Verwendung von \hat{B} und der Selbstreduzierbarkeit von SAT eine erfüllende Belegung konstruiert werden. Dies würde jedoch der dritten Bedingung widersprechen, die B erfüllt. Folglich ist \hat{B} nicht in P.

Dass \hat{B} in coUP ist, liegt an der Äquivalenz

$$\langle \varphi, t \rangle \notin \hat{B} \iff \varphi \notin B \vee (\varphi \in B \wedge \bar{\varphi}_t \text{ hat eine eindeutige erfüllende Belegung}),$$

wobei $\bar{\varphi}_t$ als die Formel definiert ist, die man aus φ erhält, wenn man alle die Belegungen nimmt, die t widersprechen. Ist beispielsweise $t = (a_1, a_2, \dots, a_k)$, so ist

$$\bar{\varphi}_t = \varphi_{(\neg a_1)} \vee \varphi_{(a_1, \neg a_2)} \vee \dots \vee \varphi_{(a_1, a_2, \dots, a_{k-1}, \neg a_k)}.$$

Somit hat $\bar{\varphi}_t$ höchstens eine erfüllende Belegung. Es folgt $\hat{B} \in (UP \cap coUP) - P$. \square

3.7 Übungen und Probleme

Aufgabe 3.1 (a) Beweise, dass jede k -bändrige DTM, die in der Zeit t arbeitet, durch eine äquivalente einbändrige DTM simuliert werden, die in der Zeit $\mathcal{O}(t^2)$ arbeitet.

(b) Beweise das analoge Resultat für nichtdeterministische Turingmaschinen.

Aufgabe 3.2 In Definition 3.2 wurden deterministische Zeit- und Raumklassen definiert. Ersetze die Bedingung „ $\text{time}_M(n) \leq t(n)$ “ in der Definition der Komplexitätsklasse $\text{DTIME}(t)$ durch „ $\text{time}_M(n) \leq_{ae} t(n)$ “ und ersetze die Bedingung „ $\text{space}_M(n) \leq s(n)$ “ in der Definition der Komplexitätsklasse $\text{DSPACE}(s)$ durch „ $\text{space}_M(n) \leq_{ae} s(n)$ “. Entstehen durch diese Änderungen in der Definition andere Komplexitätsklassen? Was ist mit den nichtdeterministischen Klassen $\text{NTIME}(t)$ und $\text{NSPACE}(s)$ aus Definition 3.4?

Aufgabe 3.3 Zeige, dass die deterministischen und nichtdeterministischen Zeit- und Raumfunktionen aus den Definitionen 3.1 und 3.3 Blumsche Komplexitätsmaße sind.

Aufgabe 3.4 Nimm im Beweis von Satz 3.10 an, dass eine DTM M' gegeben ist, die keine der Bedingungen erfüllt, die die DTM M in diesem Beweis erfüllen soll. Konstruiere eine DTM M , die äquivalent zu M' ist (d.h., $L(M) = L(M')$) und die jede dieser Bedingungen erfüllt: (a) M hat nur ein Band, welches (b) einseitig unendlich ist, (c) die Bandzellen sind mit 1, 2, etc. nummeriert, und (d) der Kopf von M macht nur auf geradzahligen Zellen eine Linkswendung. Eine informale Beschreibung von M genügt.

Aufgabe 3.5 Definiere die folgenden Funktionen:

$$c(n) = n; \quad d(n) = 5n; \quad e(n) = n \log n; \quad f(n) = n^2;$$

$$g(n) = n((n \bmod 2) + n((n+1) \bmod 2));$$

$$h(n) = n(n(n \bmod 2) + (n+1) \bmod 2).$$

- (a) Beweise, dass $c \preceq d$ und $d \preceq c$ und $c \prec e$ und $d \prec e \prec f$ gilt.
 (b) Beweise, dass $g \prec_{\text{io}} h$ und $g \succ_{\text{io}} h$ und $g \preceq f$ und $g \prec_{\text{io}} f$ und $g \succeq_{\text{io}} f$ gilt.

Aufgabe 3.6 (a) Zeige, dass die folgenden Funktionen:

$$\lceil \log n \rceil, \quad n^2, \quad 2^n \quad \text{und} \quad n!$$

raumkonstruierbar und dass alle bis auf $\lceil \log n \rceil$ zeitkonstruierbar sind.

- (b) Zeige, dass wenn $s_1 : \mathbb{N} \rightarrow \mathbb{N}$ und $s_2 : \mathbb{N} \rightarrow \mathbb{N}$ raumkonstruierbare Funktionen sind, dann sind auch die folgenden Funktionen raumkonstruierbar:

$$s_1(n) + s_2(n), \quad s_1(n) \cdot s_2(n), \quad 2^{s_1(n)} \quad \text{und} \quad s_1(n)^{s_2(n)}.$$

- (c) Zeige, dass wenn $t_1 : \mathbb{N} \rightarrow \mathbb{N}$ und $t_2 : \mathbb{N} \rightarrow \mathbb{N}$ zeitkonstruierbare Funktionen sind, dann sind auch die folgenden Funktionen zeitkonstruierbar:

$$t_1(n) + t_2(n), \quad t_1(n) \cdot t_2(n), \quad 2^{t_1(n)} \quad \text{und} \quad t_1(n)^{t_2(n)}.$$

Aufgabe 3.7 (a) Beweise Korollar 3.18.

- (b) Verwende geeignete Ergebnisse aus den Abschnitten 3.3 und 3.4, um Korollar 3.32 zu der folgenden Aussage zu verschärfen:

$$\text{NLINSPACE} \subset \text{PSPACE}.$$

- (c) Zeige, dass für jede Konstante $k > 1$ gilt:

$$\begin{aligned} L &\subset \text{DSPACE}((\log n)^k) \subset \text{DSPACE}((\log n)^{k+1}) \\ &\subset \text{POLYLOGSPACE} \subset \text{LINSPACE} \\ &\subset \text{DSPACE}(n^k) \quad \subset \text{DSPACE}(n^{k+1}) \\ &\subset \text{PSPACE} \quad \subset \text{DSPACE}(2^{k \cdot n}) \\ &\subset \text{DSPACE}(2^{(k+1)n}) \subset \text{EXPSPACE}. \end{aligned}$$

Aufgabe 3.8 Sei M eine DTM mit k Arbeitsbändern, die bei Eingaben der Länge n im Raum $s(n)$ und in der Zeit $t(n)$ arbeitet. Konstruiere eine einbändige DTM N , die M simuliert, so dass gilt:

- (a) $L(N) = L(M)$;
 (b) N arbeitet im Raum $s(n)$;
 (c) N arbeitet in der Zeit $(t(n))^2$.

Hinweis: Unterteile das Arbeitsband von N in k Spuren. Falls nötig, verwende den Satz über die lineare Beschleunigung, um Konstanten loszuwerden.

Aufgabe 3.9 Beweise Lemma 3.25: Für jede Menge $L \subseteq \Sigma^*$ gilt:

1. $L \in \text{E} \iff \text{Tally}(L) \in \text{P}$ und
2. $L \in \text{NE} \iff \text{Tally}(L) \in \text{NP}$.

Aufgabe 3.10 Zeige, dass $\text{NP} \subseteq \text{E}$ genau dann gilt, wenn für jede Menge L in NP die zugehörige Menge $\text{Tally}(L)$ in P ist.

Aufgabe 3.11 Beweise Lemma 3.36:

1. Aus $A \leq_m^p B$ folgt $\overline{A} \leq_m^p \overline{B}$, doch im Allgemeinen gilt $A \leq_m^p \overline{A}$ nicht.
2. Die Relation \leq_m^p ist sowohl reflexiv als auch transitiv, aber nicht antisymmetrisch.
3. P , NP und PSPACE sind \leq_m^p -abgeschlossen.
4. Gilt $A \leq_m^p B$ und ist $A \leq_m^p$ -hart für eine Komplexitätsklasse \mathcal{C} , so ist auch $B \leq_m^p$ -hart für \mathcal{C} .
5. Seien \mathcal{C} und \mathcal{D} beliebige Komplexitätsklassen. Ist $\mathcal{C} \leq_m^p$ -abgeschlossen und ist $B \leq_m^p$ -vollständig für \mathcal{D} , so gilt $\mathcal{D} \subseteq \mathcal{C}$ genau dann, wenn B in \mathcal{C} ist. Ist B insbesondere NP -vollständig, so gilt $\text{P} = \text{NP}$ genau dann, wenn B in P ist.

Aufgabe 3.12 (a) Beweise Satz 3.40: L und NL sind \leq_m^{\log} -abgeschlossen.

(b) Beweise, dass $\text{GAP}_{\text{acyclic}}$ in NL ist; siehe Lemma 3.47.

Aufgabe 3.13 Im Beweis des Satzes von Cook wurde die boolesche Formel $f(x) = F_x$ konstruiert; siehe Satz 3.49.

- (a)** Argumentiere, dass die Cook-Reduktion f in Polynomialzeit berechnet werden kann.
- (b)** Argumentiere, dass die Cook-Reduktion f sogar in logarithmischem Raum berechnet werden kann.

Aufgabe 3.14 (a) Beweise Lemma 3.53: Für jeden Graphen G und für jede Teilmenge $U \subseteq V(G)$ sind die folgenden Eigenschaften paarweise äquivalent:

- a) U ist eine Knotenüberdeckung von G .
- b) $\overline{U} = V(G) - U$ ist eine unabhängige Menge von G .
- c) $\overline{U} = V(G) - U$ ist eine Clique des co-Graphen von G , welcher definiert ist als der Graph mit der Knotenmenge $V(G)$ und der Kantenmenge

$$\{\{u, v\} \mid u, v \in V(G) \text{ und } \{u, v\} \notin E(G)\}.$$

- (b)** Beweise mit Hilfe von Lemma 3.53, dass die Probleme **Clique**, **IS** und **VC** paarweise \leq_m^p -äquivalent sind, d.h., für je zwei Probleme A und B , gewählt unter **Clique**, **IS** und **VC**, gilt $A \leq_m^p B$ und $B \leq_m^p A$.

- (c)** Beweise, dass die Probleme **Clique**, **IS** und **VC** in NP liegen.

Aufgabe 3.15 Satz 3.56 sagt, dass **3-Colorability** NP -vollständig ist. Beweise (3.13) und (3.14) aus dem Beweis dieses Satzes.

Aufgabe 3.16 Beweise, dass **2-Colorability** in P liegt.

Aufgabe 3.17 Satz 3.63 sagt, dass 3-DM NP-vollständig ist. Argumentiere formal, dass (3.17) aus diesem Beweis gilt:

$$\varphi \text{ ist erfüllbar} \iff R \text{ enthält ein tripartites Matching der Größe } \|U\|.$$

Hinweis: Verwende die während der Konstruktion der ternären Relation R in diesem Beweis gemachten Bemerkungen.

Aufgabe 3.18 Im Beweis von Satz 3.65 wird behauptet, dass das 3-DM-Problem eine Einschränkung des Problems SetPacking ist. Wieso ist diese Behauptung richtig?

Aufgabe 3.19 Beweise, dass S0S in NP liegt; siehe Satz 3.67.

Aufgabe 3.20 Beweise Fakt 3.69: GI ist in NP.

Aufgabe 3.21 Beweise die im Beweis von Satz 3.82 aufgestellte Behauptung: Für jede polynomiell längenbeschränkte Funktion g (d.h., es gibt ein $p \in \mathbb{I}\text{Pol}$, so dass $|g(x)| \leq p(|x|)$ für jedes x gilt) ist Π_g genau dann in P, wenn g in FP ist. Hier ist Π_g definiert durch

$$\Pi_g = \{\langle x, y \rangle \mid x \in D_g \text{ und } y \leq g(x)\}.$$

Hinweis: Die Richtung von rechts nach links ist trivial, und beim Beweis der Richtung von links nach rechts wird ein simpler Algorithmus für eine Binärsuche benutzt. Der Beweis kann in [Mil76] gefunden werden.

Aufgabe 3.22 Die Klasse der Komplemente von NP-Mengen wurde definiert als $\text{coNP} = \{L \mid \overline{L} \in \text{NP}\}$.

- (a) Nenne fünf Probleme deiner Wahl (und definiere sie formal), die \leq_m^p -vollständig für coNP sind.
- (b) Hat die Klasse $\text{NP} \cap \text{coNP}$ \leq_m^p -vollständige Mengen?
- (c) Was ist mit $\text{NP} \cup \text{coNP}$?

Aufgabe 3.23 Beweise die Äquivalenz der ersten beiden Aussagen von Satz 3.84: $P \neq UP \cap coUP$ gilt genau dann, wenn es surjektive Einwegfunktionen gibt.

Aufgabe 3.24 Definiere ausgehend von Definition 3.72 die folgende Relation auf Mengen:

$$A \cong_p B \iff A \text{ ist } p\text{-isomorph zu } B.$$

Beweise, dass \cong_p eine Äquivalenzrelation ist, d.h., sie erfüllt die folgenden Eigenschaften:

- *Reflexivität:* Für alle A gilt $A \cong_p A$.
 - *Symmetrie:* Für alle A und B gilt, dass $B \cong_p A$ aus $A \cong_p B$ folgt.
 - *Transitivität:* Für alle A , B und C gilt $A \cong_p C$, wenn sowohl $A \cong_p B$ als auch $B \cong_p C$ gelten.
-

Problem 3.1 (Beweis unterer Schranken durch Kreuzungsfolgen)

- (a) Entwirf eine DTM M mit einem Eingabeband und einem Arbeitsband, die die Menge $S = \{x2^{|x|} | x \in \{0,1\}^*\}$ in REALTIME entscheidet. Das heißt, der Kopf von M liest eine Eingabe z von links nach rechts, wobei er pro Schritt ein Feld weiterrückt, die Berechnung hält nach genau $|z|$ Schritten, und M akzeptiert z genau dann, wenn z in S ist. Beschreibe M sowohl informell als auch formal durch Angabe der Liste ihrer Turingbefehle.
- (b) Zeige, dass der Zeitaufwand einer jeden DTM mit nur einem Arbeitsband und ohne separates Eingabeband, die die obige Menge S löst, wenigstens quadratisch in der Eingabegröße ist.

Hinweis für (b): Sei M mit $L(M) = S$ eine wie oben angegebene DTM, und sei $x = uv$ ein Eingabewort der Länge n .

Eine Zustandsfolge von M , die mit $\text{cs}(u|v) = (s_1, s_2, \dots, s_n)$ bezeichnet wird, heißt *Kreuzungsfolge von $M(x)$ an der Grenze zwischen den Zellen u und v* , falls der Kopf von M diese Zellgrenze während der Berechnung von $M(x)$ genau n -mal kreuzt und M bei der i -ten Kreuzung im Zustand s_i ist.

Lemma 3.85. Falls $uv \in L(M)$ und $yz \in L(M)$ und $\text{cs}(u|v) = \text{cs}(y|z)$ ist, so ist $uz \in L(M)$ und $yv \in L(M)$.

Beweise Lemma 3.85 und zeige dann unter Benutzung von Lemma 3.85, dass $\text{time}_M(n) >_{\text{io}} c \cdot n^2$ für eine Konstante c gilt. Verwende dazu den Begriff der *kurzen Kreuzungsfolge bezüglich n* , welche als eine Kreuzungsfolge mit weniger als $n/\log q - 1$ Zuständen definiert ist, wobei $q > 1$ die Anzahl der Zustände von M ist. Zeige insbesondere, dass es weniger kurze Kreuzungsfolgen bezüglich n als Wörter der Länge $3n$ in S gibt.

Problem 3.2 (Primzahl-Problem)

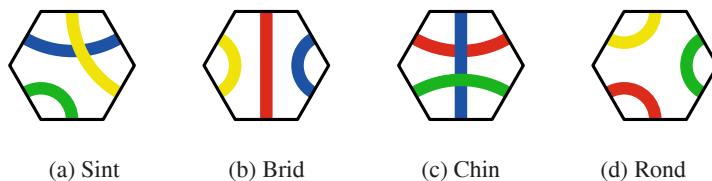
- (a) Beweise, dass das Primzahl-Problem (siehe Definition 7.4) in coNP liegt.
 (b) Was ist mit der Zugehörigkeit dieses Problems zu NP?

Hinweis: Der erste bekannte NP-Algorithmus für das Primzahl-Problem geht auf Pratt zurück, und den entsprechenden nichttrivialen Beweis findet man in [Pra75].

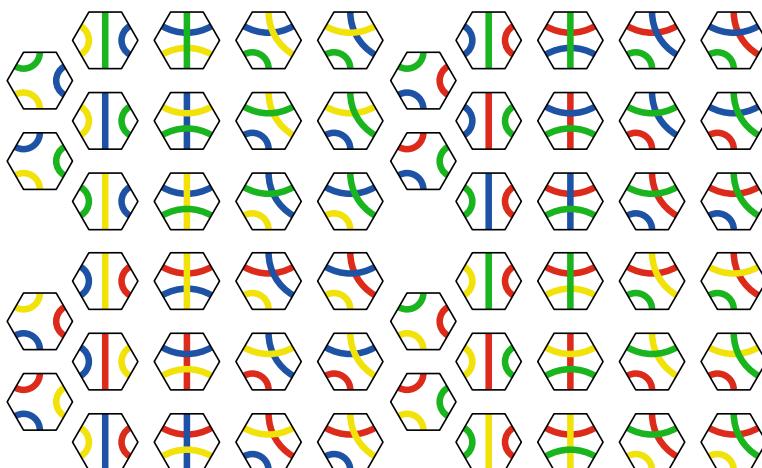
Abschnitt 7.2 liefert stärkere als die oben angegebenen Resultate, nämlich insbesondere das herausragende Ergebnis von Agrawal, Kayal und Saxena [AKS02], die zeigten, dass das Primzahl-Problem sogar in P liegt, siehe Satz 7.27 und auch Problem 7.2.

Problem 3.3 (TantrixTM-Rotation-Puzzle-Problem)

Das von dem Neuseeländer Mike McManaway erfundene TantrixTM-Spiel erfreut sich größter Beliebtheit. Es wird mit hexagonalen Teilen in der Ebene gespielt, die jeweils drei verschieden gefärbte Linien in unterschiedlichen Mustern haben. Die vier Teiltypen heißen *Sint*, *Brid*, *Chin* und *Rond* und sind in Abbildung 3.7 dargestellt. Für die Linien sind vier Farben möglich: rot, blau, gelb und grün. Insgesamt gibt es somit die in Abbildung 3.8 dargestellten 56 TantrixTM-Teile mit vier Farben.

**Abb. 3.7.** Die TantrixTM-Teile

Teile dieser Art werden fest in der Ebene angeordnet und bleiben in ihrer Position. Das Ziel des Spiels⁹ besteht darin, diese Teile so zu drehen, dass die Linien an je zwei benachbarten Kanten immer dieselbe Farbe haben; Abbildung 3.9 zeigt ein Beispiel. Gemäß dieser Regel wird das *TantrixTM-Rotation-Puzzle-Problem mit vier Farben* (kurz 4-TRP) wie folgt als ein Entscheidungsproblem definiert: Gegeben eine Anordnung von (endlich vielen) TantrixTM-Teilen in der Ebene, ist es möglich, diese so zu drehen, dass die Linien an je zwei benachbarten Kanten immer dieselbe Farbe haben?

**Abb. 3.8.** Sämtliche 56 TantrixTM-Teile mit vier Farben

- (a)** Zeige, dass 4-TRP NP-vollständig ist.

⁹ Es wird hier nur eine von mehreren möglichen Spielvarianten beschrieben. Und anders als im ursprünglichen TantrixTM-Spiel sei hier erlaubt, dass es in der Anordnung von Teilen Löcher geben darf.

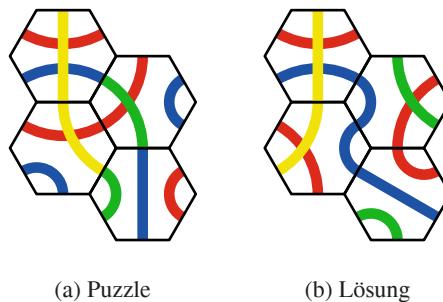


Abb. 3.9. Ein TantrixTM-Puzzle und seine Lösung

- (b) 3-TRP sei das entsprechend definierte Problem, bei dem nur drei Farben erlaubt sind: rot, blau und gelb. Demzufolge gibt es die in Abbildung 3.10 gezeigten 14 Teile. Zeige, dass 3-TRP NP-vollständig ist.

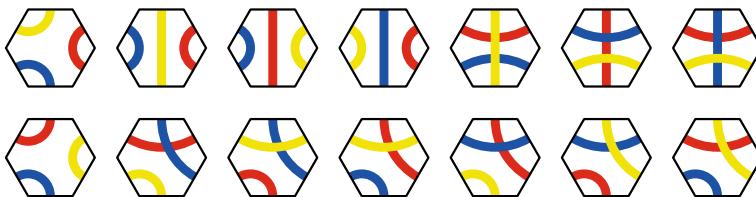


Abb. 3.10. Sämtliche 14 TantrixTM-Teile mit drei Farben

- (c) 2-TRP sei nun das entsprechend definierte Problem, bei dem nur noch zwei Farben erlaubt sind: rot und blau. Demzufolge gibt es die in Abbildung 3.11 gezeigten 8 Teile. Zu beachten ist hier erstens, dass nun natürlich nicht mehr verlangt werden kann, dass die drei Linien auf den Teilen unterschiedlich gefärbt sind. Zweitens ist nur die Farbsequenz eines Teils (im Uhrzeigersinn) von Belang. Beispielsweise ist ein Teil mit der Farbsequenz rot-rot-rot-rot-rot-rot durch unterschiedliche Teiltypen darstellbar (in Abbildung 3.11 ist das *Sint* gewählt worden) und in allen sechs Orientierungen identisch. Diese Farbsequenz wird daher als nur eine Lösung betrachtet, und redundante Lösungen werden ignoriert. Ebenso reduzieren wir die Lösungen für die Farbsequenz blau-blau-blau-blau-blau-blau von sechs auf eins und die Farbsequenzen rot-blau-blau-rot-blau-blau (bzw. blau-rot-rot-blau-rot-rot) von sechs auf drei.

Ist auch 2-TRP NP-vollständig?

Hinweis: Holzer and Holzer [HH04] zeigten, dass 4-TRP NP-vollständig ist, indem sie vom Erfüllbarkeitsproblem für planare Schaltkreise mit UND- und

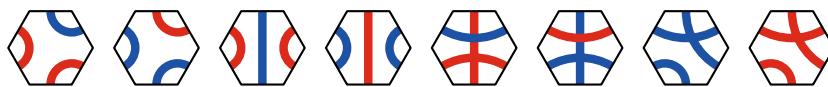


Abb. 3.11. Sämtliche 8 Tantrix™-Teile mit zwei Farben

NICHT-Gattern reduzierten. Dabei benutzten sie Schaltkreiskonstruktionen von Goldschlager [Gol77] und McColl [McC81]. Baumeister und Rothe [BR07] modifizierten diese Reduktion so, dass sie „parsimonious“ ist. Die Lösung für die Teilaufgaben (b) und (c) findet man in [BR08]. Die Abbildungen 3.7 bis 3.11 stammen aus [BR08].

3.8 Zusammenfassung und bibliographische Notizen

Allgemeine Bemerkungen: Es gibt viele sehr gute Lehrbücher und Monographien über Komplexitätstheorie. Es gibt viele sehr gute Bücher über Kryptologie. Das vorliegende Buch soll diese Bücher – deren jedes sich entweder auf die Komplexitätstheorie oder die Kryptologie konzentriert – nicht ersetzen, sondern dadurch ergänzen, dass es die Verflechtung dieser beiden Gebiete betont und einen einheitlichen Zugang wählt.

In der Komplexitätstheorie sind die folgenden Bücher bereits Klassiker oder dabei, welche zu werden: die Bücher von Balcázar, Díaz und Gabarró [BDG95, BDG90], Bovet und Crescenzi [BC93], Du und Ko [DK00], Garey und Johnson [GJ79], L. Hemaspaandra und Ogiwara [HO02], Homer und Selman [HS01], Papadimitriou [Pap95] und Papadimitriou und Steiglitz [PS82], Reischuk [Rei90], Wagner und Wechsung [WW86, Wec00] sowie die Bücher von Wegener [Weg87, Weg03]. Vollmer [Vol99] hat ein sehr nützliches Buch über Schaltkreiskomplexität geschrieben, ein Thema, das hier nicht behandelt wird. Brandstädt et al. [BLS99] geben einen sehr umfangreichen Überblick über Graphklassen und die algorithmische Komplexität der auf ihnen definierten Graphenprobleme; gewissermaßen kann dieses Buch als ein Nachfolger des Klassikers von Golumbic [Gol80] verstanden werden. In [Rot07a] werden bestimmte Algorithmen in der Komplexitätstheorie behandelt.

Spezielle Bemerkungen: Die Anfänge der Komplexitätstheorie werden durch die frühen Resultate von Hartmanis, Lewis und Stearns markiert. Ihre bahnbrechenden Arbeiten [HS65, SHL65, HLS65, LSH65] betreffen die Robustheit des Modells der Mehrband-Turingmaschine, führen die Komplexitätsmaße Zeit und Raum ein und beschreiben ihre grundlegenden Eigenschaften. Ihnen sind insbesondere die Sätze über lineare Raumkompression und Beschleunigung und die Hierarchiesätze für Zeit und Raum zu verdanken; Satz 3.19, die stärkste bekannte Version des Zeit-Hierarchiesatzes, wurde von Hennie und Stearns [HS66] bewiesen. Ursprünglich wurde ihr Resultat folgendermaßen formuliert (wobei die Konstruierbarkeitsvoraussetzungen hier im Sinne einer einfacheren Lesbarkeit weggelassen werden: Ist $t_1 \log t_1 \prec_{\text{io}} t_2$, so gilt $\text{DTIME}(t_2) \not\subseteq \text{DTIME}(t_1)$). Im Gegensatz dazu drückt Satz 3.19

ihr Resultat wie folgt aus: Ist $t_1 \prec_{\text{io}} t_2$, so gilt $\text{DTIME}(t_2 \log t_2) \not\subseteq \text{DTIME}(t_1)$. Diese beiden Formulierungen sind äquivalent; diese Einsicht verdankt der Autor Gerd Wechsung, der ihm großzügigerweise seine persönlichen Notizen zur Verfügung stellte, siehe auch [Wec00, WW86].

Stearns [Ste90] schrieb eine hübsche Abhandlung über die intellektuellen Abenteuer, die Begeisterung und Faszination dieser frühen Jahre. In Anerkennung ihrer Arbeit, die die Fundamente des Gebiets der Komplexitätstheorie legte, erhielten Juris Hartmanis und Richard Stearns 1993 den hochkarätigen Turing Award.

Die elegante Theorie der abstrakten Komplexitätsmaße, die man nun als Blumsche Komplexitätsmaße bezeichnet, wurde von Blum [Blu67] entwickelt. Im Jahr 1995 erhielt Manuel Blum ebenfalls den Turing Award, in Anerkennung seiner Beiträge zu den Grundlagen der Komplexitätstheorie und ihrer Anwendungen in der Kryptographie und Programmverifikation.

Books „Upward-Separation“-Technik [Boo74], die u.a. Satz 3.26 hervorbrachte, wurde in zweierlei Hinsicht verstärkt: bezüglich der Sprache, die die Separation bezeugt, und bezüglich des Anwendungsbereiches auf andere Komplexitätstklassen als NP und NE. Hartmanis, Immerman und Sewelson [Har83b, HIS85] bewiesen, dass $\text{NE} = \text{E}$ genau dann gilt, wenn jede *dünne* Sprache aus NP in P liegt.¹⁰ Zu diesem Zweck entwickelten sie eine clevere Codierung von dünnen Mengen in tally Mengen. Buhrman, E. Hemaspaandra und Longpré [BHL95] entdeckten eine noch mächtigere tally Codierung der dünnen Mengen. Unter Verwendung dieser stärkeren Codierung erweiterten Rao, Rothe und Watanabe [RRW94] das Resultat von Hartmanis et al. [Har83b, HIS85] auf eine Reihe von Paaren exponentiell verwandter, von NP und NE verschiedener Komplexitätstklassen. Das Hauptergebnis von Rao et al. [RRW94] ist eine allgemeine hinreichende Bedingung dafür, wann eine Separation nach oben durch dünne Mengen möglich ist. Insbesondere zeigt diese hinreichende Bedingung, dass FewP eine dünne Menge, die nicht in P liegt, genau dann enthält, wenn $\text{FewE} \neq \text{E}$ gilt, wobei FewE und E die Exponentialzeit-Analoga von FewP und P sind. Dieses Resultat widerlegt eine Vermutung von Allender [All91], derzu folge sich FewP in geeigneten relativierten Welten einer Separation nach oben widersetzt. Es ist noch offen, ob ein analoges Resultat für UP gilt. Eine Reihe von Resultaten zeigt die Grenzen der „Upward-Separation“-Technik auf, darunter die Arbeiten von Allender [All91] und von L. Hemaspaandra und Jha [HJ95]; siehe [All91, HJ95, RRW94] für verwandte Ergebnisse und eine umfangreichere Liste von Literaturhinweisen.

Satz 3.29, der eine quadratische obere Schranke für die Kosten liefert, Nichtdeterminismus durch Determinismus zu simulieren, geht auf Savitch [Sav70] zurück.

Die Komplexitätstheorie entsprang dem Wunsch, effiziente (oder „machbare“) Berechnung und ihre Grenzen zu verstehen. Ganz zentral in der Komplexitätstheorie sind daher die fundamentalen Komplexitätstklassen P und NP, deterministische und nichtdeterministische Polynomialzeit. Cobham [Cob64] und Edmonds [Edm65]

¹⁰ Zur Erinnerung: Eine Sprache heißt *dünn*, falls sie für jede Länge höchstens polynomiel viele Wörter enthält; siehe auch Definition 5.58. Also ist jede tally Menge dünn, da tally Mengen höchstens ein Wort pro Länge haben.

erkannten als die Ersten, dass die Klasse P die vernünftigste formale Darstellung des informalen Begriffs der machbaren Berechnung ist. Interessanterweise sprach Gödel das Problem, „effiziente Berechnung“ formal zu definieren, bereits 1956 an. Sein wieder aufgefunder Brief an von Neumann betont die Bedeutung der Schrittzahl, die eine Turingmaschine zur Lösung eines Problems benötigt. Außerdem gab er zwei polynomiale Zeitschranken an, Linearzeit und quadratische Zeit, als Beispiele für „effiziente Berechnung“ im Gegensatz zu exponentiellen Zeitschranken von Algorithmen, die mit brutaler Gewalt (englisch „by brute force“) vorgehen. Hartmannis [Har89] und Sipser [Sip92] gaben Überblicke über die Geschichte der P-versus-NP-Frage, in denen jeweils auch Gödels Brief erörtert wird.

Cook begründete die Theorie der NP-Vollständigkeit mit seinem Beweis von Satz 3.49, welcher das erste NP-Vollständigkeitsresultat darstellt. Für seine einflussreiche Arbeit [Coo71] erhielt er 1982 den Turing Award. Sein Resultat, dass SAT NP-vollständig ist, wurde unabhängig von Levin [Lev73] entdeckt. Cook bewies auch Satz 3.51: 3-SAT ist NP-vollständig.

In der Folge von Cooks wegweisendem Resultat entwickelte sich die Erkundung der Grenzen und Eigenschaften der Klasse der NP-vollständigen Probleme zu einem der aktivsten und wichtigsten Forschungsgebiete in der Informatik. Am bemerkenswertesten ist hier die Arbeit von Karp [Kar72], der die Technik einführt, die heute die Standardmethode zum Nachweis der NP-Vollständigkeit von Problemen bezüglich der \leq_m^p -Reduzierbarkeit ist und die zur Klassifikation von inzwischen Tausenden NP-vollständiger Probleme geführt hat. Diese Probleme betrachtet man als algorithmisch widerspenstig oder unzugänglich. Für diese Leistungen und für seine Beiträge zur Algorithmik, einschließlich der Entwicklung effizienter Algorithmen für Netzwerkfluss- und andere kombinatorische Optimierungsprobleme, erhielt Karp 1985 den Turing Award. Die Sätze 3.54, 3.63, 3.65 und 3.67 gehen auf ihn zurück.

Der Beweis von Satz 3.56 ist aus Stockmeyers Arbeit [Sto73], siehe auch Garey, Johnson und Stockmeyer [GJS76]. Satz 3.58 ist Garey, Johnson und Tarjan zu verdanken (wie in [GJ79] zitiert); der hier präsentierte Beweis geht auf Kaplan und R. Shamir [KS94] zurück. Die Literatur zu NP-Vollständigkeitsresultaten ist so umfangreich und das Thema wird in so vielen Büchern behandelt, dass es nicht sinnvoll ist, an dieser Stelle weiter ins Detail zu gehen. Eine der besten Quellen zur Theorie der NP-Vollständigkeit ist immer noch der Klassiker von Garey und Johnson [GJ79]; siehe auch Johnsons fortlaufende NP-Vollständigkeitskolumnen [Joh81].

Für viele NP-vollständige Probleme ist bekannt, dass sie für geeignet eingeschränkte Instanzen effizient gelöst werden können. So taucht die Frage auf, wo genau die Grenze zwischen leicht lösbarer Instanzen und harten Instanzen liegt. Zur Illustration werden nun einige solcher Resultate für das Beispiel des Domatische-Zahl-Problems angegeben:¹¹ Gemäß der im Beweis von Satz 3.58 präsentierten Konstruktion bleibt dieses Problem sogar dann NP-vollständig, wenn der gegebene Graph zu bestimmten speziellen Klassen perfekter Graphen gehört, einschließlich

¹¹ Graphentheoretische Begriffe und spezielle Graphklassen, die hier nicht definiert werden, findet man z.B. in der Monographie von Brandstädt et al. [BLS99] oder in dem Klassiker von Golumbic [Gol80].

der Circular-arc-Graphen (siehe auch [Bon85]), Split-Graphen (welche insbesondere die chordalen und co-chordalen Graphen beinhalten) und der bipartiten Graphen (zu denen insbesondere die Komparabilitätsgraphen gehören). Im Gegensatz dazu ist DNP für bestimmte andere Graphenklassen in Polynomialzeit lösbar, einschließlich der stark chordalen Graphen (welche speziell die Intervallgraphen und die Pfadgraphen umfassen) [Far84] und der strikten Circular-arc-Graphen [Bon85].

Weiter sind die Approximierbarkeitseigenschaften von praktisch allen wichtigen NP-vollständigen Problemen tief schürfend untersucht worden; siehe beispielsweise die Bücher [ACG⁺03, Vaz03, Pap95]. Um ein solches Resultat für das Domatische-Zahl-Problem anzugeben: Feige, Halldórsson, Kortsarz und Srinivasan [FHKS02] zeigten, dass jeder Graph G mit n Knoten eine domatische Zerlegung in $(1 - o(1))(min-deg(G) + 1)/\ln n$ Mengen besitzt, die in Polynomialzeit gefunden werden kann. Also gibt es einen $((1 + o(1)) \ln n)$ -Approximationsalgorithmus für die domatische Zahl $\delta(G)$. Dies ist eine knappe Schranke, da die domatische Zahl nicht in einem Faktor von $(1 - \varepsilon) \ln n$ approximiert werden kann, wobei $\varepsilon > 0$ eine beliebig fixierte Konstante ist, außer wenn $NP \subseteq \text{DTIME}(n^{\log \log n})$ gelten würde.

Die Entwicklung der Theorie der NP-Vollständigkeit zog die Suche nach Vollständigkeitsresultaten auch für andere Komplexitätsklassen nach sich, einschließlich der Identifizierung \leq_m^{\log} -vollständiger Probleme für P und NL. Satz 3.43 wurde unabhängig von Savitch [Sav73] und von Jones [Jon75] bewiesen, die somit das erste \leq_m^{\log} -vollständige Problem für NL fanden, nämlich GAP. Die Lemmata 3.46 und 3.48, aus denen die coNL-Vollständigkeit von 2-SAT folgt, gehen auf Jones, Lien und Laaser [JLL76] zurück. Satz 3.45 folgt aus diesem Resultat mittels der Gleichheit $NL = coNL$, welche unabhängig von Immerman [Imm88] und Szelepcsenyi [Sze88] bewiesen wurde; siehe Satz 3.33 und sein Korollar 3.34. Probleme, die \leq_m^{\log} -vollständig für P sind, findet man etwa in [Coo74, JL76]; siehe auch Satz 5.72 in Abschnitt 5.6. Logarithmisch raumbeschränkte Reduzierbarkeiten wie zum Beispiel \leq_m^{\log} wurden von Ladner und Lynch [LL76] studiert.

Satz 3.68 ist Ladner [Lad75] zu verdanken. Er stellt das komplexitätstheoretische Analogon zur Lösung des von Post in der Berechenbarkeitstheorie aufgeworfenen Problems dar, welches fragt, ob es mehr als zwei rekursiv aufzählbare Turinggrade gibt. Das Problem von Post wurde 1956 unabhängig von Friedberg und Muchnik gelöst. Der Friedberg–Muchnik–Satz sagt, dass es rekursiv aufzählbare Mengen gibt, die weder entscheidbar noch vollständig für RE sind, die Klasse der rekursiv aufzählbaren Mengen.

Das Graphisomorphie-Problem (siehe Definition 2.49) ist in der Komplexitätstheorie eingehend untersucht worden. Dennoch hat es sich bisher noch jedem Versuch der Klassifizierung entzogen. GI ist einer der prominentesten Kandidaten für ein Problem, das weder in P liegt noch NP-vollständig ist, und Kapitel 6 wird einige Indizien liefern, die diesen Standpunkt untermauern. Die beste Quelle für Resultate und den (damals aktuellen) Wissensstand über dieses Problem ist das Buch [KST93] von Köbler, Schöning und Torán.

Selbstreduzierbarkeit ist in einer Vielzahl von Formen in Erscheinung getreten und viele verschiedene Begriffe der Selbstreduzierbarkeit sind eingeführt und ein-

gehend untersucht worden. Aus der noch immer wachsenden Masse von Resultaten zur Selbstreduzierbarkeit seien hier als besonders erwähnenswert die Arbeiten von Schnorr [Sch76, Sch79], A. Meyer und Paterson [MP79], Balcázar [Bal90], Buhrman, van Helden und Torenvliet [BT96, BvHT93] sowie von E. Hemaspaandra, Naik, Ogihsara und Selman [Sel88a, Sel79, Sel82a, Sel82b, HNOS96] herausgegriffen. Für einen Überblick über frühe Resultate zur Selbstreduzierbarkeit seien die Artikel von Goldsmith, Joseph und Young [GJY87, JY90] erwähnt.

Die als „*search reducing to decision*“ bekannte Eigenschaft ist ein Beispiel eines Themas, für das der Begriff der Selbstreduzierbarkeit zentral ist, siehe [HNOS96] und die darin zitierten Literaturhinweise. Darunter versteht man, dass wenn ein Entscheidungsproblem effizient lösbar und selbstreduzierbar ist, dann kann eine tatsächliche Lösung effizient konstruiert werden: Man reduziert in diesem Sinn die Suchversion des Problems auf seine Entscheidungsversion. Nehmen wir z.B. an, das Problem GI wäre effizient lösbar (auch wenn dies ja ein bekanntermaßen offenes Problem ist). GI ist auch selbstreduzierbar (siehe Satz 3.71), und ein Isomorphismus zwischen den gegebenen Graphen könnte dann effizient konstruiert werden.

Gál, Halevi, Lipton und Petrank [GHL99] wählten einen ähnlichen, aber etwas anderen Ansatz, indem sie eine Verbindung herstellten zwischen der Komplexität, einer partielle Lösung eines harten NP-Problems zu finden, und der Komplexität, eine vollständige Lösung dieses Problems zu finden. Diese Eigenschaft könnte man „*complete search reducing to partial search*“ taufen. Insbesondere zeigten sie für mehrere NP-Probleme A , dass für eine gegebene Instanz x in A das Berechnen eines kleinen Teils einer Lösung für x nicht leichter ist als das Berechnen einer vollständigen Lösung für x . Sind beispielsweise zwei isomorphe Graphen gegeben, so ist die Berechnung von ungefähr logarithmisch vielen Paaren von Knoten, die ein vollständiger Isomorphismus φ zwischen diesen Graphen aufeinander abbildet, ebenso schwer wie die Berechnung von φ selbst [GHL99]. Große, Rothe und Wechsung [GRW02] verschärfen dieses Resultat optimal insofern, als sie zeigten, dass die Berechnung auch nur eines einzelnen Paares von Knoten, die durch einen vollständigen Isomorphismus φ zwischen zwei isomorphen Graphen aufeinander abgebildet werden, ebenso schwer ist wie das Berechnen von φ selbst; siehe auch Problem 5.3. Der Beweis dieses Resultats ist vom Beweis der Selbstreduzierbarkeit von GI inspiriert.

Die Isomorphievermutung von Berman und Hartmanis [BH77] ist eine der Fragen, die in der Komplexitätstheorie am intensivsten studiert wurden. Ihre Arbeit zu p-isomorphen Mengen in NP initiierte mehrere Forschungsrichtungen und viele weitere Resultate. Die Bedeutung dieser Vermutung und der Ergebnisse, die sie hervorbrachte, kann schon allein an der Anzahl der Überblicks- und Originalarbeiten zu diesem Gegenstand ermessen werden, zu denen u.a. die Arbeiten von Mahaney [Mah86, Mah89], P. Young [You90, You92], Kurtz, Mahaney und Royer [KMR90, KMR87, KMR88], L. Hemaspaandra, Ogihsara und Watanabe [HOW92] und Arvind et al. [AHH⁺93] gehören. Insbesondere kulminierte die Arbeit zur Frage, ob NP dünne \leq_m^p -vollständige Mengen hat oder nicht, in Mahaneys [Mah82] Lösung einer anderen Vermutung von Berman und Hartmanis, die hier als Satz 3.75 angegeben ist. Sein Resultat baut auf der Arbeit von Berman [Ber78] auf, der be-

wies, dass es unter der Annahme $P \neq NP$ keine tally coNP-harten Mengen gibt, und weiter auf der Arbeit von Fortune [For79], der bewies, dass es unter der Annahme $P \neq NP$ keine dünnen coNP-harten Mengen gibt. Satz 3.76 geht auf Hartmanis und Mahaney [HM80] zurück. Abschnitt 5.9 liefert weitere Details zu Resultaten, die Satz 3.75 nachfolgten, und gibt insbesondere Ogihara und Watanabes [OW91] Verbesserung von Satz 3.75 auf Reduktionen an, die allgemeiner als \leq_m^p sind.

Die Frage, ob es dünne vollständige Mengen geben kann oder nicht, wurde auch für andere Komplexitätsklassen als NP und für andere Reduzierbarkeiten als \leq_m^p untersucht. Für Exponentialzeitklassen bewies A. Meyer (siehe [BH77]) durch Diagonalisierung, dass weder E noch NE dünne \leq_m^p -vollständige Mengen hat. Auf Grund von Indizien, die denen ähneln, welche ihn zu Vermutung 3.73 führten, warf Hartmanis [Har78] die analoge Frage auch für die Klassen NL und P auf, wobei der Begriff des p-Isomorphismus durch den des in logarithmischem Raum berechenbaren Isomorphismus zu ersetzen ist. Auf wichtigen Anfangsresultaten von Ogihara [Ogi95] aufbauend, gelang Cai und Sivakumar schließlich die Lösung der Vermutungen von Hartmanis sowohl für P als auch für NL:

- Ist $L \neq P$, so ist keine dünne Menge \leq_m^{\log} -vollständig in P; siehe [CS99].
- Ist $L \neq NL$, so ist keine dünne Menge \leq_m^{\log} -vollständig in NL; siehe [CS00].

Weitere Resultate in Bezug auf komplexitätsbeschränkte Isomorphismen und die Isomorphievermutung wurden von Allender [All88], Ganesan und Homer [GH92], Hartmanis [Har83a], Homer und Selman [HS89], Ko, Long und Du [KLD86], Mahaney und P. Young [MY85] und Watanabe [Wat91] erzielt.

Die Isomorphievermutung wurde auch in Hinsicht auf Relativierungen intensiv untersucht. Indem sie vorherige Konstruktionen von Kurtz (aus einer unveröffentlichten Arbeit) und Rackoff [Rac82] kombinierten, konstruierten Hartmanis und L. Hemaspaandra [HH91] ein Orakel, relativ zu dem die Isomorphievermutung nicht gilt und keine Einwegfunktionen existieren, womit sie ein relativiertes Gegenbeispiel zu Vermutung 3.80 fanden. Kurtz, Mahaney und Royer [KMR89] bewiesen, dass die Isomorphievermutung relativ zu einem zufälligen Orakel nicht gilt, und in [KMR87] gaben sie weitere relativierte Resultate an. Ausgehend von vorherigen Ergebnissen von Goldsmith und Joseph [GJ86] konstruierten Fenner, Fortnow und Kurtz [FFK92] ein Orakel, relativ zu dem die Isomorphievermutung gilt. Sie fragten, ob es eine relativierte Welt gibt, in der die Isomorphievermutung und dennoch $P \neq UP$ (äquivalent zur Existenz von Einwegfunktionen) gilt. Diese Frage wurde von J. Rogers [Rog97] beantwortet, der nämlich ein relativiertes Gegenbeispiel zu Vermutung 3.79 lieferte.

Valiant [Val76] führte die Klasse UP ein, siehe Definition 3.81. In ihrer Arbeit über „P-printable“ Mengen¹² und über dünne Mengen in P verallgemeinerten Allender und Rubinstein [AR88, All86] die Klasse UP zur Klasse FewP. Eine Sprache ist in FewP, falls sie von einer NPTM akzeptiert wird, die niemals mehr als eine polynomiale Anzahl akzeptierender Pfade hat. Satz 3.82 sagt insbesondere, dass injektive (also „one-to-one“) Einwegfunktionen genau dann existieren, wenn $P \neq UP$

¹² Informal gesagt ist eine Menge P-*printable*, falls alle ihre Elemente bis zu einer gegebenen Länge in Polynomialzeit ausgegeben werden können.

gilt. Analog dazu bewies Allender [All86, All85], dass „polynomial-to-one“ Einwegfunktionen genau dann existieren, wenn $P \neq \text{FewP}$ gilt; siehe [RH02] für eine Anzahl verwandter Resultate. Das Studium von Einwegfunktionen in der Komplexitätstheorie wurde von Grollmann und Selman [GS88] und anderen initiiert. Der in Definition 3.78 eingeführte Begriff und die Äquivalenz der ersten beiden Aussagen in den Sätzen 3.82 und 3.84 sind Grollmann und Selman zu verdanken und, unabhängig, Berman [Ber77] und Ko [Ko85]. Die Äquivalenz der ersten und dritten Aussage dieser beiden Sätze wurde jeweils von Hartmanis und L. Hemaspaandra [HH88] gezeigt; siehe auch [RH02]. Korollar 3.83 ist ebenfalls aus [HH88]. Außerdem präsentiert diese Arbeit ein Orakel, relativ zu dem UP keine vollständige Menge hat, und ein weiteres Orakel, relativ zu dem $P \neq \text{UP} \neq \text{NP}$ gilt und UP eine vollständige Menge besitzt.

Dass $P \neq \text{UP} \cap \text{coUP}$ die dritte Aussage von Satz 3.84 impliziert, ist das UP-Analogon des Satzes von Borodin und Demers: Aus $P \neq \text{NP} \cap \text{coNP}$ folgt die Existenz einer Menge S in P von erfüllbaren booleschen Formeln, so dass keine in Polynomialzeit berechenbare Funktion eine erfüllende Belegung für jede Formel in S ausgeben kann. Einige verwandte Resultate wurden von Fenner, Fortnow, Naik und J. Rogers [FFNR96] sowie, unabhängig, von L. Hemaspaandra, Rothe und Wechsung [HRW97a, HRW97b, RH02] erzielt. Beispielsweise wird in [HRW97a] die Klasse $\text{EASY}_{\vee}^{\vee}$ definiert, die alle NP-Mengen L enthält, für die jede NPTM, die L akzeptiert, stets (d.h. bei jeder Eingabe) in Polynomialzeit berechenbare Zertifikate hat. Man kann zeigen, dass $\text{EASY}_{\vee}^{\vee} \subseteq P$ gilt. In dieser Schreibweise hat der Satz von Borodin und Demers die folgende Gestalt: Ist $P \neq \text{NP} \cap \text{coNP}$, so gilt $P \neq \text{EASY}_{\vee}^{\vee}$. Außerdem werden Charakterisierungen von $\text{EASY}_{\vee}^{\vee}$ und verwandten Klassen in Bezug auf die Kolmogorov-Komplexität gegeben. In [RH02] werden die UP- und FewP-Analoga von $\text{EASY}_{\vee}^{\vee}$ untersucht.

In diesem Zusammenhang untersuchten L. Hemaspaandra und Rothe [HR00, RH02] die Frage, ob die Existenz von Einwegpermutationen durch eine geeignete Separation von Komplexitätsklassen charakterisiert werden kann, eine Frage, die auch in [GS88] gestellt wurde. Eine Einwegpermutation ist eine totale, injektive und surjektive Einwegfunktion. Diese Frage wurde schließlich von Homan und Thakur [HT02, HT03b] beantwortet: Einwegpermutationen existieren genau dann, wenn $P \neq \text{UP} \cap \text{coUP}$ gilt.

Zum Studium der Komplexitätsklassen von Mengen, wie etwa P und NP , gehört das analoge Studium der Komplexitätsklassen von Funktionen. Beispielsweise ist FP das funktionale Analogon von P . In seinen einflussreichen Arbeiten [Val79a, Val79b] führte Valiant die Funktionenklassen $\#P$ und $\#P_1$ ein, die die Komplexität des Zählens von NP-Lösungen einfangen: $\#P$ ist die Klasse der Funktionen, die die Anzahl der Lösungen von NP-Problemen angeben (siehe auch Abschnitt 6.4), und $\#P_1$ ist die Klasse der Funktionen, die die Anzahl der Lösungen von tally NP-Problemen angeben. Der einzige Unterschied zwischen $\#P$ - und $\#P_1$ -Funktionen ist also, dass die Eingaben der erstenen in Binärdarstellung und die der letzteren in Unärdarstellung vorliegen.

Betrachte zum Beispiel das Problem, die Permanente einer gegebenen Matrix zu berechnen. Bezeichnet $a_{i,j}$ den (i, j) -Eintrag einer $n \times n$ Matrix A über den ganzen

Zahlen, so ist die *Permanente von A* definiert als $\text{perm}(A) = \sum_{\pi \in \mathfrak{S}_n} \prod_{i=1}^n a_{i,\pi(i)}$. Valiant [Val79a] bewies, dass das Berechnen der Permanente $\#P$ -vollständig ist, d.h., $\text{perm} \in \#P$ und $\#P \subseteq \text{FP}^{\text{perm}}$, wobei perm als ein Funktionenorakel verwendet wird. Im Gegensatz dazu kann die Determinante einer Matrix mit dem Eliminierungsverfahren von Gauß in Polynomialzeit berechnet werden. Neuere Resultate über die Härte der Permanente wurden von Cai, Pavan und Sivakumar [CPS99] erzielt.

$\#P_1$ enthält ebenso interessante natürliche Probleme, etwa das Self-Avoiding-Walk-Problem (siehe z.B. Welsh [Wel93]), ein klassisches Problem der statistischen Physik und Polymerchemie. Das Self-Avoiding-Walk-Problem besteht darin, für eine unär gegebene Zahl n im zweidimensionalen Gitter die Anzahl der im Koordinatenursprung beginnenden „self-avoiding walks“ der Länge n zu berechnen. Valiant [Val79b] fragte, ob dieses Problem $\#P_1$ -vollständig ist. Diese Frage ist noch immer offen. Liśkiewicz, Ogiara und Toda [OT01, LOT03] beantworteten sie teilweise, indem sie zeigten, dass bestimmte Varianten des Self-Avoiding-Walk-Problems in zweidimensionalen Gittergraphen und in Hyperwürfel-Graphen $\#P$ -vollständig sind.

Die wichtigste Frage bezüglich $\#P$ und $\#P_1$ ist natürlich, ob diese Klassen in FP enthalten sind oder nicht. Köbler [Köb89] zeigte, dass $\#P = \text{FP}$ äquivalent zu $P = \text{PP}$ ist, wobei die Klasse PP („Probabilistic Polynomial Time“) später in Kapitel 6 definiert und studiert wird. Da PP vermutlich nicht gleich P ist, ist es ebenso sehr unwahrscheinlich, dass jede $\#P$ -Funktion in FP berechnet werden kann. Weitere Resultate über die wichtige Klasse $\#P$ und andere funktionale Komplexitätsklassen findet man in den exzellenten Übersichtsartikeln von Selman [Sel94] und Fortnow [For97], in Kapitel 9 von Wechsungs Buch [Wec00] und auch in Kapitel 6.

Wenn $\#P_1 \subseteq \text{FP}$ gilt, dann sind alle tally NP-Mengen in P, woraus mit Books Upward-Separation-Resultat (siehe Satz 3.26) sofort $\text{NE} = \text{E}$ folgt. Goldsmith, Ogiara und Rothe [GOR98, GOR00] bewiesen, dass aus der Annahme $\#P_1 \subseteq \text{FP}$ sogar noch unwahrscheinlichere Kollapse von Komplexitätsklassen folgen. Insbesondere zeigten sie, dass aus $\#P_1 \subseteq \text{FP}$ die Gleichheit $P = \text{BPP}$ und die Inklusion $\text{PH} \subseteq \oplus P$ folgen, wobei PH die Polynomialzeit-Hierarchie ist (siehe Definition 5.29 in Abschnitt 5.2) und BPP und $\oplus P$ in Kapitel 6 definiert werden. Außerdem zeigten sie, dass $\#P_1 \subseteq \text{FP}$ genau dann gilt, wenn jede Menge in P eine leichte (d.h. in Polynomialzeit berechenbare) Zensusfunktion hat. Das Hauptergebnis in [GOR00] ist, dass jede Funktion aus $\#P_1^{\text{PH}}$ in $\text{FP}^{\#P_1^{\#P_1}}$ berechnet werden kann. Das bedeutet, dass genau dann eine jede Menge in P eine leichte Zensusfunktion hat, wenn jede Menge in der Polynomialzeit-Hierarchie eine solche hat.

Zensusfunktionen sind ein zentraler Begriff in der Komplexitätstheorie und haben sich in vielerlei Hinsicht als nützlich erwiesen, so etwa hinsichtlich der Isomorphievermutung von Berman und Hartmanis (siehe [BH77] und Abschnitt 3.6.2), der Arbeit zur Existenz von Turing-harten dünnen Mengen für verschiedene Komplexitätsklassen (siehe [KL80, KS85, BBS86a, HR97b] und Abschnitt 5.9), hinsichtlich der Resultate, die die Zeitschranken von NP-Mengen in Relation zu ihrer Dichte setzen, und der Resultate über die „P-printability“ (siehe Fußnote 12 und [HY84, AR88, GH96]), hinsichtlich der Upward-Separation-Technik (siehe [Har83b, HIS85, All91, RRW94, HJ95] und Abschnitt 3.3), der Resultate über positive Relativierung und über Relativierung mit dünnen Orakelmengen (siehe [Lon85, LS86, BBS86a]),

des von L. Hemaspaandra [Hem89] gezeigten Kollapses der starken Exponentialzeit-Hierarchie und schließlich hinsichtlich der oben erwähnten Arbeit über $\#P_1$, die tally NP-Mengen in Beziehung zu leichten Zensusfunktionen setzt [GOR00].

Grundlagen der Kryptologie

„But“, said I, returning him the slip, „I am as much in the dark as ever. Were all the jewels of Golconda awaiting me upon my solution of this enigma, I am quite sure that I should be unable to earn them“.

„And yet“, said Legrande, „the solution is by no means so difficult as you might be led to imagine from the first hasty inspection of the characters. These characters, as any one might readily guess, form a cipher—that is to say, they convey a meaning; but then from what is known of Kidd, I could not suppose him capable of constructing any of the more abstruse cryptographs. I made up my mind, at once, that this was of a simple species—such, however, as would appear, to the crude intellect of the sailor, absolutely insoluble without the key“.

(Aus „The Gold-Bug“ von Edgar Allan Poe, Random House, Inc., 1965)

4.1 Aufgaben und Ziele der Kryptologie

Kryptographie ist die Kunst und Wissenschaft davon, wie man Texte und Nachrichten so verschlüsselt, dass eine nicht autorisierte Entschlüsselung verhindert wird. Kryptoanalyse ist die Kunst und Wissenschaft vom Brechen vorhandener Kryptosysteme, d.h., ihr Ziel ist es, die verwendeten Entschlüsselungsschlüssel zu bestimmen und die verschlüsselten Nachrichten unerlaubt zu entschlüsseln. Kryptologie umfasst diese beiden Gebiete, die Kryptographie und die Kryptoanalyse.

Kryptographie

Ein typisches kryptographisches Szenario ist in Abbildung 1.1 von Kapitel 1 dargestellt. Alice und Bob kommunizieren miteinander über einen unsicheren Kanal, etwa eine öffentliche Telefonverbindung oder über das Internet. Erich belauscht ihre Unterhaltung. Um sich gegen Lauscher wie ihn zu schützen, verschlüsseln Alice und Bob ihre Nachrichten mit einem Kryptosystem.

Definition 4.1 (Kryptosystem).

- Ein Kryptosystem (oder, synonym dazu, eine Chiffre) ist ein Quintupel $\mathbf{S} = (M, C, K, \mathcal{E}, \mathcal{D})$ mit den folgenden Eigenschaften:

1. *M, C und K sind endliche Mengen, wobei M der Klartextraum ist (englisch auch als „plaintext space“, „cleartext space“ oder „message space“ bezeichnet), C ist der Schlüsseltextraum (englisch „ciphertext space“) und K ist der Schlüsselraum (englisch „key space“).*
2. *$\mathcal{E} = \{E_k | k \in K\}$ ist eine Familie von Funktionen $E_k : M \rightarrow C$, die für die Verschlüsselung verwendet werden, und $\mathcal{D} = \{D_k | k \in K\}$ ist eine Familie von Funktionen $D_k : C \rightarrow M$, die für die Entschlüsselung verwendet werden.*
3. *Für jeden Schlüssel $e \in K$ gibt es einen passenden Schlüssel $d \in K$, so dass für jede Nachricht $m \in M$ gilt:*

$$D_d(E_e(m)) = m. \quad (4.1)$$

- Ein Kryptosystem heißt symmetrisch (oder „private-key“), falls $d = e$ gilt oder falls d wenigstens „leicht“ aus e berechnet werden kann.
- Ein Kryptosystem heißt asymmetrisch (oder „public-key“), falls $d \neq e$ gilt und es „praktisch nicht machbar“ ist, d aus e zu berechnen. Hier ist d der private Schlüssel und e der öffentliche Schlüssel.

Man kann auch unterschiedliche Schlüsselräume zur Ver- und Entschlüsselung benutzen, was zu einer leichten Abänderung der obigen Definition führt.

Kryptosysteme ermöglichen üblicherweise einen Dialog oder eine Unterhaltung zwischen den beteiligten Parteien, wobei eine „Partei“ zum Beispiel eine einzelne Person, aber auch ein Computer sein kann. In einem solchen Dialog, den man ein kryptographisches Protokoll nennt, werden Nachrichten zwischen Alice und Bob hin und her geschickt, um bestimmte kryptographische Aufgaben zu erfüllen. Eine spezielle solche Aufgabe könnte etwa darin bestehen, dass sie sich auf einen gemeinsamen geheimen Schlüssel für ein symmetrisches Kryptosystem einigen. Protokolle können als Algorithmen aufgefasst werden, an deren Ausführung jeweils mehrere (authorisierte) Parteien beteiligt sind.

Kryptoanalyse

Die Kryptoanalyse zielt auf das Brechen von Kryptosystemen und das unerlaubte Dechiffrieren von Schlüsseltexten ab. Insbesondere versucht ein Kryptoanalytiker die in einem kryptographischen Protokoll verwendeten Schlüssel zu bestimmen. In Abhängigkeit von der dem Kryptoanalytiker verfügbaren Information kann man verschiedene Typen von Angriffen unterscheiden, welche bestimmte Sicherheitsstufen des betrachteten Kryptosystems charakterisieren. Natürlich ist das nur eine sehr grobe Klassifizierung. Da man für diese Angriffstypen üblicherweise die englischen Bezeichnungen verwendet, werden sie hier nicht ins Deutsche übertragen:

- **Ciphertext-only-Angriff:** Der Kryptoanalytiker kennt lediglich einige Schlüsseltexte, aus denen er die entsprechenden Klartexte oder Schlüssel zu bestimmen versucht. Dies ist die schwächste Form eines Angriffs. Ein Kryptosystem, das diesem Angriff nicht widersteht, ist nicht viel wert.

- **Known-Plaintext-Angriff:** Der Kryptoanalytiker kennt einige Paare von Schlüsseltexten und zugehörigen Klartexten, aus denen er die verwendeten Schlüssel zu ermitteln oder andere Schlüsseltexte zu entschlüsseln versucht.
- **Chosen-Plaintext-Angriff:** Der Kryptoanalytiker kann Klartexte nach Belieben wählen und erfährt die zugehörigen Schlüsseltexte, aus denen er die Schlüssel zu bestimmen versucht.
- **Chosen-Ciphertext-Angriff:** Der Kryptoanalytiker hat zeitweilig Zugang zum Entschlüsselungsgerät erhalten und kann einen Schlüsseltext wählen, um einen entsprechenden Klartext zu konstruieren.
- **Key-only-Angriff:** Diese Angriffsart ist für Public-Key-Kryptosysteme relevant. Der Kryptoanalytiker kennt nur den öffentlichen Schlüssel, aber er hat noch keine Schlüsseltexte abgefangen. Er versucht, den entsprechenden privaten Schlüssel zu bestimmen. Ein Unterschied zu den vorherigen Angriffsarten ist, dass der Angreifer nun so viel Zeit hat, wie er möchte, um seine Berechnungen auszuführen. Deshalb benötigen asymmetrische Kryptosysteme einen stärkeren Schutz, etwa durch sehr große Schlüssel, um die Aufgabe des Angreifers zu erschweren und hinreichend sicher zu sein. Aus diesem Grund sind asymmetrische Kryptosysteme oft weniger effizient als symmetrische Kryptosysteme.

Man könnte sich fragen, ob vielleicht auch geheim gehalten werden sollte, welches Kryptosystem verwendet wird. Sicherlich könnte es die Aufgabe des Kryptoanalytikers beträchtlich erschweren, wenn ihm das verwendete Kryptosystem verborgen bleibt. Jedoch wäre es dumm und äußerst gefährlich, sich auf seine Unfähigkeit zu verlassen, das verwendete Kryptosystem in Erfahrung zu bringen. Die Geschichte der Kryptologie kennt eine Fülle von Begebenheiten, bei denen jemand auf die Geheimhaltung des verwendeten Kryptosystems vertraute, es den Angreifern jedoch schließlich gelang, dieses auszuspionieren. Daher hat man sich das Kerckhoffssche Prinzip zu Eigen gemacht, welches erstmals von dem holländischen Philologen und Kryptologen Jean Guillaume Hubert Victor François Alexandre Auguste Kerckhoffs von Nieuwenhof (1835 bis 1903) in seinem Buch „La cryptographie militaire“ formuliert wurde.

Prinzip 4.2 (Kerckhoffssches Prinzip) *Die Sicherheit eines Kryptosystems darf nicht von der Geheimhaltung des verwendeten Systems abhängen. Stattdessen darf die Sicherheit eines Kryptosystems nur von der Geheimhaltung der verwendeten Schlüssel abhängen.*

Kryptosysteme und kryptographische Protokolle werden für eine Vielzahl von Zwecken eingesetzt, unter anderem dafür, vertrauliche Informationen und Daten geheim zu halten. Geheimhaltung ist also eine zentrale Aufgabe der Kryptographie, allerdings nicht die einzige.

Authentikation und digitale Signaturen

Eine andere wichtige Aufgabe der Kryptographie ist die Authentikation. Beispielsweise sollten Dokumente, wie etwa Verträge, fälschungssicher unterzeichnet werden; die Signatur authentifiziert somit das Dokument, sie bestätigt seine Echtheit.

Handschriftliche Signaturen sind normalerweise sehr schwer zu fälschen. Wird das Dokument jedoch elektronisch übertragen, so muss es durch eine *digitale Signatur* authentifiziert werden: Alice möchte ihre (verschlüsselten) Nachrichten an Bob also so signieren, dass

- (a) Bob verifizieren kann, dass sie tatsächlich die Senderin der Nachricht ist, und
- (b) auch dritte Parteien (die Bob vielleicht nicht trauen) sich von der Authentizität ihrer Unterschrift überzeugen können.

Weder Erich noch eine andere Partei sollten in der Lage sein, Alice' digitale Signatur zu fälschen. Die Eigenschaft (a) wird bereits durch symmetrische Authentikationscodes gewährleistet. Die spezifische Asymmetrie digitaler Signaturen wird durch die Eigenschaft (b) ausgedrückt. Es ist diese Eigenschaft (b), die digitale Signaturen so nützlich und notwendig für zum Beispiel sicheren E-Commerce macht, da Interessenkonflikte zwischen Alice und Bob dann nicht nur möglich, sondern sogar zu erwarten sind.

Ein *Authentikationscode* ermöglicht es, die Integrität einer Nachricht sicherzustellen. Wir haben es nun nicht mehr nur mit einem passiven, sondern mit einem *aktiven Angriff* zu tun: Zusätzlich zu seinem Lauschangriff auf die Konversation zwischen Alice und Bob könnte Erich nun versuchen, sich an den übertragenen Nachrichten zu schaffen zu machen (ein *Substitutionsangriff*), oder er könnte versuchen, eine eigene Nachricht in den Kanal einzuführen, in der Hoffnung, dass diese von Bob als echt, also als von Alice abgeschickt akzeptiert wird (ein *Personifikationsangriff*, englisch „*impersonation attack*“). Diese Art eines aktiven Angriffs wird auch als „*Man-in-the-Middle*“-Angriff oder als „*Intruder-in-the-Middle*“-Angriff bezeichnet. Es ist also festzuhalten, dass nicht nur elektronische Dokumente wie etwa E-Mails, sondern auch *Personen* oder, allgemeiner, *Parteien* Authentikation benötigen können.

Hinsichtlich der oben genannten Arten von aktiven Angriffen kann man die folgenden Authentikationsprobleme unterscheiden:

- **Nachrichtenintegrität:** Wie kann man sicherstellen, dass sich kein Eindringling an der empfangenen Nachricht zu schaffen gemacht und sie manipuliert hat?
- **Nachrichtenauthentikation:** Wie kann man sicherstellen, dass eine Nachricht tatsächlich vom behaupteten Sender stammt und nicht durch einen Eindringling eingeführt wurde?
- **Nutzerauthentikation:** Wie kann man sich von der Identität einer Person überzeugen?

In den folgenden Abschnitten und Kapiteln werden wir uns mit einer Vielzahl von kryptographischen Methoden und Protokollen befassen, die alle versuchen, diese Probleme zu lösen.

4.2 Einige klassische Kryptosysteme und ihre Kryptoanalyse

In diesem Abschnitt werden einige klassische symmetrische Kryptosysteme vorgestellt, und es wird eine grobe Klassifizierung solcher Kryptosysteme angegeben. Aus

der gewaltigen Vielfalt von Kryptosystemen, die bisher vorgeschlagen wurden, wird in diesem Kapitel nur eine sehr kleine Anzahl präsentiert werden. Moderne Public-Key-Kryptosysteme werden später, in den Kapiteln 7 und 8, eingeführt.

4.2.1 Substitutions- und Permutationschiffren

Sei Σ ein Alphabet. Nachrichten sind Elemente von Σ^* , wobei Σ^* die Menge der Wörter über Σ bezeichnet. In vielen Kryptosystemen werden Nachrichten $m \in \Sigma^*$ in Blöcke gleicher Länge, etwa n , unterteilt und dann blockweise verschlüsselt. Die einzelnen Blöcke von m sind Elemente von Σ^n , außer möglicherweise der letzte Block, der kürzer sein kann. Blockchiffren bilden von Σ^n in Σ^n ab. Es gibt eine Vielzahl von Methoden, mit denen man lange Nachrichten blockweise verschlüsseln kann, siehe Abschnitt 4.2.3.

Definition 4.3 (Blockchiffre und Substitutionschiffre).

- Eine Blockchiffre ist ein Kryptosystem, in dem der Klartextraum und der Schlüsselraum jeweils Σ^n ist, die Menge der Wörter der Länge n über einem Alphabet Σ . Die Zahl n heißt die Blocklänge (oder manchmal die Periode) des Systems.
- Eine Substitutionschiffre ist eine Blockchiffre mit Blocklänge eins.

Da einer jeden Verschlüsselungsfunktion eine Entschlüsselungsfunktion entspricht, sind die Verschlüsselungsfunktionen einer Blockchiffre injektiv. Eine injektive Funktion, die von Σ^n auf Σ^n abbildet, ist eine Bijektion (bzw. eine Permutation). Daraus ergibt sich die folgende Beobachtung.

Beobachtung 4.4 Die Verschlüsselungsfunktionen einer Blockchiffre sind Permutationen.

Gemäß Beobachtung 4.4 kann die allgemeinste Blockchiffre folgendermaßen beschrieben werden. Fixiere ein Alphabet Σ und eine Blocklänge n und definiere den Klartextraum und Schlüsselraum durch $M = C = \Sigma^n$. Der Schlüsselraum K sei als die Menge aller Permutationen von Σ^n gegeben. Für jeden Schlüssel $\pi \in K$ werden die Verschlüsselungsfunktion E_π und die Entschlüsselungsfunktion D_π , die beide von Σ^n in Σ^n abbilden, wie folgt definiert:

$$\begin{aligned} E_\pi(\mathbf{x}) &= \pi(\mathbf{x}); \\ D_\pi(\mathbf{y}) &= \pi^{-1}(\mathbf{y}), \end{aligned}$$

wobei π^{-1} die zu π inverse Permutation ist. Hat Σ genau m Buchstaben, so enthält der Schlüsselraum $(m^n)!$ Elemente.

Dieses Kryptosystem ist jedoch nicht praktikabel, da man die Permutation π (bzw. ihre Inverse, π^{-1}) zum Entschlüsseln einer Nachricht kennen muss. Stellt man $\pi \in K$ als eine Tabelle dar, die $\pi(\mathbf{x})$ für jedes $\mathbf{x} \in \Sigma^n$ enthält, so erhält man eine Tabelle der Größe m^n . Deshalb ist es sinnvoller, nur solche Permutationen zu verwenden, die sich aus dem Vertauschen der Positionen von Klartextbuchstaben ergeben. Dies ist

die *Permutationschiffre*, die auch die *Transpositionschiffre* genannt wird. Anders als die Substitutionschiffren ersetzt die Permutationschiffre nicht die Klartextbuchstaben durch andere Buchstaben des Schlüsseltextalphabets. Stattdessen werden Klartextbuchstaben lediglich auf andere Positionen im Schlüsseltext verschoben, bleiben ansonsten aber unverändert.

Beispiel 4.5 (Permutationschiffre). Sei Σ ein Alphabet, und sei $n \in \mathbb{N}$ die Blocklänge. Sei $M = C = \Sigma^n$, und der Schlüsselraum $K = \mathfrak{S}_n$ sei die Permutationsgruppe mit n Elementen. Für jeden Schlüssel $\pi \in \mathfrak{S}_n$ seien die Verschlüsselungsfunktion E_π und die Entschlüsselungsfunktion D_π , die beide von Σ^n in Σ^n abbilden, wie folgt definiert:

$$\begin{aligned} E_\pi(x_1x_2 \cdots x_n) &= x_{\pi(1)}x_{\pi(2)} \cdots x_{\pi(n)}; \\ D_\pi(y_1y_2 \cdots y_n) &= y_{\pi^{-1}(1)}y_{\pi^{-1}(2)} \cdots y_{\pi^{-1}(n)}. \end{aligned}$$

Hier hat der Schlüsselraum $n!$ Elemente, und jeder Schlüssel kann als eine Folge von n Zahlen codiert werden.

Nun werden einige konkrete Blockchiffren beschrieben. Unser Alphabet ist $\Sigma = \{A, B, \dots, Z\}$, welches in vielen Fällen sowohl als Klartext- und Schlüsselraum als auch als Schlüsselraum verwendet wird. Viele Kryptosysteme beruhen auf einfachen arithmetischen Operationen wie etwa der modularen Arithmetik; siehe Problem 2.1. Um diese Operationen mit Buchstaben auszuführen, als ob diese Zahlen wären, identifiziert man Σ mit dem Ring $\mathbb{Z}_{26} = \{0, 1, \dots, 25\}$, siehe Beispiel 2.35. Die Zahl 0 repräsentiert A, die Zahl 1 repräsentiert B und so weiter. Diese Codierung von Klartextbuchstaben durch natürliche Zahlen und die Decodierung von \mathbb{Z}_{26} zurück zu Σ ist nicht Teil der eigentlichen Ver- bzw. Entschlüsselung.

Eine der einfachsten Blockchiffren ist die Verschiebungschiffre, die die Blocklänge eins hat und somit eine Substitutionschiffre ist.

Beispiel 4.6 (Verschiebungschiffre). Die Verschiebungschiffre ist ein monoalphabetisches symmetrisches Kryptosystem. Sei $K = M = C = \mathbb{Z}_{26}$. Bei der *Verschiebungschiffre* werden Nachrichten dadurch verschlüsselt, dass jedes Klartextzeichen (modulo 26) um dieselbe Anzahl k von Buchstaben im Alphabet verschoben wird, wobei $k \in \mathbb{Z}_{26}$ der Schlüssel ist. Verschiebt man jedes Zeichen des Schlüsseltextes mit demselben Schlüssel k zurück, wird die ursprüngliche Nachricht enthüllt. Das heißt, für jeden Schlüssel $k \in \mathbb{Z}_{26}$ sind die Verschlüsselungsfunktion E_k und die Entschlüsselungsfunktion D_k , die beide von \mathbb{Z}_{26} in \mathbb{Z}_{26} abbilden, definiert durch:

$$\begin{aligned} E_k(x) &= (x + k) \bmod 26; \\ D_k(y) &= (y - k) \bmod 26. \end{aligned}$$

Wählen wir beispielsweise den Schlüssel $k = 17 = R$, so wird die Nachricht „BRUTUS FORCE EASILY BREAKS CAESAR“ wie in Table 4.1 gezeigt verschlüsselt.

Tabelle 4.1. Verschlüsselung durch die Verschiebungschiffre mit Schlüssel $k = 17$

m	B R U T U S	F O R C E	E A S I L Y	B R E A K S	C A E S A R
c	S I L K L J	W F I T V	V R J Z C P	S I V R B J	T R V J R I

Für den Spezialfall $k = 3$ heißt die Verschiebungschiffre auch *Cäsar-Chiffre*, da der römische Kaiser und Diktator Julius Cäsar diese Chiffre angeblich benutzt hat, um militärische Botschaften geheim zu halten, siehe Abschnitt 4.5. Verschiebungschiffren sind sehr einfache Substitutionschiffren, da jeder Klartextbuchstabe an derselben Position im Text verbleibt, aber durch einen bestimmten Buchstaben des Schlüsseltextalphabets ersetzt wird.

Da der Schlüsselraum ausgesprochen klein ist, können Verschiebungschiffren – und insbesondere die Cäsar-Chiffre – leicht mit „brutaler Gewalt“ gebrochen werden: Indem man einfach jeden der 26 möglichen Schlüssel überprüft, kann man leicht den entdecken, der einen sinnvollen Klartext ergibt, vorausgesetzt, dass der Schlüsseltextr lang genug ist, um eine eindeutige Entschlüsselung zu erlauben. Somit ist jede Verschiebungschiffre anfällig gegen Ciphertext-only-Angriffe, die schwächste Form eines Angriffs.

Ein weiteres Beispiel einer Substitutionschiffre ist die *affine Chiffre*, die mittels affiner Funktionen verschlüsselt, also mittels Abbildungen der Form

$$E(x) = ax + b \bmod m$$

für $a, b \in \mathbb{Z}_{26}$. Die Zahlen a und b stellen den Schlüssel dar. Für den Spezialfall $a = 1$ entartet die affine Chiffre zur Verschiebungschiffre.

Beispiel 4.7 (Affine Chiffre). Die affine Chiffre ist ein monoalphabetisches symmetrisches Kryptosystem mit Schlüsselraum $K = \{(a, b) | a, b \in \mathbb{Z}_{26}, \text{ggT}(a, 26) = 1\}$ und Klartext- bzw. Schlüsseltextram $M = C = \mathbb{Z}_{26}$. Die *affine Chiffre* verschlüsselt Nachrichten Buchstabe für Buchstabe. Für jeden Schlüssel $(a, b) \in \mathbb{Z}_{26} \times \mathbb{Z}_{26}$ mit $\text{ggT}(a, 26) = 1$ sind die Verschlüsselungsfunktion $E_{(a,b)}$ und die Entschlüsselungsfunktion $D_{(a^{-1},b)}$, die beide von \mathbb{Z}_{26} in \mathbb{Z}_{26} abbilden, definiert durch:

$$\begin{aligned} E_{(a,b)}(x) &= ax + b \bmod 26; \\ D_{(a^{-1},b)}(y) &= a^{-1}(y - b) \bmod 26, \end{aligned}$$

wobei a^{-1} das inverse Element von a in \mathbb{Z}_{26} ist, d.h., $aa^{-1} \equiv a^{-1}a \equiv 1 \bmod 26$. Mit dem erweiterten Euklidischen Algorithmus kann a^{-1} leicht bestimmt werden; siehe Abbildung 2.2 in Kapitel 2.

Wähle zum Beispiel den Schlüssel $k = (5, 7)$ zur Verschlüsselung. 21 ist das inverse Element von 5 modulo 26, denn $5 \cdot 21 = 105 = 1 + 4 \cdot 26 \equiv 1 \bmod 26$. Folglich ist $k' = (21, 7)$ der Entschlüsselungsschlüssel. Betrachte die Nachricht m und ihre Verschlüsselung c in Tabelle 4.2. Der erste Klartextbuchstabe ist ein „T“, das als 19 codiert wird. Der entsprechende erste Buchstabe des Schlüsseltextrates wird durch

$$E_{(5,7)}(19) = 5 \cdot 19 + 7 \equiv 24 \bmod 26$$

bestimmt. Somit verschlüsselt der Schlüsseltextbuchstabe „Y“, der der 24 entspricht, das „T“.

Um zu überprüfen, ob der Entschlüsselungsschlüssel $k' = (21, 7)$ diesen Buchstaben korrekt entschlüsselt, berechnet man:

$$D_{(21,7)} = 21(24 - 7) = 357 \equiv 19 \pmod{26}.$$

Gut, es klappt. Verschlüsselt allgemein ein Schlüsseltextbuchstabe y einen Klartextbuchstaben x mit dem Schlüssel (a, b) , so ergibt sich:

$$\begin{aligned} y \equiv ax + b \pmod{26} &\iff ax \equiv y - b \pmod{26} \\ &\iff a^{-1}ax \equiv a^{-1}(y - b) \pmod{26} \\ &\iff x \equiv a^{-1}(y - b) \pmod{26}, \end{aligned}$$

womit (4.1) erfüllt ist. Der restliche Schlüsseltext ist in Tabelle 4.2 angegeben.

Tabelle 4.2. Verschlüsselung durch die affine Chiffre mit Schlüssel $k = (5, 7)$

m	T H E E L E C T I V E A F F I N I T I E S B Y G O E T H E
c	Y Q B B K B R Y V I B H G G V U V Y V B T M X L Z B Y Q B

Kryptoanalyse der affinen Chiffre

Für das Alphabet \mathbb{Z}_{26} hat die affine Chiffre nur $26 \cdot \varphi(26) = 26 \cdot 12 = 312$ Schlüssel, da es 26 Wahrmöglichkeiten für $b \in \mathbb{Z}_{26}$ gibt und $\varphi(26)$ die Anzahl der zu 26 teilerfremden $a \in \mathbb{Z}_{26}$ ist, wobei φ die Euler-Funktion aus Definition 2.36 ist. Somit bricht ein Ciphertext-only-Angriff die affine Chiffre mit brutaler Gewalt, d.h., durch eine erschöpfende Suche im Schlüsselraum.

Das folgende Beispiel demonstriert einen Known-Plaintext-Angriff, bei dem zwei Klartextbuchstaben und ihre Verschlüsselungen bekannt sind. Der Angriff verwendet einfache lineare Algebra an, siehe Problem 2.1 in Kapitel 2.

Beispiel 4.8 (Known-Plaintext-Angriff auf die affine Chiffre). Angenommen, der Kryptoanalytiker kennt den Schlüsseltext c aus Tabelle 4.2 in Beispiel 4.7, und er kennt auch die ersten beiden Klartextbuchstaben, „T“ und „H“, die den ersten beiden Schlüsseltextbuchstaben, „Y“ und „Q“, entsprechen. Dann kann er die Schlüssel folgendermaßen bestimmen:

- Da „Y“ das „T“ und „Q“ das „H“ verschlüsselt, erhält man die Kongruenzen:

$$19a + b \equiv 24 \pmod{26}; \tag{4.2}$$

$$7a + b \equiv 16 \pmod{26}. \tag{4.3}$$

- (4.3) ist äquivalent zu $b \equiv 16 - 7a \pmod{26}$. Setzt man dies in (4.2) ein, erhält man $19a + 16 - 7a \equiv 24 \pmod{26}$ und somit $12a \equiv 8 \pmod{26}$, woraus folgt:

$$6a \equiv 4 \pmod{13}. \quad (4.4)$$

- Multipliziert man (4.4) mit dem inversen Element 11 von 6 modulo 13, so ergibt sich:

$$a \equiv 44 \equiv 5 \pmod{13}.$$

- Es folgt $a = 5$ und $b = 7$.

Sowohl die Verschiebungschiffre als auch die affine Chiffre sind monoalphabetisch, da jeder Buchstabe im Klartext stets durch denselben Buchstaben im Schlüsseltext ersetzt wird. Die Methode der *Häufigkeitsanalyse* ist oft nützlich, um monoalphabetische Kryptosysteme zu brechen. Sie nutzt die Redundanz der natürlichen Sprache aus, in der der Klartext verfasst ist. In vielen Sprachen kommt der Buchstabe „E“ statistisch signifikant am häufigsten vor. So erscheint z.B. das „E“ mit einem Prozentsatz von 12.31% in englischen, von 15.87% in französischen, von 13.15% in spanischen, von 11.79% in italienischen und sogar von 18.46% in deutschen Texten, vorausgesetzt, sie sind lang und „typisch“ genug. Wenn also ein typischer deutscher Text hinreichender Länge durch die Verschiebungschiffre verschlüsselt wird und z.B. der Buchstabe „Y“ (der an sich im Deutschen ziemlich selten ist) im Schlüsseltext mit der höchsten Häufigkeit vorkommt, dann verschlüsselt das „Y“ höchstwahrscheinlich das „E“, und „U“ ($k = 20$) ist demnach der verwendete Schlüssel.

Tabelle 4.3. Häufigkeiten von Buchstaben in langen, typischen englischen Texten

Mit großer Häufigkeit auftretende Buchstaben									Insgesamt
Buchstabe	E	T	A	O	N	I	S	R	H
Häufigkeit in %	12.31	9.59	8.05	7.94	7.19	7.18	6.59	6.03	5.14
70.02%									
Mit mittlerer Häufigkeit auftretende Buchstaben									
Buchstabe	L	D	C	U	P	F	M	W	Y
Häufigkeit in %	4.03	3.65	3.20	3.10	2.29	2.28	2.25	2.03	1.88
24.71%									
Mit geringer Häufigkeit auftretende Buchstaben									
Buchstabe	B	G	V	K	Q	X	J	Z	
Häufigkeit in %	1.62	1.61	0.93	0.52	0.20	0.20	0.10	0.09	
5.27%									

In anderen Sprachen können andere Buchstaben mit der höchsten Häufigkeit vorkommen; beispielsweise ist „A“ der häufigste Buchstabe in durchschnittlichen finnischen Texten, mit einem Prozentsatz von 12.06%, siehe Salomaas Buch [Sal96]. Tabelle 4.3 zeigt die Häufigkeiten der Buchstaben in typischen englischen Texten hinreichender Länge; die Werte stammen aus [Gai39]. Es muss allerdings betont werden, dass die Buchstabenhäufigkeiten, die in verschiedenen Büchern zusammengetragen wurden, von Quelle zu Quelle variieren. Diese Tatsache ist nicht überraschend; sie hebt lediglich die Schwierigkeit hervor, zu definieren, was ein „typischer“

Text in einer natürlichen Sprache ist. Offenbar macht es einen Unterschied, welcher Texttyp vorliegt, sei es Poesie, Prosa, ein Zeitungsartikel, ein technischer Text, ein wissenschaftlicher Text, der Straßenslang im Kiez, ein Dialekt und so weiter. Dennoch haben die Verteilungen von Buchstaben in allen Häufigkeitstabellen einige gemeinsame Eigenschaften. Beispielsweise steht der Buchstabe „E“ in englischen (und ebenso in deutschen) Häufigkeitstabellen immer an der Spitze und ihm folgt in englischen Tabellen stets das „T“; die Reihenfolge einiger der anderen Buchstaben kann sich hingegen von Tabelle zu Tabelle unterscheiden.

Beispiel 4.9 (Kryptoanalyse der affinen Chiffre durch Häufigkeitsanalyse). Ange- nommen, der Kryptoanalytiker Erich fängt den Schlüsseltext

$$c = Y Q B B K B R Y V I B H G G V U V Y V B T M X L Z B Y Q B$$

aus Beispiel 4.7 ab, und er vermutet, dass Alice ihre Nachricht mit der affinen Chiffre verschlüsselt hat, welche monoalphabetisch ist. Er ist schlau genug, c zu analysieren, indem er die Häufigkeiten zählt, mit denen die einzelnen Buchstaben auftreten, und er erhält Tabelle 4.4.

Tabelle 4.4. Häufigkeiten der Buchstaben im Schlüsseltext aus Beispiel 4.7

Buchstabe	B	Y	V	Q	G	K	R	I	H	U	T	M	X	L	Z
Häufigkeit	7	4	4	2	2	1	1	1	1	1	1	1	1	1	1

Da „B“ mit der höchsten Häufigkeit vorkommt, gefolgt von „Y“ und „V“, rät Erich, dass „B“ das „E“ verschlüsselt und dass „Y“ und „V“ jeweils einen der Buchstaben „T“, „A“, „O“, „N“ oder „I“ verschlüsseln. Nach der Überprüfung dieser Möglichkeiten, schließt er, dass „Y“ höchstwahrscheinlich „T“ verschlüsselt, da die ersten drei Buchstaben „T?E“ sehr wahrscheinlich dem gebräuchlichen englischen Wort „THE“ entsprechen, woraus er noch einen weiteren Buchstaben erhält: „Q“ verschlüsselt „H“. Außerdem verschlüsselt „V“ sehr wahrscheinlich das „I“, denn „...ITIES“ ist eine typische Endung englischer Substantive in der Mehrzahl; siehe Tabelle 4.5. Indem er in dieser Versuch-und-Fehler-Weise fortfährt und statistische Informationen über die Häufigkeiten der Buchstaben in typischen englischen Texten aus Tabelle 4.3 verwendet, entschlüsselt Erich schließlich die gesamte Nachricht und ermittelt den verwendeten Schlüssel.

Es muss jedoch betont werden, dass der Text in der obigen Beispieldnung kurz ist, was bedeutet, dass Erich beim Raten ein ziemlich glückliches Händchen hatte, wodurch es ihm gelang, den Text mittels der Häufigkeitsanalyse zu entschlüsseln. Andererseits können kurze Nachrichten immer mit brutaler Gewalt entschlüsselt werden; wenn sie nur kurz genug sind, braucht man nicht einmal einen Computer dafür. Die Methode der Häufigkeitsanalyse funktioniert um so besser, je länger die Nachricht ist. Man hat jedoch keine Garantie dafür, dass sie bei jeder Nachricht zum Erfolg führt; sie stellt lediglich einige statistische Indizien zur Verfügung, die dem Kryptoanalytiker helfen könnten.

Tabelle 4.5. Raten in der Häufigkeitsanalyse: „B“ ist „E“, „Y“ ist „T“ und „Q“ ist „H“

<i>c</i>	Y Q B B K B R Y V I B H G G V U V Y V B T M X L Z B Y Q B
V is A	T H E E ? E ? T A ? E ? ? A ? A T A E ? ? ? ? ? E T H E
V is O	T H E E ? E ? T O ? E ? ? O ? O T O E ? ? ? ? ? E T H E
V is N	T H E E ? E ? T N ? E ? ? N ? N T N E ? ? ? ? ? E T H E
V is I	T H E E ? E ? T I ? E ? ? I ? I T I E ? ? ? ? ? E T H E
<i>m</i>	T H E E L E C T I V E A F F I N I T I E S B Y G O E T H E

Zusätzlich zum Zählen der Vorkommen einzelner Buchstaben in einem Text kann man auch die Häufigkeiten von Buchstabenpaaren (Digrammen), von Buchstabentripeln (Trigrammen) und so weiter bestimmen. Digramme und Trigramme treten ebenfalls gemäß gewisser Wahrscheinlichkeitsverteilungen in langen, typischen Texten einer gegebenen natürlichen Sprache auf, und ihr Vorkommen in einem durch ein monoalphabetisches Kryptosystem erzeugten Schlüsseltext kann dem Kryptoanalytiker zusätzliche nützliche Hinweise geben.

4.2.2 Affin-lineare Blockchiffren

Im Gegensatz zu monoalphabetischen Chiffren können *polyalphabetische Chiffren* Klartextbuchstaben in Abhängigkeit von ihrer Position im Text durch verschiedene Schlüsseltextbuchstaben ersetzen. Ein berühmtes solches polyalphabetisches System wurde von dem französischen Diplomaten und Kryptoanalytiker Blaise de Vigenère (1523 bis 1596) erfunden. Seine Chiffre funktioniert wie die Verschiebungschiffre, nur dass der Schlüsseltextbuchstabe, der einen gegebenen Klartextbuchstaben X verschlüsselt, mit der Position von X im Klartext variiert.

Beispiel 4.10 (Vigenère-Chiffre). Dieses polyalphabetische symmetrische Kryptosystem benutzt ein so genanntes *Vigenère-Quadrat*, eine Matrix mit 26 Zeilen und ebenso vielen Spalten, siehe Tabelle 4.6. Jede Zeile enthält die 26 Buchstaben des Alphabets, von Zeile zu Zeile um jeweils eine Position nach links verschoben (gemäß der Arithmetik modulo 26). Anders gesagt können die einzelnen Zeilen (und ebenso die einzelnen Spalten) als die Verschiebungschiffre mit den Schlüsseln $0, 1, \dots, 25$ aufgefasst werden. Welche Zeile des Vigenère-Quadrats für die Verschlüsselung eines Klartextbuchstabens verwendet wird, hängt von dessen Position im Text ab.

Nachrichten werden in Blöcke der Länge n unterteilt und dann blockweise verschlüsselt. Das heißt, $K = M = C = \mathbb{Z}_{26}^n$, wobei n die Blocklänge des Systems ist. Für jeden Schlüssel $\mathbf{k} \in \mathbb{Z}_{26}^n$ sind die Verschlüsselungsfunktion $E_{\mathbf{k}}$ und die Entschlüsselungsfunktion $D_{\mathbf{k}}$, die beide von \mathbb{Z}_{26}^n in \mathbb{Z}_{26}^n abbilden, definiert durch:

$$\begin{aligned} E_{\mathbf{k}}(\mathbf{x}) &= (\mathbf{x} + \mathbf{k}) \bmod 26; \\ D_{\mathbf{k}}(\mathbf{y}) &= (\mathbf{y} - \mathbf{k}) \bmod 26, \end{aligned}$$

wobei die Addition und Subtraktion mit \mathbf{k} modulo 26 zeichenweise ausgeführt werden. Noch etwas konkreter schreibt man den Schlüssel $\mathbf{k} \in \mathbb{Z}_{26}^n$ Symbol für Symbol

Tabelle 4.6. Vigenère-Quadrat: Klartext „H“ wird mit Schlüssel „E“ als „L“ verschlüsselt

0	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
2	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	
3	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	
4	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	
5	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	
6	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	
7	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	
8	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	
9	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	
10	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	
11	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	
12	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	
13	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	
14	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	
15	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	
16	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	
17	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	
18	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	
19	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	
20	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	
21	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	
22	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	
23	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	
24	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	
25	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	

über jeden Block $\mathbf{x} \in \mathbb{Z}_{26}^n$ des Klartextes. Hat der letzte Block weniger als n Symbole, so werden entsprechend weniger Symbole des Schlüssels verwendet. Sei s_i das i -te Symbol eines gegebenen Wortes $\mathbf{s} = s_1 s_2 \cdots s_n$. Um das i -te Klartextsymbol x_i von \mathbf{x} zu verschlüsseln, über welchem das i -te Schlüsselsymbol k_i von \mathbf{k} steht, verwendet man die i -te Zeile des Vigenère-Quadrats wie die Verschiebungschiffre mit Schlüssel k_i . Man sieht leicht, dass ein und dasselbe Klartextsymbol somit durch unterschiedliche Schlüsseltextsymbole verschlüsselt werden kann.

Tabelle 4.7. Verschlüsselung durch die Vigenère-Chiffre mit Schlüssel ELLA

Schlüssel	ELLAELLAELL AEL LAELL AE LLAELLA
Nachricht	HUNGARIAN IS ALL GREEK TO GERMANS
Schlüsseltext	LFYGECTARTDAPWRRIPVTSRPRQLYS

Wähle zum Beispiel die Periode $n = 4$ und den Schlüssel $\mathbf{k} = \text{ELLA}$. Tabelle 4.7 zeigt die Verschlüsselung eines Klartextes, der aus sieben Blöcken besteht, in einen

Schlüsseltext unter Verwendung der Vigenère-Chiffre mit diesem Schlüssel. Über dem ersten Buchstaben des Klartextes, „H“, steht das Schlüsselsymbol „E“. Die „H“-Spalte schneidet sich mit der „E“-Zeile des Vigenère-Quadrats im Buchstaben „L“, welcher somit das erste Symbol des Schlüsseltextes ist, siehe Tabelle 4.6.

Offenbar kann, wie oben erwähnt, dasselbe Klartextsymbol tatsächlich durch verschiedene Schlüsseltextsymbole verschlüsselt werden. Beispielsweise

- erscheint der Klartextbuchstabe „A“ viermal und wird zweimal durch „A“ verschlüsselt, einmal durch „E“ und einmal durch „L“;
- erscheint der Klartextbuchstabe „E“ dreimal und wird einmal durch „I“ verschlüsselt und zweimal durch „P“;
- erscheint der Klartextbuchstabe „G“ dreimal und wird einmal durch „G“ verschlüsselt und zweimal durch „R“;
- erscheint der Klartextbuchstabe „N“ dreimal und wird einmal durch „R“ verschlüsselt und zweimal durch „Y“;
- erscheint der Klartextbuchstabe „R“ dreimal und wird einmal durch „C“ verschlüsselt und zweimal durch „R“.

Diese Beobachtung zeigt auch zwei Schwächen des gewählten Schlüssels:

1. Zwei Buchstaben des Schlüssels ELLA sind gleich und
2. ein Buchstabe des Schlüssels ist ein „A“, welches den entsprechenden Klartextbuchstaben nicht verändert.

Die Vigenère-Chiffre ist ein Spezialfall der *affin-linearen Blockchiffre*, welche die affine Chiffre verallgemeinert. Bevor die affin-lineare Blockchiffre definiert wird, ist es zweckmäßig, sich einige grundlegende Begriffe der linearen Algebra ins Gedächtnis zu rufen; siehe auch Definition 2.34 in Abschnitt 2.4. Insbesondere benötigen affin-lineare Blockchiffren Operationen auf Matrizen über dem Ring \mathbb{Z}_m , d.h., die Matrixeinträge sind Elemente von \mathbb{Z}_m und die Matrixoperationen beruhen auf der Arithmetik modulo m , siehe Problem 2.1 in Abschnitt 2.6.

Definition 4.11 (Inverse Matrix, Determinante und Adjunkte).

- Sei $\mathbf{u}_i = (0, \dots, 0, 1, 0, \dots, 0)$ der i -te Einheitsvektor der Länge n , d.h., die i -te Koordinate von \mathbf{u}_i ist eins, und die j -te Koordinate von \mathbf{u}_i ist null für alle $j \neq i$.
- Die Einheitsmatrix der Dimension $n \times n$ ist definiert durch $U_n = (\mathbf{u}_i)_{1 \leq i \leq n}$, wobei die i -te Zeile (und Spalte) von U_n der i -te Einheitsvektor der Länge n ist.
- Betrachte eine $(n \times n)$ -Matrix A über dem Ring \mathbb{Z}_m . Die (multiplikative) Inverse von A , bezeichnet mit A^{-1} , ist eine $(n \times n)$ -Matrix, für die gilt, dass $AA^{-1} = A^{-1}A$ die Einheitsmatrix U_n der Dimension $n \times n$ ist.
- Die Determinante von A kann rekursiv definiert werden:
 - für $n = 1$ und $A = (a)$ sei $\det A = a$;
 - für $n > 1$ und für jedes $i \in \{1, 2, \dots, n\}$ sei

$$\det A = \sum_{j=1}^n (-1)^{i+j} a_{i,j} \det A_{i,j},$$

wobei $a_{i,j}$ der (i,j) -Eintrag von A ist und sich die $((n-1) \times (n-1))$ -Matrix $A_{i,j}$ aus A durch Streichen der i -ten Zeile und der j -ten Spalte ergibt.

- Definiere die Adjunkte von A (auch die adjungierte Matrix von A genannt) durch $A_{\text{adj}} = ((-1)^{i+j} \det A_{j,i})$.

Eine $(n \times n)$ -Matrix A über dem Ring \mathbb{Z}_m hat genau dann eine (multiplikativ) inverse Matrix, wenn $\text{ggT}(\det A, m) = 1$ gilt, wobei $\det A$ die Determinante von A ist. Im Allgemeinen ist eine $(n \times n)$ -Matrix über den reellen Zahlen genau dann invertierbar, wenn ihre Determinante nicht verschwindet.

Die Determinante einer Matrix über \mathbb{Z} oder über \mathbb{Z}_m kann effizient berechnet werden, siehe Problem 4.1. Man kann zeigen, dass gilt:

$$A^{-1} = (\det A)^{-1} A_{\text{adj}}.$$

Nun wird die affin-lineare Blockchiffre definiert.

Definition 4.12 (Affin-lineare Blockchiffre). Eine Blockchiffre mit Klartext- und Schlüsselraum \mathbb{Z}_m^n und mit der Blocklänge n heißt affin-linear, falls alle ihre Verschlüsselungsfunktionen affin-linear sind, wenn sie also alle die folgende Form haben:

$$E_{(A, \mathbf{b})}(\mathbf{x}) = A\mathbf{x} + \mathbf{b} \pmod{m}, \quad (4.5)$$

wobei A eine $(n \times n)$ -Matrix über \mathbb{Z}_m ist, so dass $\text{ggT}(\det A, m) = 1$ gilt, und wobei \mathbf{x} , \mathbf{y} und \mathbf{b} Vektoren in \mathbb{Z}_m^n sind; dabei wird in der Arithmetik modulo m gearbeitet. Ist A^{-1} die zu A inverse Matrix, so ist die zugehörige Entschlüsselungsfunktion:

$$D_{(A^{-1}, \mathbf{b})}(\mathbf{y}) = A^{-1}(\mathbf{y} - \mathbf{b}) \pmod{m}.$$

Eine lineare Blockchiffre ist eine affin-lineare Blockchiffre, für die \mathbf{b} in (4.5) der Nullvektor ist.

Wie oben erwähnt ist die Vigenère-Chiffre affin-linear. Ein klassisches Beispiel einer linearen Chiffre ist die *Hill-Chiffre*, die 1929 von Lester Hill erfunden wurde. Tatsächlich ist die Hill-Chiffre die allgemeinste lineare Blockchiffre.

Beispiel 4.13 (Hill-Chiffre). Sei Σ ein Alphabet mit m Buchstaben, und sei n die Blocklänge. Der Klartext- und Schlüsselraum ist $M = C = \mathbb{Z}_m^n$. Der Schlüsselraum K ist die Menge aller $(n \times n)$ -Matrizen A mit Einträgen aus \mathbb{Z}_m , so dass $\text{ggT}(\det A, m) = 1$ gilt. Diese Bedingung stellt sicher, dass die Matrizen invertierbar sind, denn die inverse Matrix A^{-1} wird als der Entschlüsselungsschlüssel benutzt, der zum Verschlüsselungsschlüssel A gehört. Die Verschlüsselungsfunktion E_A und die Entschlüsselungsfunktion $D_{A^{-1}}$ sind definiert durch:

$$\begin{aligned} E_A(\mathbf{x}) &= A\mathbf{x} \pmod{m}; \\ D_{A^{-1}}(\mathbf{y}) &= A^{-1}\mathbf{y} \pmod{m}. \end{aligned}$$

Die Hill-Chiffre funktioniert am besten, wenn die Größe m des Alphabets eine Primzahl ist. Um das zu erreichen, fügt man zum Beispiel das Leerzeichen \square (codiert

als 26), das Komma (codiert als 27) und den Punkt (codiert als 28) zu den 26 Buchstaben des englischen Alphabets hinzu, welche durch die Zahlen 0, 1, ..., 25 codiert sind. Dann ist $m = 29$ eine Primzahl, und man arbeitet in der Arithmetik über \mathbb{Z}_{29} .

Wähle die Blocklänge $n = 2$ und eine invertierbare (2×2) -Matrix A und berechne die inverse Matrix A^{-1} in der Arithmetik modulo 29. Beispielsweise seien

$$A = \begin{pmatrix} 3 & 4 \\ 7 & 2 \end{pmatrix} \quad \text{und} \quad A^{-1} = \begin{pmatrix} 21 & 16 \\ 28 & 17 \end{pmatrix}$$

gewählt. Angenommen, es soll die Nachricht „THE FOOL ON THE HILL“ verschlüsselt werden. Tabelle 4.8 zeigt die Verschlüsselung dieses Klartexts mit dem Schlüssel A .

Tabelle 4.8. Beispiel einer Verschlüsselung mit der Hill-Chiffre

Klartext	T	H	E	□	F	O	O	L	□	O	N	□	T	H	E	□	H	I	L	L
codierter Klartext	19	7	4	26	5	14	14	11	26	14	13	26	19	7	4	26	7	8	11	11
codierter Schlüsseltext	27	2	0	22	13	5	28	4	18	7	27	27	27	2	0	22	24	7	19	12
Schlüsseltext	,	C	A	W	N	F	.	E	S	H	,	,	,	C	A	W	Y	H	T	M

Nun wird gezeigt, dass die in Beispiel 4.5 eingeführte Permutationschiffre linear ist. Somit ist sie ein Spezialfall der Hill-Chiffre.

Satz 4.14. *Die Permutationschiffre ist linear.*

Beweis. Sei $\pi \in \mathfrak{S}_n$ eine Permutation. Sei $U_n = (\mathbf{u}_i)_{1 \leq i \leq n}$ die $(n \times n)$ -Einheitsmatrix, deren i -te Zeile \mathbf{u}_i ist, der i -te Einheitsvektor der Länge n . Sei M_π die Matrix, deren i -te Zeile $\mathbf{u}_{\pi(i)}$ ist. Diese Matrix ergibt sich aus U_n durch Permutation der Zeilen gemäß π . Folglich ist $\mathbf{u}_{\pi(j)}$ die j -te Spalte von M_π , und es folgt

$$(x_{\pi(1)}, x_{\pi(2)}, \dots, x_{\pi(n)}) = M_\pi \mathbf{x}$$

für jeden Vektor $\mathbf{x} = (x_1, x_2, \dots, x_n)$ in Σ^n . □

Kryptoanalyse der Vigenère-Chiffre

In Abhängigkeit von der gewählten Blocklänge n kann der Schlüsselraum der Vigenère-Chiffre ziemlich groß sein: Er hat m^n Elemente, wobei m die Anzahl der Buchstaben im Alphabet ist. Die Methode der Häufigkeitsanalyse, die oft zum Brechen monoalphabetischer Systeme eingesetzt wird, ist nicht auf polyalphabetische Systeme anwendbar, deren Periode (d.h. Blocklänge) nicht bekannt ist. Daher schien die Vigenère-Chiffre, ebenso wie ähnliche periodische Kryptosysteme mit unbekannter Periode, der Kryptoanalyse durch Zählen und statistische Auswertung der

Häufigkeiten von Buchstaben, Digrammen und Trigrammen im Schlüsseltext zu widerstehen. Erst 1863 fand der deutsche Kryptoanalytiker Friedrich Wilhelm Kasiski eine Methode zum Brechen der Vigenère-Chiffre. Seine Leistung markiert einen Durchbruch in der Geschichte der Kryptoanalyse, da die Vigenère-Chiffre zuvor für nicht zu brechen gehalten wurde. Insbesondere zeigt Kasiskis Methode, wie die Periode aus den Wiederholungen derselben Teilwörter im Schlüsseltext bestimmt werden kann.

Kasiskis Methode wurde unabhängig von Charles Babbage erfunden, einem britischen Genie und bekannten Ekzentriker, der außerdem auch einen frühen Prototyp des Computers erfand; siehe Abschnitt 4.5. Bevor die Methode im Detail erklärt wird, sehen wir uns an, wie man sogar polyalphabetische Kryptosysteme brechen kann, sofern die Periode bekannt ist. In diesem Fall kann das Problem, das gegebene polyalphabetische Kryptosystem zu brechen, nämlich auf das Problem reduziert werden, ein monoalphabetisches Kryptosystem mittels der Häufigkeitsanalyse zu brechen.

Angenommen, die Periode ist $n = 7$. Den Schlüsseltext $C_0C_1C_2 \dots C_k$, wobei jeder C_j ein Buchstabe ist, ordnet man nun in sieben Spalten an, so dass die i -te Spalte aus den Buchstaben $C_i, C_{i+7}, C_{i+2 \cdot 7}, \dots, C_{i+j \cdot 7}$ besteht, wobei $i \in \mathbb{Z}_7$ und $i + j \cdot 7 \leq k$; siehe Tabelle 4.9. Da alle Buchstaben in der i -ten Spalte wie bei einem monoalphabetischen System durch dasselbe Schlüsselsymbol verschlüsselt werden, kann es mit einer normalen Häufigkeitsanalyse gelingen, den Schlüsseltext Spalte für Spalte zu entschlüsseln und die einzelnen Schlüsselsymbole zu bestimmen. Natürlich hat diese Methode um so wahrscheinlicher Erfolg, je länger der Schlüsseltext ist.

Tabelle 4.9. Kryptoanalyse eines polyalphabetischen Systems mit Periode 7

C_0	C_1	C_2	C_3	C_4	C_5	C_6
C_7	C_8	C_9	C_{10}	C_{11}	C_{12}	C_{13}
C_{14}	C_{15}	C_{16}	C_{17}	C_{18}	C_{19}	C_{20}
\vdots						
C_{k-8}	C_{k-7}	C_{k-6}	C_{k-5}	C_{k-4}	C_{k-3}	C_{k-2}
C_{k-1}	C_k					

Das folgende Beispiel erläutert Kasiskis Methode.

Beispiel 4.15 (Kasiskis Methode). Angenommen, der in Tabelle 4.10 dargestellte Schlüsseltext wurde abgefangen und man weiß, dass er durch eine Vigenère-Chiffre entstanden ist. Der Schlüsseltext hat 373 Buchstaben, und die verwendete Periode (also die Länge des Schlüssels) ist nicht bekannt. Analysiert man den Schlüsseltext sorgfältig, so wird man bestimmte Folgen von Buchstaben entdecken, die mehrfach im Text auftreten. Einige dieser sich wiederholenden Muster aus je drei Buchstaben sind in Tabelle 4.10 durch verschiedene Graustufen hervorgehoben.

Tritt ein solches Muster wiederholt auf, so kann das entweder der Tatsache zu verdanken sein, dass dasselbe Wort oder Teilwort im Klartext mit denselben Buchstaben des Schlüssels verschlüsselt wurde, oder es kann reiner Zufall sein. Nehmen

Tabelle 4.10. Kasiskis Methode: ein durch die Vigenère-Chiffre erzeugter Schlüsseltext

L	E	B	L	D	V	R	Y	L	T	U	U	H	T	N	H	P	U	T	N
I	H	U	E	Y	T	A	L	L	N	S	W	Y	E	R	P	V	Y	W	L
T	D	U	Y	D	L	R	I	E	E	P	N	X	S	E	B	I	H	R	W
P	Y	N	Z	O	Z	M	Y	E	U	C	A	Z	T	S	W	I	H	R	A
C	D	C	N	A	J	G	B	E	F	D	U	L	N	A	C	S	U	Y	D
L	E	F	L	U	V	H	Y	O	A	C	D	U	W	I	R	E	N	Z	K
A	A	M	L	S	Z	E	X	X	E	X	F	C	H	A	K	I	H	W	O
K	E	Q	T	T	W	G	Y	C	T	G	U	X	P	S	I	E	C	Y	B
T	C	U	F	S	T	I	BL	D	S	E	X	T	C	P	T	Y	O	A	
Q	O	I	V	O	U	P	I	P	M	H	T	I	S	E	G	E	P	P	N
I	H	I	F	G	W	T	B	P	Y	L	E	L	P	T	H	E	F	T	O
I	S	U	Y	D	X	S	U	T	D	N	E	M	T	L	D	V	Y	O	H
T	R	V	F	T	X	T	W	Z	U	A	D	H	P	V	T	R	Q	Z	R
Z	B	Y	N	A	J	S	Y	D	H	T	W	U	D	F	P	R	N	Z	O
X	N	X	P	L	A	I	A	P	N	I	F	I	C	M	T	A	H	O	A
A	I	W	P	T	D	K	F	L	S	P	G	L	P	E	S	A	H	O	T
W	E	H	H	E	E	U	N	Z	N	H	O	G	P	B	D	X	C	Y	G
V	L	I	G	E	H	A	H	O	G	T	R	N	C	U	S	E	M	E	E
X	N	V	C	O	Z	E	G	J	N	D	S	Y							

wir mal an, es war kein Zufall. Dann wird uns der Abstand zwischen wiederholt auftretenden Mustern etwas über die verwendete Schlüssellänge sagen. Mit Abstand ist die Anzahl der Positionen gemeint, um die eines dieser Muster verschoben werden muss, um mit einem anderen, identischen Muster übereinzustimmen. Beispielsweise

- tritt das Muster „AHO“ dreimal mit den Abständen 20 und 30 auf;
- tritt das Muster „UYD“ dreimal mit den Abständen 55 und 125 auf;
- tritt das Muster „ACD“ zweimal mit dem Abstand 30 auf;
- tritt das Muster „IHR“ zweimal mit dem Abstand 20 auf;
- tritt das Muster „BLD“ zweimal mit dem Abstand 165 auf.

Wenn das wiederholte Auftreten der Muster kein Zufall ist, dann muss die Schlüssellänge (also die Periode des Systems) alle Abstände teilen. Beispielsweise bedeutet ein Abstand von 20, dass die Periode entweder 2 oder 4 oder 5 oder 10 oder 20 ist. Da auch 30 ein Abstand zwischen Mustern ist, kann man die potenziellen Perioden 4 und 20 eliminieren. Unter den verbleibenden möglichen Perioden, 2 und 5 und 10, teilt nur die Periode 5 die Abstände 55, 125 und 165. Somit haben wir die Schlüssellänge 5 ermittelt.

Nun können wir versuchen, den Schlüssel zu finden und die Nachricht zu entschlüsseln. Dies kann mit der oben beschriebenen Methode erfolgen: Kennen wir die Periode, so können wir diese Aufgabe auf die Aufgabe zurückführen, ein monoalphabetisches System mittels Häufigkeitsanalyse zu brechen. Ordnen wir den Schlüsseltext also in fünf Spalten neu an, so erhalten wir fünf monoalphabetische

Verschlüsselungen. Insbesondere hat die zweite Spalte 75 Buchstaben, siehe Tabelle 4.11.

Tabelle 4.11. Kasiskis Methode: zweite Spalte des neu angeordneten Schlüsseltexts

E	R	U	P	H	A	W	V	D	R	N	I	Y	M	A	I	D	G	U	S	E	H	D	E	A
E	F	I	E	G	U	E	C	I	E	T	O	P	T	E	H	T	E	E	S	S	E	V	R	T
D	R	B	S	W	R	N	I	F	A	I	K	G	A	E	U	O	X	L	A	R	E	N	S	

Wie man sieht, kommt der Buchstabe „E“ am häufigsten vor: 14-mal, was einem prozentualen Anteil von 10.5% entspricht. Aber das bedeutet, dass die Buchstaben in der zweiten Spalte überhaupt nicht verschlüsselt worden sind! Die Analyse der fünften Spalte liefert dasselbe Ergebnis. Somit ist der zweite und der fünfte Buchstabe des Schlüssels ein „A“. Anders gesagt, wer auch immer diese Nachricht verschlüsselt hat, er hat seine Lektion aus Beispiel 4.10 nicht gelernt, da er denselben Buchstaben zweimal im Schlüssel verwendet und da er ein „A“ verwendet hat.

Setzt man die Entschlüsselung in dieser Weise fort, so erhält man schließlich den verwendeten Schlüssel: „PAULA“. Tabelle 4.12 zeigt die vollständig entschlüsselte Nachricht, eine Erinnerung an die Zwanziger von Woody Allen, die sich mit Satzzeichen folgendermaßen liest:

We had great fun in Spain that year and we travelled and wrote and Hemingway took me tuna fishing and I caught four cans and we laughed and Alice Toklas asked me if I was in love with Gertrude Stein because I had dedicated a book of poems to her even though they were T.S. Eliot's and I said, yes, I loved her, but it could never work because she was far too intelligent for me and Alice Toklas agreed and then we put on some boxing gloves and Gertrude Stein broke my nose.

(Aus „A Twenties Memory“ von Woody Allen, Random House, Inc., 1971)

Kryptoanalyse affin-linearer Blockchiffren

Affin-lineare Blockchiffren können leicht durch Known-Plaintext-Angriffe gebrochen werden, d.h., für einen Angreifer, der einige Klartexte mit den zugehörigen Schlüsseltexten kennt, ist es nicht allzu schwer, die Schlüssel zu bestimmen, die bei der Verschlüsselung dieser Klartexte verwendet wurden, und da es sich um symmetrische Systeme handelt, kann er leicht den Entschlüsselungsschlüssel bestimmen und weitere Schlüsseltexte zu entschlüsseln. Noch anfälliger sind affin-lineare Blockchiffren gegen Chosen-Plaintext-Angriffe, bei welchen der Angreifer einige Klartexte selbst wählen kann und die entsprechenden Schlüsseltexte erfährt, was dann nützlich sein kann, wenn er bestimmte Vermutungen über die verwendeten Schlüssel hat. Im Folgenden wird ein Known-Plaintext-Angriff beschrieben.

Tabelle 4.12. Kasiskis Methode: entschlüsselter Vigenère-Schlüsseltext

Schlüssel	P A U L A P A U L A P A U L A P A U L A
Klartext	W E H A D G R E A T F U N I N S P A I N
Schlüsseltext	L E B L D V R Y L T U U H T N H P U T N
Klartext	T H A T Y E A R A N D W E T R A V E L L
Schlüsseltext	I H U E Y T A L L N S W Y E R P V Y W L
Klartext	E D A N D W R O T E A N D H E M I N G W
Schlüsseltext	T D U Y D L R I E E P N X S E B I H R W
Klartext	A Y T O O K M E T U N A F I S H I N G A
Schlüsseltext	P Y N Z O Z M Y E U C A Z T S W I H R A
Klartext	N D I C A U G H T F O U R C A N S A N D
Schlüsseltext	C D C N A J G B E F D U L N A C S U Y D
Klartext	W E L A U G H E D A N D A L I C E T O K
Schlüsseltext	L E F L U V H Y O A C D U W I R E N Z K
Klartext	L A S A S K E D M E I F I W A S I N L O
Schlüsseltext	A A M L S Z E X X E X F C H A K I H W O
Klartext	V E W I T H G E R T R U D E S T E I N B
Schlüsseltext	K E Q T T W G Y C T G U X P S I E C Y B
Klartext	E C A U S E I H A D D E D I C A T E D A
Schlüsseltext	T C U F S T I B L D S E X T C P T Y O A
Klartext	B O O K O F P O E M S T O H E R E V E N
Schlüsseltext	Q O I V O U P I P M H T I S E G E P P N
Klartext	T H O U G H T H E Y W E R E T S E L I O
Schlüsseltext	I H I F G W T B P Y L E L P T H E F T O
Klartext	T S A N D I S A I D Y E S I L O V E D H
Schlüsseltext	I S U Y D X S U T D N E M T L D V Y O H
Klartext	E R B U T I T C O U L D N E V E R W O R
Schlüsseltext	T R V F T X T W Z U A D H P V T R Q Z R
Klartext	K B E C A U S E S H E W A S F A R T O O
Schlüsseltext	Z B Y N A J S Y D H T W U D F P R N Z O
Klartext	I N T E L L I G E N T F O R M E A N D A
Schlüsseltext	X N X P L A I A P N I F I C M T A H O A
Klartext	L I C E T O K L A S A G R E E D A N D T
Schlüsseltext	A I W P T D K F L S P G L P E S A H O T
Klartext	H E N W E P U T O N S O M E B O X I N G
Schlüsseltext	W E H H E E U N Z N H O G P B D X C Y G
Klartext	G L O V E S A N D G E R T R U D E S T E
Schlüsseltext	V L I G E H A H O G T R N C U S E M E E
Klartext	I N B R O K E M Y N O S E
Schlüsseltext	X N V C O Z E G J N D S Y

Beispiel 4.16 (Known-Plaintext-Angriff auf affin-lineare Blockchiffren). Sei (A, \mathbf{b}) ein fest gewählter Schlüssel. Der Klartext $\mathbf{x} \in \mathbb{Z}_m^n$ wird dann als $\mathbf{y} = E_{(A, \mathbf{b})}(\mathbf{x}) = A\mathbf{x} + \mathbf{b} \pmod{m}$ verschlüsselt, wobei A mit $\text{ggT}(\det A, m) = 1$ eine $(n \times n)$ -Matrix über \mathbb{Z}_m ist und \mathbf{y} und \mathbf{b} Vektoren in \mathbb{Z}_m^n sind. Wie bisher wird in der Arithmetik modulo m gearbeitet.

Angenommen, der Kryptoanalytiker kennt $n + 1$ Klartexte $\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_n$ und die zugehörigen Schlüsseltexte $\mathbf{y}_0, \mathbf{y}_1, \dots, \mathbf{y}_n$ mit

$$\mathbf{y}_i = A\mathbf{x}_i + \mathbf{b} \bmod m.$$

Es folgt

$$\mathbf{y}_i - \mathbf{y}_0 \equiv A(\mathbf{x}_i - \mathbf{x}_0) \bmod m. \quad (4.6)$$

Definiere die Matrizen X und Y durch

$$X = (\mathbf{x}_1 - \mathbf{x}_0, \mathbf{x}_2 - \mathbf{x}_0, \dots, \mathbf{x}_n - \mathbf{x}_0) \bmod m;$$

$$Y = (\mathbf{y}_1 - \mathbf{y}_0, \mathbf{y}_2 - \mathbf{y}_0, \dots, \mathbf{y}_n - \mathbf{y}_0) \bmod m.$$

Das heißt, die i -te Spalte von X ist die Differenz $\mathbf{x}_i - \mathbf{x}_0 \bmod m$, und die i -te Spalte von Y ist die Differenz $\mathbf{y}_i - \mathbf{y}_0 \bmod m$, wobei $1 \leq i \leq n$. Aus (4.6) folgt, dass

$$AX \equiv Y \bmod m$$

gilt. Sind $\det X$ und m teilerfremd, dann gilt $X^{-1} = (\det X)^{-1}X_{\text{adj}}$, wobei $(\det X)^{-1}$ die Inverse von $\det X \bmod m$ bezeichnet. Somit erhalten wir

$$A \equiv Y((\det X)^{-1}X_{\text{adj}}) \bmod m.$$

Da

$$\mathbf{b} = (\mathbf{y}_0 - A\mathbf{x}_0) \bmod m$$

gilt, haben wir also den Schlüssel (A, \mathbf{b}) aus $n + 1$ Paaren von Klartexten und zugehörigen Schlüsseltexten bestimmt.

Ist das Kryptosystem sogar linear, so ist $\mathbf{b} = 0$, und wir dürfen $\mathbf{x}_0 = \mathbf{y}_0 = 0$ wählen. Ist beispielsweise $n = 2$ und wurden zwei Paare von Klartexten mit den zugehörigen Schlüsseltexten abgefangen, etwa die ersten beiden Blöcke der Verschlüsselung durch die Hill-Chiffre in Tabelle 4.8 aus Beispiel 4.13, dann zeigt Tabelle 4.13 diese beiden bekannten Paare: $\mathbf{x}_1 = (19, 7)$ und $\mathbf{y}_1 = (27, 2)$, sowie $\mathbf{x}_2 = (4, 26)$ und $\mathbf{y}_2 = (0, 22)$.

Tabelle 4.13. Known-Plaintext-Angriff auf die Hill-Chiffre

Klartext	T	H	E	□
codierter Klartext	19	7	4	26
codierter Schlüsseltext	27	2	0	22
Schlüsseltext	,	C	A	W

Es ergeben sich die Matrizen $X = \begin{pmatrix} 19 & 4 \\ 7 & 26 \end{pmatrix}$ und $Y = \begin{pmatrix} 27 & 0 \\ 2 & 22 \end{pmatrix}$. Da $\det X = 19 \cdot 26 - 4 \cdot 7 = 2$ und $m = 29$ teilerfremd sind, erhält man weiter $(\det X)^{-1} = 15$ und

$$X_{\text{adj}} = \begin{pmatrix} 26 & -4 \\ -7 & 19 \end{pmatrix} = \begin{pmatrix} 26 & 25 \\ 22 & 19 \end{pmatrix}.$$

Folglich kann der verwendete Schlüssel wie folgt ermittelt werden:

$$\begin{aligned} A &\equiv Y((\det X)^{-1} X_{\text{adj}}) \bmod m \\ &= \begin{pmatrix} 27 & 0 \\ 2 & 22 \end{pmatrix} \left(15 \begin{pmatrix} 26 & 25 \\ 22 & 19 \end{pmatrix} \right) = \begin{pmatrix} 27 & 0 \\ 2 & 22 \end{pmatrix} \begin{pmatrix} 13 & 27 \\ 11 & 24 \end{pmatrix} \\ &= \begin{pmatrix} 3 & 4 \\ 7 & 2 \end{pmatrix}. \end{aligned}$$

4.2.3 Block- und Stromchiffren

In den Abschnitten 4.2.1 und 4.2.2 wurden verschiedene Blockchiffren vorgestellt. In diesem Abschnitt werden mehrere Verfahren diskutiert, wie man mittels Blockchiffren lange Klartexte verschlüsseln kann. Außerdem werden Blockchiffren den Stromchiffren im Vergleich gegenübergestellt.

Tripel-Verschlüsselung

Die Sicherheit einer Blockchiffre kann dadurch erhöht werden, dass sie mehrmals mit verschiedenen Schlüsseln angewandt wird. Diese Methode kann den Schlüsselraum beträchtlich vergrößern. Eine übliche Art, dies zu tun, stellt die *Tripel-Verschlüsselung* dar. Dabei wählt man zunächst drei Schlüssel, etwa k_1 , k_2 und k_3 . Dann verschlüsselt man den Klartext x durch

$$y = E_{k_1}(D_{k_2}(E_{k_3}(x))),$$

wobei E_{k_i} die Verschlüsselungsfunktion und D_{k_i} die Entschlüsselungsfunktion für k_i ist, $i \in \{1, 2, 3\}$. Der Schlüsseltext y kann dann durch

$$x = D_{k_3}(E_{k_2}(D_{k_1}(y)))$$

wieder entschlüsselt werden.

ECB-Modus

Eine Blockchiffre mit Blocklänge n sei gegeben. Nachrichten seien Wörter in Σ^* , wobei Σ ein Alphabet ist, und der Schlüsselraum sei K . Für jeden Schlüssel $k \in K$ sei E_k die Verschlüsselungsfunktion und D_k die Entschlüsselungsfunktion.

Um einen Klartext m im „electronic codebook mode“ (kurz ECB) zu verschlüsseln, wird er in Blöcke der Länge n unterteilt, wobei der letzte Block womöglich durch zufällig gewählte Buchstaben aufgefüllt werden muss, um sicherzustellen, dass n ein Teiler von $|m|$ ist. Ist $e \in K$ der Verschlüsselungsschlüssel, so wird jeder Block der Länge n mit e verschlüsselt. Der Schlüsseltext ist die resultierende Folge von

Schlüsseltextblöcken. Ist $d \in K$ der zu e gehörige Entschlüsselungsschlüssel, so werden die Schlüsseltextblöcke nacheinander mit d entschlüsselt, woraus der ursprüngliche Klartext m hervorgeht. Sämtliche Beispiele von Blockchiffren in den Abschnitten 4.2.1 und 4.2.2 wurden im ECB-Modus verschlüsselt.

Ein offensichtlicher Nachteil des ECB-Modus ist, dass gleiche Klartextblöcke in dieselben Schlüsseltextblöcke verschlüsselt werden. So können Regelmäßigkeiten im Klartext zu Regelmäßigkeiten im Schlüsseltext führen. Ein Kryptoanalytiker kann diese aus dem Schlüsseltext erhaltene Information möglicherweise ausnutzen und die Chiffre brechen. Sieht man sich etwa Beispiel 4.15 an, in welchem Kasiskis Methode zum Brechen der Vigenère-Chiffre beschrieben wird, so resultieren die in Tabelle 4.10 hervorgehobenen Muster im Schlüsseltext aus der Verwendung des ECB-Modus für die Vigenère-Chiffre. Insbesondere verschlüsseln die Muster „AHO“, „UYD“ und „ACD“ im Schlüsseltext jeweils den Klartext „AND“.

Ein weiterer Nachteil des ECB-Modus ist, dass sich ein Angreifer an den verschlüsselten Nachrichten bei der Übermittlung zu schaffen machen kann. Schlüsseltextblöcke können gelöscht oder ihre Reihenfolge kann verändert werden, oder es können zusätzliche Schlüsseltextblöcke eingefügt werden, falls der Schlüssel ermittelt worden ist. In jedem dieser Fälle wird der Empfänger bei der Entschlüsselung eine andere als die ursprüngliche Nachricht erhalten. Deshalb ist der ECB-Modus für die Verschlüsselung langer Nachrichten nicht zu empfehlen. Die Sicherheit des ECB-Modus kann dadurch erhöht werden, dass in alle Klartextblöcke zufällig gewählte Zeichen eingefügt werden.

Beispiel 4.17 (ECB-Modus). Betrachte die Permutationschiffre mit der Blocklänge 5 und dem Alphabet $\Sigma = \{0, 1\}$. Der Schlüsselraum ist $K = \mathfrak{S}_5$. Für jeden Schlüssel $\pi \in \mathfrak{S}_5$ ist die Verschlüsselungsfunktion $E_\pi : \Sigma^5 \rightarrow \Sigma^5$ definiert durch

$$E_\pi(x_1x_2 \cdots x_5) = x_{\pi(1)}x_{\pi(2)} \cdots x_{\pi(5)}.$$

Ist etwa $m = 100111010101001001$ die zu verschlüsselnde Nachricht, so unterteilt man sie in vier Blöcke der Länge 5, wobei der letzte Block mit einem Suffix von zwei Nullen aufgefüllt wird:

$$m = 10011 \ 10101 \ 01001 \ 00100.$$

So erhält man die Blöcke $\mathbf{b}_1 = 10011$, $\mathbf{b}_2 = 10101$, $\mathbf{b}_3 = 01001$ und $\mathbf{b}_4 = 00100$. Ist der Schlüssel durch $\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 1 & 5 & 2 \end{pmatrix}$ gegeben, so erhält man die vier Schlüsseltextblöcke

$$\begin{array}{ll} \mathbf{c}_1 = E_\pi(\mathbf{b}_1) = 01110 & \mathbf{c}_2 = E_\pi(\mathbf{b}_2) = 10110 \\ \mathbf{c}_3 = E_\pi(\mathbf{b}_3) = 00011 & \mathbf{c}_4 = E_\pi(\mathbf{b}_4) = 10000 \end{array}$$

und somit den Schlüsseltext

$$c = 01110 \ 10110 \ 00011 \ 10000.$$

CBC-Modus

Der „*cipherblock chaining mode*“ (kurz CBC) umgeht die Nachteile des ECB-Modus, indem er in einer „kontextsensitiven“ Weise vorgeht; vergleiche dazu den Begriff der kontextsensitiven Grammatik aus Definition 2.9. Nämlich hängt die Verschlüsselung eines Klartextblockes im CBC-Modus nicht nur vom zu verschlüsselnden Block und dem Schlüssel ab, sondern auch von vorherigen Blöcken. Folglich werden identische Muster im Klartext in Abhängigkeit von ihrem Kontext unterschiedlich verschlüsselt. Hat sich ein Angreifer am Schlüsseltext zu schaffen gemacht, so kann in der Regel nicht mehr richtig entschlüsselt werden, wodurch der Angriffsversuch aufgedeckt wird. Der CBC-Modus wird nun am folgenden Beispiel genauer erläutert.

Beispiel 4.18 (CBC-Modus). Seien $\Sigma = \{0, 1\}$ ein Alphabet, n die Blocklänge und \mathfrak{S}_n der Schlüsselraum. Betrachte erneut die Permutationschiffre mit der Verschlüsselungsfunktion E_π und der Entschlüsselungsfunktion $D_{\pi^{-1}}$ für den Schlüssel $\pi \in \mathfrak{S}_n$. Definiere die logische Operation *Exklusives-Oder*, $\oplus : \{0, 1\}^2 \rightarrow \{0, 1\}$, durch die in Tabelle 4.14 gezeigte Wahrheitstafel. Diese Operation \oplus entspricht der Addition von Bitvektoren im Restklassenkörper \mathbb{Z}_2 . Gegeben zwei Vektoren $\mathbf{x} = (x_1, x_2, \dots, x_n)$ und $\mathbf{y} = (y_1, y_2, \dots, y_n)$ in $\{0, 1\}^n$, definiere $\mathbf{x} \oplus \mathbf{y} = (x_1 \oplus y_1, x_2 \oplus y_2, \dots, x_n \oplus y_n)$. Aus Bequemlichkeit schreiben wir Vektoren aus $\{0, 1\}^n$ hier als Wörter, verzichten also auf Klammern und Kommas. Sind etwa $\mathbf{x} = 01100$ und $\mathbf{y} = 11001$ gegeben, so ist $\mathbf{x} \oplus \mathbf{y} = 10101$.

Tabelle 4.14. Wahrheitstafel der Operation *Exklusives-Oder*

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Sei $n = 5$. Betrachte den Schlüssel $\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 5 & 1 & 2 & 4 \end{pmatrix} \in \mathfrak{S}_5$ und dieselbe Nachricht wie in Beispiel 4.17:

$$m = 10011 \ 10101 \ 01001 \ 00100,$$

die aus den Blöcken $\mathbf{b}_1 = 10011$, $\mathbf{b}_2 = 10101$, $\mathbf{b}_3 = 01001$ und $\mathbf{b}_4 = 00100$ besteht. Um den CBC-Modus zu beschreiben, wählen wir einen Anfangsvektor $\mathbf{c}_0 = 11010$ in $\{0, 1\}^5$. Für i mit $1 \leq i \leq 4$ wird der i -te Block durch

$$\mathbf{c}_i = E_\pi(\mathbf{c}_{i-1} \oplus \mathbf{b}_i)$$

verschlüsselt. Man erhält die vier Schlüsseltextblöcke:

$$\begin{array}{ll} \mathbf{c}_1 = E_\pi(\mathbf{c}_0 \oplus \mathbf{b}_1) = 01010 & \mathbf{c}_2 = E_\pi(\mathbf{c}_1 \oplus \mathbf{b}_2) = 11111 \\ \mathbf{c}_3 = E_\pi(\mathbf{c}_2 \oplus \mathbf{b}_3) = 10101 & \mathbf{c}_4 = E_\pi(\mathbf{c}_3 \oplus \mathbf{b}_4) = 01100. \end{array}$$

Folglich ist der Schlüsseltext:

$$c = 01010\ 11111\ 10101\ 01100.$$

Die inverse Permutation $\pi^{-1} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 1 & 5 & 2 \end{pmatrix}$ liefert den Entschlüsselungsschlüssel, welcher $D_{\pi^{-1}}(E_\pi(\mathbf{b})) = \mathbf{b}$ für jeden Block \mathbf{b} erfüllt. Für i mit $1 \leq i \leq 4$ wird der i -te Schlüsseltextblock durch

$$\mathbf{b}_i = \mathbf{c}_{i-1} \oplus D_{\pi^{-1}}(\mathbf{c}_i)$$

verschlüsselt. Tatsächlich ergibt sich

$$\mathbf{b}_1 = \mathbf{c}_0 \oplus D_{\pi^{-1}}(\mathbf{c}_1) = 11010 \oplus D_{\pi^{-1}}(01010) = 11010 \oplus 01001 = 10011,$$

und dass die anderen Blöcke korrekt entschlüsselt werden, kann in ähnlicher Weise gezeigt werden.

Identische Nachrichten werden im CBC-Modus durch Veränderung des Anfangsvektors unterschiedlich verschlüsselt. Da die Verschlüsselung von Klartextblöcken außerdem von den vorhergehenden Blöcken abhängt, werden gleiche Klartextblöcke in verschiedenen Kontexten unterschiedlich verschlüsselt. Versuche, die übermittelten Schlüsseltextblöcke zu verfälschen, etwa durch Veränderung ihrer Reihenfolge oder durch das Löschen oder Einfügen von Blöcken, werden entdeckt werden, weil eine korrekte (authorisierte) Entschlüsselung dadurch in der Regel verhindert wird. Das ist ein offensichtlicher Vorteil des CBC-Modus gegenüber dem ECB-Modus. Daher ist der CBC-Modus für das Verschlüsseln langer Texte mittels einer Blockchiffre sehr geeignet.

Der Nachteil des CBC-Modus besteht darin, dass der Empfänger stets auf den nächsten Schlüsseltextblock warten muss, bevor er mit der Entschlüsselung beginnen kann. Diese Verzögerungen bewirken eine gewisse Ineffizienz, insbesondere dann, wenn die Blocklänge sehr groß ist.

CFB-Modus

Der oben erwähnte Nachteil des CBC-Modus kann beim „cipher feedback mode“ (kurz CFB) vermieden werden. Die Idee ist einfach, die Nachricht in Blöcke zu unterteilen, die kürzer als die Blocklänge n der verwendeten Blockchiffre sind, und nicht die eigentliche Verschlüsselungsfunktion der Blockchiffre zu verwenden, sondern stattdessen diese kürzeren Blöcke dadurch zu verschlüsseln, dass sie mit geeigneten Schlüsselblöcken modulo 2 addiert werden. Diese Schlüsselblöcke können vom Empfänger und Sender des Schlüsseltextes nahezu gleichzeitig erzeugt werden. Der CFB-Modus wird anhand des folgenden Beispiels erklärt.

Beispiel 4.19 (CFB-Modus). Betrachte wieder die Permutationschiffre mit dem Alphabet $\Sigma = \{0, 1\}$, der Blocklänge n und dem Schlüsselraum \mathfrak{S}_n . Wähle außerdem ein k mit $1 \leq k \leq n$ und einen Anfangsvektor $\mathbf{z}_0 \in \{0, 1\}^n$. Die Nachricht m jedoch wird diesmal in $d = \lceil |m|/k \rceil$ Blöcke $\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_d$ der Länge k unterteilt. Für $i = 1, 2, \dots, d$ werden nacheinander jeweils die folgenden Schritte ausgeführt:

Schritt 1: Berechne $\mathbf{x}_i = E_\pi(\mathbf{z}_{i-1})$.

Schritt 2: Sei \mathbf{y}_i das Wort in $\{0,1\}^k$, das aus den ersten k Bits von $\mathbf{x}_i \in \{0,1\}^n$ besteht.

Schritt 3: Berechne $\mathbf{c}_i = \mathbf{b}_i \oplus \mathbf{y}_i$.

Schritt 4: Berechne $\mathbf{z}_i = 2^k \mathbf{z}_{i-1} + \mathbf{c}_i \bmod 2^n$, d.h., die ersten k Bits werden in \mathbf{z}_{i-1} gelöscht und \mathbf{c}_i wird als ein Suffix angehängt.

Der sich ergebende Schlüsseltext besteht aus den Blöcken $\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_d$.

Für $n = 5$ und $k = 4$ betrachten wir die Nachricht aus den Beispielen 4.17 und 4.18: $m = 10011 10101 01001 00100$. Unterteile die Nachricht in fünf Blöcke der Länge k : $\mathbf{b}_1 = 1001$, $\mathbf{b}_2 = 1101$, $\mathbf{b}_3 = 0101$, $\mathbf{b}_4 = 0010$ und $\mathbf{b}_5 = 0100$.

Ist $\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 5 & 1 & 2 & 4 \end{pmatrix} \in \mathfrak{S}_5$ wieder unser Schlüssel und ist $\mathbf{z}_0 = 11010$ unser Anfangsvektor, so verschlüsseln wir diese Blöcke wie in Tabelle 4.15 dargestellt.

Tabelle 4.15. Blockverschlüsselung im CFB-Modus

i	\mathbf{b}_i	\mathbf{x}_i	\mathbf{y}_i	\mathbf{c}_i	\mathbf{z}_i
0	—	—	—	—	11010
1	1001	00111	0011	1010	01010
2	1101	00011	0001	1100	01100
3	0101	10010	1001	1100	01100
4	0010	10010	1001	1011	01011
5	0100	01011	0101	0001	10001

Die Entschlüsselung funktioniert fast genauso wie die Verschlüsselung. Der einzige Unterschied tritt im dritten Schritt auf. Für $i = 1, 2, \dots, d$ werden nacheinander jeweils die folgenden Schritte ausgeführt:

Schritt 1: Berechne $\mathbf{x}_i = E_\pi(\mathbf{z}_{i-1})$.

Schritt 2: Sei \mathbf{y}_i das Wort in $\{0,1\}^k$, das aus den ersten k Bits von $\mathbf{x}_i \in \{0,1\}^n$ besteht.

Schritt 3: Berechne $\mathbf{b}_i = \mathbf{c}_i \oplus \mathbf{y}_i$.

Schritt 4: Berechne $\mathbf{z}_i = 2^k \mathbf{z}_{i-1} + \mathbf{c}_i \bmod 2^n$, d.h., die ersten k Bits werden in \mathbf{z}_{i-1} gelöscht und \mathbf{c}_i wird als ein Suffix angehängt.

Die entschlüsselte Nachricht besteht aus den Blöcken $\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_d$.

Offenbar können Sender und Empfänger \mathbf{y}_1 bestimmen, sobald der Anfangsvektor \mathbf{z}_0 gewählt ist. Dann berechnet der Sender $\mathbf{c}_1 = \mathbf{b}_1 \oplus \mathbf{y}_1$ und schickt es ab, und der Empfänger berechnet $\mathbf{b}_1 = \mathbf{c}_1 \oplus \mathbf{y}_1$. Anschließend können beide \mathbf{y}_2 bestimmen und so weiter. Der im Vergleich mit dem CBC-Modus erlangte Vorteil ist, dass die Blocklänge k viel kürzer als die eigentliche Blocklänge n sein kann. Somit wird viel weniger Zeit untätig damit verschwendet, dass der Empfänger warten muss, bis der Sender alle Berechnungen durchgeführt hat, und beide können nahezu simultan ver- und entschlüsseln.

OFB-Modus

Der „*output feedback mode*“ (kurz OFB) ist dem CFB-Modus ganz ähnlich. Die Initialisierung und die ersten drei Schritte sowohl der Verschlüsselungs- als auch der Entschlüsselungsprozedur sind identisch. Der einzige Unterschied tritt im vierten Schritt auf, welcher die Vektoren \mathbf{z}_i für $1 \leq i \leq d$ bestimmt. Bei der Verschlüsselung geht man im OFB-Modus wie folgt vor:

Schritt 1: Berechne $\mathbf{x}_i = E_\pi(\mathbf{z}_{i-1})$.

Schritt 2: Sei \mathbf{y}_i das Wort in $\{0,1\}^k$, das aus den ersten k Bits von $\mathbf{x}_i \in \{0,1\}^n$ besteht.

Schritt 3: Berechne $\mathbf{c}_i = \mathbf{b}_i \oplus \mathbf{y}_i$.

Schritt 4: Berechne $\mathbf{z}_i = \mathbf{x}_i$.

Beispiel 4.20 (OFB-Modus). Tabelle 4.15 in Beispiel 4.19 zeigte die Blockverschlüsselung im CFB-Modus. Tabelle 4.16 unten gibt die entsprechende Blockverschlüsselung im OFB-Modus für dieselbe Nachricht und denselben Schlüssel und Anfangsvektor an.

Tabelle 4.16. Blockverschlüsselung im OFB-Modus

i	\mathbf{b}_i	\mathbf{x}_i	\mathbf{y}_i	\mathbf{c}_i	\mathbf{z}_i
0	—	—	—	—	11010
1	1001	00111	0011	1010	00111
2	1101	11001	1100	0001	11001
3	0101	01110	0111	0010	01110
4	0010	10011	1001	1011	10011
5	0100	01101	0110	0010	01101

Ein Vorteil des OFB-Modus gegenüber dem CFB-Modus ist, dass möglicherweise auftretende Übertragungsfehler im OFB-Modus weniger Schaden anrichten: Fehler treten bei der Entschlüsselung nur an genau dieser Stelle auf. Im Gegensatz dazu treten von einem Übertragungsfehler verursachte Entschlüsselungsfehler im CFB-Modus so lange auf, bis der fehlerhafte Block aus dem Vektor \mathbf{z}_i heraus verschoben wurde, was von den Blocklängen n und k abhängt.

Übung 4.7 liefert weitere Beispiele für die verschiedenen Möglichkeiten, Blockchiffren anzuwenden.

Stromchiffren

Blockchiffren, wie etwa die Vigenère-Chiffre oder die Hill-Chiffre, unterteilen den Klartext in Blöcke gleicher Länge und verschlüsseln jeden Block mit demselben Schlüssel. Allerdings können Blockchiffren auch verwendet werden, um die Klartextblöcke in kontextsensitiver Weise zu verschlüsseln. Im CBC-Modus zum Beispiel hängt die Verschlüsselung von Blöcken von den vorhergehenden Blöcken ab.

Dieses Prinzip wird durch den Begriff der *Stromchiffre* verallgemeinert. Stromchiffren erzeugen einen kontinuierlichen Strom von Schlüsseln, so dass jeder Schlüssel von den vorhergehenden Schlüsseln und vom Kontext im bereits verschlüsselten Klartext abhängen kann.

Das nächste Beispiel stellt eine populäre Stromchiffre vor, die auf einem linearen Rückkopplungsschieberegister („*linear feedback shift register*“) beruht, und erklärt so die allgemeine Idee der Stromchiffren. Beispiele kryptoanalytischer Angriffe auf Stromchiffren können in den Übungen 4.8 und 4.9 gefunden werden.

Beispiel 4.21 (Stromchiffre). Das Alphabet $\Sigma = \{0, 1\}$ ist sowohl der Klartext- als auch der Schlüsselraum. Für festes $n \in \mathbb{N}$ ist Σ^n der Schlüsselraum. Eine Nachricht $\mathbf{m} = m_1 m_2 \cdots m_z$ in Σ^* wird Symbol für Symbol wie folgt verschlüsselt. Sei $z \geq n$. Erzeuge ausgehend von einem gegebenen Schlüssel

$$\mathbf{k} = (k_1, k_2, \dots, k_n)$$

in Σ^n einen Schlüsselstrom

$$\mathbf{s} = (s_1, s_2, \dots, s_z, \dots),$$

dessen erste n Bits durch \mathbf{k} initialisiert werden:

$$s_i = k_i \quad \text{für } 1 \leq i \leq n,$$

und der dann gemäß der folgenden linearen Rekursion der Ordnung n fortfährt:

$$s_i = \sum_{j=1}^n a_j s_{i-j} \bmod 2 \quad \text{für } i > n, \tag{4.7}$$

wobei $a_1, a_2, \dots, a_n \in \{0, 1\}$ feste Koeffizienten sind. Bezeichnet man die ersten z Bits des Schlüsselstroms \mathbf{s} mit $\mathbf{s}(z)$, so sind die Verschlüsselungsfunktion $E_{\mathbf{k}}$ und die Entschlüsselungsfunktion $D_{\mathbf{k}}$, die beide von Σ^* in Σ^* abbilden, wie folgt definiert:

$$\begin{aligned} E_{\mathbf{k}}(\mathbf{m}) &= \mathbf{m} \oplus \mathbf{s}(|\mathbf{m}|); \\ D_{\mathbf{k}}(\mathbf{c}) &= \mathbf{c} \oplus \mathbf{s}(|\mathbf{c}|), \end{aligned}$$

wobei \oplus die Addition von Bitvektoren modulo 2 bezeichnet. Das heißt, das i -te Bit von $\mathbf{m} \oplus \mathbf{s}$ ist $m_i \oplus s_i$, das exklusive Oder von m_i und s_i ; siehe Tabelle 4.14.

Als konkretes Beispiel wählen wir $n = 5$ und die Koeffizienten $a_1 = a_3 = a_4 = 0$ und $a_2 = a_5 = 1$. Dann wird der Schlüsselstrom \mathbf{s} durch die lineare Rekursion

$$s_{i+5} = s_{i+3} + s_i \bmod 2 \tag{4.8}$$

erzeugt. Wählt man den Schlüssel $\mathbf{k} = (1, 0, 0, 1, 1)$, so erhält man

$$\mathbf{s} = (1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, \dots).$$

Die lineare Rekursion aus (4.8) kann effizient durch einen Hardware-Baustein realisiert werden, nämlich ein lineares Rückkopplungsschieberegister, wie es in Abbildung 4.1 dargestellt ist. Die Register speichern die letzten vier Bits des erzeugten Schlüsselstroms s . In jedem Rekursionsschritt wird das Bit aus dem am weitesten links stehenden Register als der aktuelle Schlüssel verwendet. Dann werden die Bits in den übrigen Registern um je eine Position nach links verschoben. Dem am weitesten rechts stehenden Register wird nun das Bit gefüttert, das aus der Addition modulo 2 der Bits aus den Registern mit Koeffizienten $a_i = 1$ resultiert.

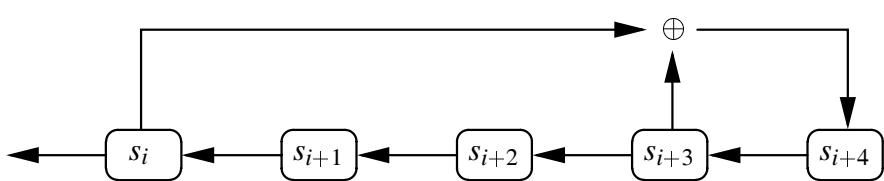


Abb. 4.1. Ein lineares Rückkopplungsschieberegister

4.3 Perfekte Geheimhaltung

Wie im letzten Abschnitt erwähnt, wurde die Vigenère-Chiffre als nicht zu brechen betrachtet, bis die raffinierten Ideen von Kasiski und Babbage zeigten, dass sie doch unsicher ist. Angesichts ihrer Leistung ist es natürlich, zu fragen, ob es irgendein Kryptosystem gibt, das beweisbar *perfekte Geheimhaltung* garantiert.

Dieser Frage werden wir uns in Abschnitt 4.3.1 zuwenden, die einen Teil der bahnbrechenden Arbeit von Claude Shannon [Sha49] beschreibt, der die Fundamente der modernen Codierungs- und Informationstheorie gelegt hat. Ein ganz zentraler Begriff in Shannons Werk ist der Entropie, welcher in der Physik, Mathematik, Informatik und in anderen Gebieten auftritt und sehr nützlich ist. In Abschnitt 4.3.2 werden die Begriffe der Entropie und der Schlüsselmehrdeutigkeit eingeführt und untersucht. Die hier wesentlichen Grundlagen der Wahrscheinlichkeitstheorie wurden in Abschnitt 2.5 beschrieben.

4.3.1 Satz von Shannon und Vernams One-Time Pad

Betrachte nochmals das Szenario in Abbildung 1.1 in Kapitel 1: Alice und Bob kommunizieren über einen unsicheren Kanal, wobei sie von Erich belauscht werden. Darauf verwenden sie ein Kryptosystem $\mathbf{S} = (M, C, K, \mathcal{E}, \mathcal{D})$, wobei M , C und K endliche Mengen und \mathcal{E} und \mathcal{D} Familien von Funktionen sind, die für die Verschlüsselung bzw. die Entschlüsselung verwendet werden; siehe Definition 4.1.

Wir gehen von den folgenden Vorbetrachtungen und Annahmen aus. Die Nachrichten sind auf M gemäß einer Wahrscheinlichkeitsverteilung \Pr_M verteilt, die von

der verwendeten natürlichen Sprache abhängen kann. Für jede neue Nachricht wählt Alice einen neuen Schlüssel aus K , der von der zu verschlüsselnden Nachricht unabhängig ist. Diese Annahme ist sinnvoll und bedeutet, dass Alice ihren Schlüssel wählt, bevor sie weiß, welcher Klartext verschlüsselt werden wird.

Die Schlüssel sind gemäß einer Wahrscheinlichkeitsverteilung \Pr_K auf K verteilt. Die Verteilungen \Pr_M und \Pr_K induzieren eine Wahrscheinlichkeitsverteilung $\Pr_{M \times K}$ auf $M \times K$. Lassen wir den Index $M \times K$ weg, so heißt das, dass für jede Nachricht $m \in M$ und für jeden Schlüssel $k \in K$

$$\Pr(m, k) = \Pr_{M \times K}(m, k) = \Pr_M(m) \Pr_K(k)$$

die Wahrscheinlichkeit dafür ist, dass die Nachricht m mit dem Schlüssel k verschlüsselt wird, wobei m und k unabhängig sind.

Es folgen einige weitere Fakten und Bezeichnungen:

- Für $m \in M$ bezeichne m das Ereignis $\{(m, k) \mid k \in K\}$. Dann ist $\Pr(m) = \Pr_M(m)$ die Wahrscheinlichkeit dafür, dass die Nachricht m verschlüsselt wird.
- Für $k \in K$ bezeichne k das Ereignis $\{(m, k) \mid m \in M\}$. Dann ist $\Pr(k) = \Pr_K(k)$ die Wahrscheinlichkeit dafür, dass der Schlüssel k verwendet wird.
- Für $c \in C$ bezeichne c das Ereignis $\{(m, k) \mid E_k(m) = c\}$. Dann ist $\Pr(m \mid c)$ die bedingte Wahrscheinlichkeit dafür, dass m verschlüsselt wurde, unter der Bedingung, dass der Schlüsseltext c empfangen wurde.

Angenommen, der Lauscher Erich hat einen Schlüsseltext c abgefangen, den Alice an Bob geschickt hat. Erich versucht, Informationen über die entsprechende Nachricht m herauszufinden, die durch c verschlüsselt ist. Da er weiß, welche Sprache Alice und Bob verwenden, kennt er auch die Wahrscheinlichkeitsverteilung \Pr_M .

Nun wird der Begriff der perfekten Geheimhaltung definiert.

Definition 4.22 (Perfekte Geheimhaltung). Ein Kryptosystem $\mathbf{S} = (M, C, K, \mathcal{E}, \mathcal{D})$ garantiert genau dann perfekte Geheimhaltung, wenn gilt:

$$(\forall m \in M) (\forall c \in C) [\Pr(m \mid c) = \Pr(m)].$$

Das heißt, ein Kryptosystem garantiert perfekte Geheimhaltung dann und nur dann, wenn das Ereignis, dass eine Nachricht m verschlüsselt wird, unabhängig von dem Ereignis ist, dass ein Schlüsseltext c empfangen wird: Die Kenntnis von c verrät Erich nichts über m .

Nun betrachten wir ein Beispiel eines Kryptosystems, das keine perfekte Geheimhaltung garantiert. Gemäß Übung 4.10 soll dieses Kryptosystem so abgeändert werden, dass es doch perfekte Geheimhaltung leistet. Beispiel 4.23 wird in den Beispielen 4.28 und 4.36 fortgesetzt.

Beispiel 4.23 (Perfekte Geheimhaltung). Sei $\mathbf{S} = (M, C, K, \mathcal{E}, \mathcal{D})$ ein Kryptosystem mit Klartextraum M , Schlüsseltextraum C und Schlüsselraum K , wobei gilt:

- $M = \{a, b\}$ mit $\Pr(a) = 1/3$ und $\Pr(b) = 2/3$;
- $K = \{\$\#, \$\#\}$ mit $\Pr(\$) = 1/4$ und $\Pr(\#) = 3/4$;

- $C = \{x, y\}$.

Die Wahrscheinlichkeiten dafür, dass ein Klartextbuchstabe $m \in M$ mit einem Schlüssel $k \in K$ verschlüsselt wird, sind:

$$\begin{aligned}\Pr(a, \$) &= \Pr(a) \cdot \Pr(\$) = \frac{1}{3} \cdot \frac{1}{4} = \frac{1}{12}; \\ \Pr(a, \#) &= \Pr(a) \cdot \Pr(\#) = \frac{1}{3} \cdot \frac{3}{4} = \frac{1}{4}; \\ \Pr(b, \$) &= \Pr(b) \cdot \Pr(\$) = \frac{2}{3} \cdot \frac{1}{4} = \frac{1}{6}; \\ \Pr(b, \#) &= \Pr(b) \cdot \Pr(\#) = \frac{2}{3} \cdot \frac{3}{4} = \frac{1}{2}.\end{aligned}$$

Seien die Verschlüsselungsfunktionen folgendermaßen gegeben:

$$E_{\$}(a) = y; \quad E_{\$}(b) = x; \quad E_{\#}(a) = x; \quad E_{\#}(b) = y.$$

Dann ergeben sich die folgenden Wahrscheinlichkeiten dafür, dass ein Schlüsseltextbuchstabe $c \in C$ auftritt:

$$\begin{aligned}\Pr(x) &= \Pr(b, \$) + \Pr(a, \#) = \frac{1}{6} + \frac{1}{4} = \frac{5}{12}; \\ \Pr(y) &= \Pr(a, \$) + \Pr(b, \#) = \frac{1}{12} + \frac{1}{2} = \frac{7}{12}.\end{aligned}$$

Somit ist (sogar für jedes Paar $(m, c) \in M \times C$) die bedingte Wahrscheinlichkeit $\Pr(m | c)$ von der Wahrscheinlichkeit $\Pr(m)$ verschieden:

$$\begin{aligned}\Pr(a | x) &= \frac{\Pr(a, \#)}{\Pr(x)} = \frac{\frac{1}{4}}{\frac{5}{12}} = \frac{3}{5} \neq \frac{1}{3} = \Pr(a); \\ \Pr(a | y) &= \frac{\Pr(a, \$)}{\Pr(y)} = \frac{\frac{1}{6}}{\frac{7}{12}} = \frac{1}{7} \neq \frac{1}{3} = \Pr(a); \\ \Pr(b | x) &= \frac{\Pr(b, \$)}{\Pr(x)} = \frac{\frac{1}{6}}{\frac{5}{12}} = \frac{2}{5} \neq \frac{2}{3} = \Pr(b); \\ \Pr(b | y) &= \frac{\Pr(b, \#)}{\Pr(y)} = \frac{\frac{1}{4}}{\frac{7}{12}} = \frac{6}{7} \neq \frac{2}{3} = \Pr(b).\end{aligned}$$

Es folgt, dass das gegebene Kryptosystem keine perfekte Geheimhaltung leistet. Sieht Erich insbesondere den Schlüsseltextbuchstaben y , so wird er mit der Vermutung, dass der verschlüsselte Klartextbuchstabe ein b war, höchstwahrscheinlich richtig liegen.

Der Satz von Shannon gibt notwendige und hinreichende Bedingungen dafür an, wann ein Kryptosystem perfekte Geheimhaltung leistet.

Satz 4.24 (Shannon). Sei $\mathbf{S} = (M, C, K, \mathcal{E}, \mathcal{D})$ ein Kryptosystem mit $\|M\| = \|C\| = \|K\| < \infty$ und $\Pr(m) > 0$ für jedes $m \in M$. \mathbf{S} garantiert genau dann perfekte Geheimhaltung, wenn

1. für jedes $m \in M$ und für jedes $c \in C$ ein eindeutiger Schlüssel $k \in K$ mit $E_k(m) = c$ existiert und
2. die Schlüssel in K gleichverteilt sind.

Beweis. Angenommen, ein Kryptosystem \mathbf{S} mit den vorausgesetzten Eigenschaften leistet perfekte Geheimhaltung. Zu zeigen ist, dass die beiden Bedingungen dann gelten.

Um die erste Bedingung zu beweisen, sei $m \in M$ eine beliebig, aber fest gewählte Nachricht. Angenommen, es gibt einen Schlüsseltext $c \in C$, so dass kein Schlüssel m in c verschlüsselt. Das heißt, für alle $k \in K$ gilt $E_k(m) \neq c$. Somit ist

$$\Pr(m) \neq 0 = \Pr(m|c).$$

Dann garantiert \mathbf{S} keine perfekte Geheimhaltung, im Widerspruch zur Annahme. Es folgt:

$$(\forall c \in C) (\exists k \in K) [E_k(m) = c].$$

Nun impliziert die Annahme $\|M\| = \|C\| = \|K\| < \infty$, dass es für jeden Schlüsseltext $c \in C$ einen eindeutigen Schlüssel k mit $E_k(m) = c$ gibt.

Um die zweite Bedingung zu beweisen, fixieren wir einen beliebigen Schlüsseltext $c \in C$. Für jedes $m \in M$ sei $k(m)$ der eindeutige Schlüssel k mit $E_k(m) = c$. Nach dem Satz von Bayes (siehe Satz 2.55) folgt, dass für jedes $m \in M$ gilt:

$$\Pr(m|c) = \frac{\Pr(c|m) \Pr(m)}{\Pr(c)} = \frac{\Pr(k(m)) \Pr(m)}{\Pr(c)}. \quad (4.9)$$

Da \mathbf{S} perfekte Geheimhaltung garantiert, gilt $\Pr(m|c) = \Pr(m)$. Nach (4.9) impliziert dies $\Pr(k(m)) = \Pr(c)$, und diese Gleichheit gilt unabhängig von m . Folglich sind die Wahrscheinlichkeiten $\Pr(k)$ für alle $k \in K$ gleich, woraus sich $\Pr(k) = 1/\|K\|$ ergibt. Die Schlüssel in K sind also gleichverteilt.

Sei umgekehrt angenommen, dass beide Bedingungen gelten. Zu zeigen ist, dass \mathbf{S} perfekte Geheimhaltung leistet. Sei $k = k(m, c)$ der eindeutige Schlüssel mit $E_k(m) = c$. Nach dem Satz von Bayes folgt:

$$\begin{aligned} \Pr(m|c) &= \frac{\Pr(m) \Pr(c|m)}{\Pr(c)} \\ &= \frac{\Pr(m) \Pr(k(m, c))}{\sum_{q \in M} \Pr(q) \Pr(k(q, c))}. \end{aligned} \quad (4.10)$$

Da die Schlüssel gemäß der zweiten Bedingung gleichverteilt sind, folgt

$$\Pr(k(m, c)) = \frac{1}{\|K\|}. \quad (4.11)$$

Außerdem gilt:

$$\sum_{q \in M} \Pr(q) \Pr(k(q, c)) = \frac{\sum_{q \in M} \Pr(q)}{\|K\|} = \frac{1}{\|K\|}. \quad (4.12)$$

Setzt man (4.11) und (4.12) in (4.10) ein, so erhält man

$$\Pr(m|c) = \Pr(m).$$

Folglich garantiert **S** perfekte Geheimhaltung. □

Mittels der Bedingungen in Satz 4.24 kann man Kryptosysteme mit perfekter Geheimhaltung entwerfen. Ein berühmtes solches Kryptosystem wurde 1917 von Gilbert Vernam erfunden. Sein symmetrisches Kryptosystem, eine Blockchiffre, ist als der One-time Pad bekannt.

Beispiel 4.25 (Vernams One-Time Pad). Fixiere das Alphabet $\Sigma = \{0, 1\}$ und definier den Klartext-, Schlüsseltext- und Schlüsselraum durch

$$M = C = K = \{0, 1\}^n$$

für eine Blocklänge $n \in \mathbb{N}$. Die Schlüssel seien auf $\{0, 1\}^n$ gleichverteilt.

Für jeden Schlüssel $\mathbf{k} \in \{0, 1\}^n$ sind die Verschlüsselungsfunktion $E_{\mathbf{k}}$ und die Entschlüsselungsfunktion $D_{\mathbf{k}}$, die beide von $\{0, 1\}^n$ in $\{0, 1\}^n$ abbilden, durch

$$\begin{aligned} E_{\mathbf{k}}(\mathbf{x}) &= \mathbf{x} \oplus \mathbf{k}; \\ D_{\mathbf{k}}(\mathbf{y}) &= \mathbf{y} \oplus \mathbf{k} \end{aligned}$$

definiert, wobei \oplus die bitweise Addition modulo 2 bezeichnet.

Nach dem Satz von Shannon leistet der One-time Pad perfekte Geheimhaltung, da es für jeden Klartext $\mathbf{x} \in \{0, 1\}^n$ und für jeden Schlüsseltext $\mathbf{y} \in \{0, 1\}^n$ einen eindeutigen Schlüssel $\mathbf{k} \in \{0, 1\}^n$ mit $\mathbf{y} = \mathbf{x} \oplus \mathbf{k}$ gibt, nämlich den Vektor $\mathbf{k} = \mathbf{x} \oplus \mathbf{y}$.

Der One-time Pad hat einige ernsthafte Nachteile, die seinen Gebrauch in den meisten praktischen Szenarien einschränken. Um perfekte Geheimhaltung zu garantieren, kann jeder Schlüssel lediglich einmal verwendet werden, und er muss so lang sein wie der Klartextblock, der durch ihn verschlüsselt wird. Würden Alice und Bob denselben Schlüssel zweimal (oder noch öfter) verwenden und mit ihm zwei (oder mehr) Blöcke verschlüsseln, würde der One-time Pad keine perfekte Geheimhaltung mehr leisten. Da für jeden Block der Nachricht ein neuer geheimer Schlüssel ausgetauscht werden muss, ist beim One-time Pad eine effiziente Schlüsselverwaltung natürlich schwierig. Dies ist ein Grund, weshalb er in kommerziellen Anwendungen nur von beschränktem Nutzen ist.

Schlimmer noch ist, dass der One-time Pad gegen Known-Plaintext-Angriffe unsicher ist, falls derselbe Schlüssel zweimal verwendet wird. Kennt Erich nämlich eine Nachricht \mathbf{x} und einen entsprechenden Schlüsseltext \mathbf{y} , so kann er den verwendeten Schlüssel einfach wie folgt bestimmen:

$$\mathbf{x} \oplus \mathbf{y} = \mathbf{x} \oplus \mathbf{x} \oplus \mathbf{k} = \mathbf{k}.$$

Dank seiner perfekten Geheimhaltung ist der One-time Pad jedoch trotz dieser Nachteile in realen Anwendungen eingesetzt worden, vor allem im diplomatischen und militärischen Bereich. Angeblich ist er für die Hotline zwischen Moskau und Washington verwendet worden, siehe S. 316 in [Sim79]. Wenn es auf unbedingte Sicherheit ankommt, sind Kryptosysteme, die wie der One-time Pad perfekte Geheimhaltung garantieren, offensichtlich von großer Bedeutung.

4.3.2 Entropie und Schlüsselmehrdeutigkeit

In der Diskussion der perfekten Geheimhaltung in Abschnitt 4.3.1 wurde angenommen, dass immer dann, wenn ein neuer Klartext verschlüsselt werden soll, ein neuer Schlüssel verwendet wird, wobei der Schlüssel und der Klartext unabhängige Zufallsvariablen sind. Im Gegensatz dazu ist die Frage nun, was eigentlich passiert, wenn mehr als ein Klartext mit demselben Schlüssel verschlüsselt wird. Wie viel Information kann ein Kryptoanalytiker aus einer (eventuell mehrfachen) Wiederverwendung von Schlüsseln gewinnen? Wie wahrscheinlich ist es, dass diese Information zum Erfolg eines Ciphertext-only-Angriffs führt? Was *ist* überhaupt „Information“, und wie kann man sie messen? Die Antwort auf diese Frage lautet: Entropie.

Entropie

Der Begriff der Entropie stammt aus der Physik (speziell aus der Thermodynamik), wo er verwendet wird, um – grob gesprochen – den Grad von Chaos (oder Unordnung, also das Gegenteil geordneter Struktur) im Universum oder in irgendeinem geordneten System zu beschreiben. In der Welt der Physik bezieht sich Entropie auf *manifeste* Ordnung oder Struktur, nicht zu verwechseln mit der regulären präzisen Bewegung mikroskopischer Teilchen, welche nicht als manifeste Ordnung verstanden wird. Das zweite Prinzip der Thermodynamik sagt, dass die Entropie in jedem geschlossenen System im Laufe der Zeit wächst, außer es handelt sich um ein „reversibles“ System, in welchem Fall die Entropie konstant bleibt.

Was hat dies mit Kryptographie zu tun? Entropie ist ein so fundamentaler Begriff, dass er nicht nur in der Physik auftritt, sondern auch in der Mathematik, der Informatik und anderen Gebieten. Beispielsweise ist sie ein zentraler Begriff in Shannons Informations- und Codierungstheorie und in der verwandten Theorie der Datenkompression. In der Algorithmik kann sie verwendet werden, um die Laufzeiten randomisierter Algorithmen zu analysieren, um die mittlere Suchzeit in der Datenstruktur der optimalen Suchbäume zu bestimmen oder die mittlere Länge verlinkter Listen für Hash-Tabellen abzuschätzen, die Kollisionen durch Verkettung auflösen, und so weiter. Immer dann, wenn ein Zufallsexperiment wie etwa ein Münzwurf ausgeführt wird, misst die Entropie die durch Kenntnis des Ausgangs dieses Experiments *gewonnene Information*. Anders gesagt ist die Entropie ein Maß für die *Ungewissheit*, die durch das Ausführen dieses Experiments beseitigt wird. Intuitiv entspricht Ungewissheit dem Chaos, wohingegen Information der geordneten Struktur entspricht. Dass Entropie auch in der Kryptographie nützlich ist, überrascht also nicht.

Nun wird der Begriff der Entropie definiert, und es werden einige nützliche Eigenschaften festgehalten, die später gebraucht werden. Betrachte ein Zufallsexperiment mit endlich vielen Ausgängen. Solch ein Ausgang wird als *Ereignis* bezeichnet. Treten diese Ereignisse etwa mit den Wahrscheinlichkeiten p_1, p_2, \dots, p_n ein, so kann diesem Zufallsexperiment eine bestimmte Zahl zugeordnet werden, nämlich seine Entropie.

Definition 4.26 (Entropie). Sei \mathbf{X} eine Zufallsvariable, die n mögliche Werte aus der Menge $X = \{x_1, x_2, \dots, x_n\}$ annehmen kann. Für jedes i mit $1 \leq i \leq n$ sei $p_i = \Pr(\mathbf{X} = x_i)$ die Wahrscheinlichkeit dafür, dass \mathbf{X} den Wert x_i annimmt. Definiere die Entropie von \mathbf{X} als

$$\mathcal{H}(\mathbf{X}) = - \sum_{i=1}^n p_i \log p_i. \quad (4.13)$$

Per Konvention wird dabei $0 \log 0 = 0$ gesetzt.

Formal hat die gewichtete Summe in (4.13) die Form eines Erwartungswertes, siehe Abschnitt 2.5. Entropie drückt den Grad an Ungewissheit aus, die mit dem Ausgang des betrachteten Experiments assoziiert wird, *bevor* es ausgeführt wurde. Umgekehrt kann sie als ein Maß für den mittleren Informationsgehalt aufgefasst werden, der durch die Ausführung des Experiments gewonnen wird, d.h., *nachdem* man seinen Ausgang kennt. Anders gesagt, tritt in diesem Experiment das i -te Ereignis ein, so ergibt sich die gewonnene Information als

$$-\log p_i = \log \frac{1}{p_i}.$$

Sind die Experimente unabhängig, dann multiplizieren sich ihre Wahrscheinlichkeiten. Nach den Rechenregeln des Logarithmus summieren sich die Beträge der gewonnenen Information demnach auf. Alle Logarithmen sind zur Basis 2. Wie üblich ist diese Wahl willkürlich, doch eine Änderung der Basis würde den Betrag der Entropie lediglich um einen konstanten Faktor ändern, vergleiche Übung 2.4.

Beispiel 4.27 (Entropie). Ein Münzwurf ist ein Zufallsexperiment mit zwei möglichen Ausgängen: Kopf oder Zahl. Sei \mathbf{X} eine Zufallsvariable, die den Ausgang dieses Experiments angibt: $\mathbf{X} = 1$, falls der Kopf nach dem Wurf oben liegt, und $\mathbf{X} = 0$, falls die Zahl oben liegt. Wenn es eine faire Münze ist, treten diese beiden Ereignisse mit denselben Wahrscheinlichkeit ein, $p_1 = p_2 = 1/2$. Die einfache Rechnung

$$\mathcal{H}(\mathbf{X}) = - \left(\frac{1}{2} \log \frac{1}{2} + \frac{1}{2} \log \frac{1}{2} \right) = \frac{1}{2} \log 2 + \frac{1}{2} \log 2 = 1$$

zeigt, dass das Zufallsexperiment eines Münzwurfs mit fairer Münze einen Informationsgehalt von genau einem Bit hat, welches entweder 1 für Kopf oder 0 für Zahl ist. Ist die Münze jedoch gezinkt, etwa mit den Wahrscheinlichkeiten $p_1 = 1/4$ für Kopf und $p_2 = 3/4$ für Zahl, so verringert sich die Entropie dieses Experiments auf etwa 0.8113 Bit.

Das folgende Beispiel illustriert, wie man die verschiedenen Komponenten eines Kryptosystems durch ihre Entropien beschreiben kann.

Beispiel 4.28 (Entropie eines Kryptosystem: Fortsetzung von Beispiel 4.23).

Betrachte das Kryptosystem $\mathbf{S} = (M, C, K, \mathcal{E}, \mathcal{D})$, das in Beispiel 4.23 definiert wurde. Man kann sich einen Schlüssel als eine Zufallsvariable \mathbf{K} vorstellen, die Werte in

$K = \{\$\#\}$ annimmt, und somit seine Entropie bestimmen. Ebenso seien \mathbf{M} und \mathbf{C} Zufallsvariablen, die den Klar- bzw. den Schlüsseltext beschreiben und Werte in M bzw. in C annehmen.

Die Entropien von \mathbf{K} , \mathbf{M} und \mathbf{C} können wie folgt berechnet werden:

$$\mathcal{H}(\mathbf{K}) = -\left(\frac{1}{4} \log \frac{1}{4} + \frac{3}{4} \log \frac{3}{4}\right) = \frac{1}{4} \log 4 + \frac{3}{4} \log \frac{4}{3} = 2 - \frac{3}{4} \log 3 \approx 0.8113;$$

$$\mathcal{H}(\mathbf{M}) = -\left(\frac{1}{3} \log \frac{1}{3} + \frac{2}{3} \log \frac{2}{3}\right) = \frac{1}{3} \log 3 + \frac{2}{3} \log \frac{3}{2} \approx 0.9183;$$

$$\mathcal{H}(\mathbf{C}) = -\left(\frac{5}{12} \log \frac{5}{12} + \frac{7}{12} \log \frac{7}{12}\right) = \frac{5}{12} \log \frac{12}{5} + \frac{7}{12} \log \frac{12}{7} \approx 0.9799.$$

Satz 4.31 unten stellt eine Reihe von nützlichen Eigenschaften der Entropie zusammen. Auf seinen Beweis wird hier verzichtet, bis auf den der ersten Aussage. Der Beweis der übrigen Aussagen wird dem Leser als Übung 4.12 überlassen. Um die erste Aussage von Satz 4.31 zu beweisen, wird die Ungleichung von Jensen benötigt, die unten ohne Beweis als Lemma 4.30 angegeben wird.

Definition 4.29. Eine reellwertige Funktion f heißt konkav auf dem Intervall $I \subseteq \mathbb{R}$, falls für alle $x, y \in I$ gilt:

$$f\left(\frac{x+y}{2}\right) \geq \frac{f(x) + f(y)}{2}.$$

Die Funktion f heißt streng konkav auf dem Intervall $I \subseteq \mathbb{R}$, falls für alle $x, y \in I$ mit $x \neq y$ gilt:

$$f\left(\frac{x+y}{2}\right) > \frac{f(x) + f(y)}{2}.$$

Lemma 4.30 (Ungleichung von Jensen). Ist f eine stetige Funktion, die auf dem reellen Intervall I streng konkav ist, und gilt $\sum_{i=1}^n p_i = 1$ für die positiven reellen Zahlen p_1, p_2, \dots, p_n , so ist

$$\sum_{i=1}^n p_i f(x_i) \leq f\left(\sum_{i=1}^n p_i x_i\right).$$

Satz 4.31 (Eigenschaften der Entropie). Sei \mathbf{X} eine Zufallsvariable, die n mögliche Werte aus der Menge $X = \{x_1, x_2, \dots, x_n\}$ annehmen kann. Für jedes i mit $1 \leq i \leq n$ sei $p_i = \Pr(\mathbf{X} = x_i)$ die Wahrscheinlichkeit dafür, dass \mathbf{X} den Wert x_i annimmt.

1. $\mathcal{H}(\mathbf{X}) \leq \log n$, wobei Gleichheit genau dann gilt, wenn für alle i , $1 \leq i \leq n$, $p_i = 1/n$ ist.
2. $\mathcal{H}(\mathbf{X}) \geq 0$, wobei Gleichheit genau dann gilt, wenn $p_i = 1$ für ein i und $p_j = 0$ für alle $j \neq i$ ist. Folglich bedeutet $\mathcal{H}(\mathbf{X}) = 0$, dass der Ausgang des Experiments vollständig bestimmt ist, d.h., es gibt keine Ungewissheit darüber bzw. es wird keine Information geliefert.

3. Ist \mathbf{Y} eine Zufallsvariable, die $n+1$ mögliche Werte aus der Menge $Y = \{y_1, y_2, \dots, y_{n+1}\}$ annehmen kann, so dass $\Pr(\mathbf{Y} = y_i) = p_i$ für $1 \leq i \leq n$ und $\Pr(\mathbf{Y} = y_{n+1}) = 0$ gilt, so ist $\mathcal{H}(\mathbf{Y}) = \mathcal{H}(\mathbf{X})$.
4. Ist $\pi \in \mathfrak{S}_n$ eine beliebige Permutation auf der Menge $\{1, 2, \dots, n\}$ und ist \mathbf{Y} eine Zufallsvariable mit $\Pr(\mathbf{Y} = x_i) = p_{\pi(i)}$, $1 \leq i \leq n$, dann gilt $\mathcal{H}(\mathbf{Y}) = \mathcal{H}(\mathbf{X})$.
5. Gruppierungseigenschaft: Sind \mathbf{Y} und \mathbf{Z} Zufallsvariablen, so dass \mathbf{Y} $n-1$ mögliche Werte mit den Wahrscheinlichkeiten $p_1 + p_2, p_3, \dots, p_n$ und \mathbf{Z} zwei mögliche Werte mit den Wahrscheinlichkeiten $p_1/(p_1 + p_2)$ und $p_2/(p_1 + p_2)$ annehmen kann, so ist $\mathcal{H}(\mathbf{X}) = \mathcal{H}(\mathbf{Y}) + (p_1 + p_2)\mathcal{H}(\mathbf{Z})$.
6. Lemma von Gibbs: Sei \mathbf{Y} eine Zufallsvariable, die n mögliche Werte mit den Wahrscheinlichkeiten q_1, q_2, \dots, q_n annehmen kann (d.h., es gilt $0 \leq q_i \leq 1$ und $\sum_{i=1}^n q_i = 1$). Dann gilt:

$$\mathcal{H}(\mathbf{X}) \leq - \sum_{i=1}^n p_i \log q_i,$$

- wobei Gleichheit genau dann gilt, wenn $(p_1, p_2, \dots, p_n) = (q_1, q_2, \dots, q_n)$.
7. Subadditivität: Sei $\mathbf{Z} = (\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_n)$ eine Zufallsvariable, deren mögliche Werte n -Tupel der Form (x_1, x_2, \dots, x_n) sind, d.h., x_i ist der Wert der Zufallsvariablen \mathbf{X}_i . Dann gilt:

$$\mathcal{H}(\mathbf{Z}) \leq \mathcal{H}(\mathbf{X}_1) + \mathcal{H}(\mathbf{X}_2) + \dots + \mathcal{H}(\mathbf{X}_n),$$

wobei Gleichheit genau dann gilt, wenn die Zufallsvariablen \mathbf{X}_i unabhängig sind, d.h.,

$$\Pr(\mathbf{X}_1 = x_1, \mathbf{X}_2 = x_2, \dots, \mathbf{X}_n = x_n) = \prod_{i=1}^n \Pr(\mathbf{X}_i = x_i).$$

Beweis. Wir beschränken uns, wie erwähnt, auf den Beweis der ersten Aussage und überlassen die übrigen Aussagen dem Leser, siehe Übung 4.12.

Es ist leicht zu sehen, dass die Funktion $\log x$ auf dem Intervall $(0, \infty)$ streng konkav ist. Mit der Jensenschen Ungleichung (Lemma 4.30) folgt:

$$\begin{aligned} \mathcal{H}(\mathbf{X}) &= - \sum_{i=1}^n p_i \log p_i = \sum_{i=1}^n p_i \log \frac{1}{p_i} \\ &\leq \log \sum_{i=1}^n \left(p_i \frac{1}{p_i} \right) = \log n. \end{aligned}$$

Dabei gilt Gleichheit genau dann, wenn $p_i = 1/n$ für jedes i mit $1 \leq i \leq n$ ist. \square

Unter Verwendung der bedingten Wahrscheinlichkeit aus Definition 2.54 können wir nun den Begriff der bedingten Entropie definieren. Intuitiv misst die bedingte Entropie den Betrag der Information über eine Zufallsvariable \mathbf{X} , die durch Kenntnis des Wertes, den eine Zufallsvariable \mathbf{Y} annimmt, enthüllt wird. Dieser Begriff wird

nützlich sein, um die Information zu quantifizieren, die der Kryptoanalytiker Erich über den verwendeten Schlüssel aus der Kenntnis eines bestimmten Schlüsseltextes erhalten kann.

Definition 4.32 (Bedingte Entropie). Seien \mathbf{X} und \mathbf{Y} zwei Zufallsvariablen, so dass \mathbf{X} n mögliche Werte aus der Menge $X = \{x_1, x_2, \dots, x_n\}$ und \mathbf{Y} m mögliche Werte aus der Menge $Y = \{y_1, y_2, \dots, y_m\}$ annehmen kann.

Definiere für jedes i und j mit $1 \leq i \leq n$ und $1 \leq j \leq m$ die bedingten Wahrscheinlichkeiten p_{ij} und die Wahrscheinlichkeiten q_j durch:

$$\begin{aligned} p_{ij} &= \Pr(\mathbf{X} = x_i \mid \mathbf{Y} = y_j); \\ q_j &= \Pr(\mathbf{Y} = y_j). \end{aligned}$$

Für festes j , $1 \leq j \leq m$, bezeichne \mathbf{X}_j die Zufallsvariable auf X , die gemäß der Wahrscheinlichkeiten $p_{1j}, p_{2j}, \dots, p_{nj}$ verteilt ist. Es gilt:

$$\mathcal{H}(\mathbf{X}_j) = - \sum_{i=1}^n p_{ij} \log p_{ij}.$$

Definiere die bedingte Entropie als

$$\mathcal{H}(\mathbf{X} \mid \mathbf{Y}) = \sum_{j=1}^m q_j \mathcal{H}(\mathbf{X}_j) = - \sum_{j=1}^m \sum_{i=1}^n q_j p_{ij} \log p_{ij}.$$

Intuitiv ist die bedingte Entropie $\mathcal{H}(\mathbf{X} \mid \mathbf{Y})$ das gewichtete Mittel (bezüglich der Wahrscheinlichkeitsverteilung auf Y) der Entropien $\mathcal{H}(\mathbf{X}_j)$ für alle j mit $1 \leq j \leq m$. Dieser Begriff wird in Beispiel 4.36 unten auf Kryptosysteme angewandt.

Der Beweis der folgenden beiden Aussagen ist simpel und wird deshalb wegge lassen; siehe Übung 4.13.

Satz 4.33. $\mathcal{H}(\mathbf{X}, \mathbf{Y}) = \mathcal{H}(\mathbf{Y}) + \mathcal{H}(\mathbf{X} \mid \mathbf{Y})$.

Korollar 4.34. $\mathcal{H}(\mathbf{X}, \mathbf{Y}) \leq \mathcal{H}(\mathbf{X})$, wobei Gleichheit genau dann gilt, wenn \mathbf{X} und \mathbf{Y} unabhängig sind.

Schlüsselmehrdeutigkeit

Die Eigenschaften der Entropie und der bedingten Entropie werden nun auf Kryptosysteme angewandt. In Beispiel 4.28 wurde festgestellt, dass man sich Schlüssel, Klartext und Schlüsseltext in einem gegebenen Kryptosystem $\mathbf{S} = (M, C, K, \mathcal{E}, \mathcal{D})$ jeweils als Zufallsvariablen \mathbf{K} , \mathbf{M} und \mathbf{C} vorstellen kann. Demnach kann man auch ihre Entropien berechnen.

Wir interessieren uns nun für die bedingte Entropie $\mathcal{H}(\mathbf{K} \mid \mathbf{C})$, welche als *Schlüsselmehrdeutigkeit von \mathbf{S}* bezeichnet und als der Betrag an Information interpretiert wird, die der abgehörte Schlüsseltext über den verwendeten Schlüssel enthüllt. Das folgende Resultat erklärt, wie man die Schlüsselmehrdeutigkeit eines gegebenen Kryptosystems berechnen kann.

Satz 4.35. Sei $\mathbf{S} = (M, C, K, \mathcal{E}, \mathcal{D})$ ein Kryptosystem, und seien \mathbf{K} , \mathbf{M} und \mathbf{C} die K , M und C entsprechenden Zufallsvariablen. Dann ist

$$\mathcal{H}(\mathbf{K} | \mathbf{C}) = \mathcal{H}(\mathbf{K}) + \mathcal{H}(\mathbf{M}) - \mathcal{H}(\mathbf{C}).$$

Beweis. Wendet man Satz 4.33 mit $\mathbf{X} = \mathbf{C}$ und $\mathbf{Y} = (\mathbf{K}, \mathbf{M})$ an, so erhält man

$$\mathcal{H}(\mathbf{C}, \mathbf{K}, \mathbf{M}) = \mathcal{H}(\mathbf{K}, \mathbf{M}) + \mathcal{H}(\mathbf{C} | \mathbf{K}, \mathbf{M}).$$

Da \mathbf{S} ein Kryptosystem ist, bestimmen ein gegebener Schlüssel $k \in K$ und ein gegebener Klartext $m \in M$ eindeutig den Schlüsseltext $c = E_k(m)$. Folglich ist $\mathcal{H}(\mathbf{C} | \mathbf{K}, \mathbf{M}) = 0$. Somit ergibt sich

$$\mathcal{H}(\mathbf{C}, \mathbf{K}, \mathbf{M}) = \mathcal{H}(\mathbf{K}, \mathbf{M}).$$

Da jedoch die Zufallsvariablen \mathbf{K} und \mathbf{M} unabhängig sind, impliziert die Subadditivität der Entropie (Eigenschaft (7) in Satz 4.31): $\mathcal{H}(\mathbf{K}, \mathbf{M}) = \mathcal{H}(\mathbf{K}) + \mathcal{H}(\mathbf{M})$. Also ist

$$\mathcal{H}(\mathbf{C}, \mathbf{K}, \mathbf{M}) = \mathcal{H}(\mathbf{K}, \mathbf{M}) = \mathcal{H}(\mathbf{K}) + \mathcal{H}(\mathbf{M}). \quad (4.14)$$

Ähnlich bestimmen ein gegebener Schlüssel $k \in K$ und ein gegebener Schlüsseltext $c \in C$ eindeutig den Klartext $m = D_k(c)$. Somit ist $\mathcal{H}(\mathbf{M} | \mathbf{K}, \mathbf{C}) = 0$. Es folgt

$$\mathcal{H}(\mathbf{C}, \mathbf{K}, \mathbf{M}) = \mathcal{H}(\mathbf{K}, \mathbf{C}). \quad (4.15)$$

Dann implizieren Satz 4.33 und die Gleichheiten (4.14) und (4.15):

$$\begin{aligned} \mathcal{H}(\mathbf{K} | \mathbf{C}) &= \mathcal{H}(\mathbf{K}, \mathbf{C}) - \mathcal{H}(\mathbf{C}) \\ &= \mathcal{H}(\mathbf{C}, \mathbf{K}, \mathbf{M}) - \mathcal{H}(\mathbf{C}) \\ &= \mathcal{H}(\mathbf{K}) + \mathcal{H}(\mathbf{M}) - \mathcal{H}(\mathbf{C}), \end{aligned}$$

womit der Satz bewiesen ist. \square

Um die Aussage von Satz 4.35 zu illustrieren, berechnen wir die Schlüsselmehrdeutigkeit des in Beispiel 4.23 definierten Kryptosystems.

Beispiel 4.36 (Schlüsselmehrdeutigkeit: Fortsetzung der Beispiele 4.23 und 4.28). Betrachte das Kryptosystem aus Beispiel 4.23. In Beispiel 4.28 wurden die Entropien der Zufallsvariablen \mathbf{K} , \mathbf{M} und \mathbf{C} , welche dem Schlüssel-, Klartext- und Schlüsselraum entsprechen, wie folgt berechnet: $\mathcal{H}(\mathbf{K}) \approx 0.8113$, $\mathcal{H}(\mathbf{M}) \approx 0.9183$ und $\mathcal{H}(\mathbf{C}) \approx 0.9799$. Aus Satz 4.35 ergibt sich:

$$\begin{aligned} \mathcal{H}(\mathbf{K} | \mathbf{C}) &= \mathcal{H}(\mathbf{K}) + \mathcal{H}(\mathbf{M}) - \mathcal{H}(\mathbf{C}) \\ &\approx 0.8113 + 0.9183 - 0.9799 \\ &= 0.7497. \end{aligned} \quad (4.16)$$

Die Schlüsselmehrdeutigkeit dieses Kryptosystems kann ebenso direkt bestimmt werden, indem man die bedingte Entropie gemäß Definition 4.32 berechnet. Nach dem Satz von Bayes (Satz 2.55) sind nun für jedes $k \in K = \{\$\,#\}$ und $c \in C = \{x, y\}$ die bedingten Wahrscheinlichkeiten $\Pr(\mathbf{K} = k | \mathbf{C} = c)$ zu bestimmen:

$$\begin{aligned}\Pr(\$|x) &= \frac{\Pr(b,\$)}{\Pr(x)} = \frac{2}{5}; & \Pr(\#|x) &= \frac{\Pr(a,\#)}{\Pr(x)} = \frac{3}{5}; \\ \Pr(\$|y) &= \frac{\Pr(a,\$)}{\Pr(y)} = \frac{1}{7}; & \Pr(\#|y) &= \frac{\Pr(b,\#)}{\Pr(y)} = \frac{6}{7}.\end{aligned}$$

Dass dabei in diesem Beispiel die folgenden Gleichheiten gelten, ist reiner Zufall:

- $\Pr(\$|x) = 2/5 = \Pr(b|x)$,
- $\Pr(\#|x) = 3/5 = \Pr(a|x)$,
- $\Pr(\$|y) = 1/7 = \Pr(a|y)$ und
- $\Pr(\#|y) = 6/7 = \Pr(b|y)$.

Im Allgemeinen ist dies nicht der Fall. Beispielsweise muss nicht notwendigerweise gelten, dass der Schlüsselraum und der Klartextraum von gleicher Größe sind.

Dann berechnen wir die Schlüsselmehrdeutigkeit gemäß Definition 4.32:

$$\mathcal{H}(\mathbf{K} | \mathbf{C}) = \frac{5}{12} \left(\frac{2}{5} \log \frac{5}{2} + \frac{3}{5} \log \frac{5}{3} \right) + \frac{7}{12} \left(\frac{1}{7} \log 7 + \frac{6}{7} \log \frac{7}{6} \right) \approx 0.7497,$$

womit das Ergebnis in (4.16) bestätigt wird, das sich aus Satz 4.35 ergab.

4.4 Übungen und Probleme

Aufgabe 4.1 (a) Entschlüssle den folgenden Schlüsseltext, der mit der Permutationschiffre erstellt wurde:

O□HWEARCD□ETE□HT AR HCTECA “T□R□BEHDE I R I N□”KI “□□BL LL” L I □U□?□TAMRMUH□ONAQU□RT I NETA□NNTARO?NI

Hinweis: Bestimme zunächst den verwendeten Schlüssel π_1 . Die Antwort auf die verschlüsselte englische Nachricht wird mit der Permutationschiffre als „HTOB“ verschlüsselt, wobei ein *anderer* Schlüssel verwendet wird.

(b) Sei π_1 der Schlüssel aus Übung 4.1(a). Wähle zwei weitere Schlüssel:

$$\pi_2 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 2 & 1 & 8 & 6 & 4 & 5 & 3 & 7 \end{pmatrix} \quad \text{und} \quad \pi_3 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 8 & 11 & 2 & 1 & 10 & 5 & 3 & 7 & 4 & 6 & 9 \end{pmatrix}.$$

Verschlüssele den Klartext aus Übung 4.1(a) mit Tripel-Verschlüsselung unter Verwendung der Schlüssel π_1 , π_2 und π_3 ; siehe Abschnitt 4.2.3. Wie groß ist der Schlüsselraum?

(c) Der Klartext „SCHAEUBLE“ soll in den Schlüsseltext „BELAUSCHE“ überführt werden. Bestimme den entsprechenden Ver- und Entschlüsselungsschlüssel der Permutationschiffre.

Tabelle 4.17. Ein mit der affinen Chiffre erzeugter Schlüsseltext für Übung 4.2(a)

PWMU DZUIFDMZ WERXMNR XSZIF XUM REIFD
 XSZIF UFZMO ZSCVBO HMZMRQDMR OIFEIFD
 BNUMOOD WMUOOMO CGRXNUIFD
 ODUNN SRX FMUDMZ
 ESB UFZMR
 WENXWMQ
 SO
 W

Aufgabe 4.2 Tabelle 4.17 zeigt einen mit der affinen Chiffre erzeugten Schlüsseltext.

- (a) Bestimme den Schlüssel und den Klartext mittels Häufigkeitsanalyse.

Hinweis: Der Klartext ist ein deutsches Gedicht von Christian Morgenstern, das aus „Ausgewählte Werke I“, Gustav Kiepenheuer Verlag, 1985, stammt. Die Häufigkeiten der Buchstaben in typischen deutschen Texten findet man zum Beispiel in [Beu02].

- (b) Der Titel des Gedichts in Tabelle 4.17 wurde mit der affinen Chiffre (unter Verwendung eines anderen Schlüssels) in den folgenden Schlüsseltext übertragen:

JMU DHMYBDUH

Bestimme den Schlüssel und entschlüssele den Schlüsseltext gemäß dem Known-Plaintext-Angriff in Beispiel 4.8. Die ersten beiden Buchstaben des Klartexts sind „DI“.

Aufgabe 4.3 Tabelle 4.18 zeigt zwei abgefangene Schlüsseltexte, c_1 und c_2 . Man weiß, dass beide denselben Klartext verschlüsseln und dass für einen die Verschiebungschiffre verwendet wurde, für den anderen die Vigenère-Chiffre. Bestimme die jeweils verwendeten Schlüssel und entschlüssele die Schlüsseltexte.

Tabelle 4.18. Zwei aus dem gleichen Klartext entstandene Schlüsseltexte für Übung 4.3

c_1	W K L V V H Q W H Q F H L V H Q F U B S W H G E B F D H V D U V N H B
c_2	N U O S J Y A Z E E W R O S V H P X Y G N R J B P W N K S R L F Q E P

Hinweis: Nach dem Entschlüsseln der Texte wird man sehen, dass einer eine wahre, der andere eine falsche Aussage macht. Ist die Methode der Häufigkeitsanalyse möglicherweise sinnvoll?

Aufgabe 4.4 Betrachte die in Beispiel 4.13 angegebene Verschlüsselung des Klartextes „THE FOOL ON THE HILL“, siehe Tabelle 4.8.

- (a) Beweise, dass $A^{-1} = \begin{pmatrix} 21 & 16 \\ 28 & 17 \end{pmatrix}$ tatsächlich die inverse Matrix von $A = \begin{pmatrix} 3 & 4 \\ 7 & 2 \end{pmatrix}$ modulo 29 ist, d.h., zeige $AA^{-1} = A^{-1}A = U_2 \bmod 29$.

- (b) Überprüfe, dass der in Tabelle 4.8 angegebene Schlüsseltext mit dem Entschlüsselungsschlüssel A^{-1} wirklich in diesen Klartext entschlüsselt wird.
- (c) Verschlüssele den gesamten Text des Beatles-Songs „THE FOOL ON THE HILL“ mit Blocklänge 3 und einer Matrix deiner Wahl. Bestimme die inverse Matrix und überprüfe, dass sich dein Schlüsseltext wirklich wieder in den richtigen Klartext entschlüsseln lässt.
- (d) Tabelle 4.19 zeigt einen Schlüsseltext, der mittels der Hill-Chiffre mit Blocklänge zwei und unter Verwendung des Alphabets $\Sigma = \{A, B, C, \dots, Z, \square, \&, -\}$ erstellt wurde. Der entsprechende Klartext beginnt mit den folgenden vier Buchstaben: „THER“. Bestimme den Schlüssel und entschlüssele den Schlüsseltext gemäß dem Known-Plaintext-Angriff aus Beispiel 4.16.

Tabelle 4.19. Ein mittels der Hill-Chiffre erzeugter Schlüsseltext für Übung 4.4(d)

S BQB Q - BKP □ RA&GQ - A U S E T H R C - □ X R Q N Q B J W & I J K B H & T D M R && J Q M J K B K P □ R □ Q - U D Q U R W & T O C X P R R R D A K I Q R R R && E Q - S BQB Q - BKP □ RA&GQ - A U S E T H R C - □ X R Q N Q B J K Q N D U R W - O S A B & R Y X K B G B K P □ R □ Q - A K R G □ I P N S B A - R C Q - I E X R K A O T R X E P □ Z R C - & R A A W Q E R S U B - O D J R S O & P G X P R H K Z O - & V S A & V R T C X A O Q R R H K Z C B K E P H R M K F J C M □ B & R Y X K B G S B A - R P A Y K Z A Y X Z R C - □ K R R S S F R X S A S B Q B Q - B K P □ R A & G Q - A U S E T H R C - □ X R J D W C B - X Z R G □ Z X R E Q R X O - A H Q R R S U B J D W C A W Q - U K S A B H & T I E K R O & D C R I & B K R R T & T C O & V R I & B R A A K S A A P U X K O R W & T E E J K A U C B D J R H U B T I M □ A U Y Z Z Q & T O G K R S P - L Q M A Z S E I Y C B X Z O & S B Q B Q - BKP □ RA&GQ - A U S E T H R C - □ X R J D W C I E X R Q N Q S R W A B Q - J D W E R A P G S B Q E R H U Z R C - □ X R J D W C M G X D Q -
--

Hinweis: Entnommen wurde der Klartext dem Album „We're only in it for the Money“ von Frank Zappa and the Mothers of Invention, Frank Zappa Music Co., Inc., 1967.

Aufgabe 4.5 Tabelle 4.20 zeigt einen Schlüsseltext, der mit der Vigenère-Chiffre erzeugt wurde, wobei die Leerstellen zwischen Wörtern und die Interpunktionszeichen weggelassen wurden. Wende Kasiskis Methode aus Beispiel 4.15 an, um die Periode, den verwendeten Schlüssel und die ursprüngliche Nachricht zu ermitteln.

Hinweis: Der englische Klartext stammt aus der Kurzgeschichte „A Twenties Memory“ von Woody Allen, Random House, Inc., 1971.

Aufgabe 4.6 Es gibt eine Reihe von anderen Quadraten, die wie das Vigenère-Quadrat für eine polyalphabetische Blockchiffre ähnlich der Vigenère-Chiffre aus Beispiel 4.10 verwendet werden können. Ein berühmtes solches Kryptosystem ist die Beaufort-Chiffre, die von Admiral Sir Francis Beaufort (1774 bis 1857) vorgeschlagen wurde (welcher außerdem die zwölfstufige Beaufort-Skala zum Messen der Windgeschwindigkeit erfand).

Tabelle 4.20. Ein mittels der Vigenère-Chiffre erzeugter Schlüsseltext für Übung 4.5

P	O	U	L	Q	K	Z	H	C	T	M	Z	O	J	G	U	A	Z	V	Y	F	P	V	Y	B	O	P	A	L	O
W	Z	O	J	G	L	V	Z	Q	T	N	T	Z	W	C	S	E	G	O	U	K	Z	A	J	V	X	M	V	Q	U
W	D	D	B	R	M	V	Q	Z	D	E	L	D	I	B	C	G	U	J	S	M	T	I	N	W	K	O	J	G	E
S	S	H	M	F	G	J	D	F	D	W	X	K	I	Y	L	V	W	Q	B	U	G	P	W	G	U	X	Q	Z	
Z	N	R	E	C	C	A	S	W	T	N	Y	L	I	A	H	W	E	W	N	T	Z	M	W	Z	I	X	N	G	U
K	K	Y	C	M	X	T	N	O	J	T	F	L	W	A	C	G	N	K	F	D	Y	E	Z	K	R	O	S	A	Q
A	L	Q	V	J	U	N	K	G	U	K	A	V	J	N	W	T	G	M	I	X	B	J	G	O	A	K	H	N	A
Z	N	S	A	R	G	J	E	Z	V	B	B	A	L	Z	B	X	F	P	R	V	N	I	L	A	A	M	A	G	U
K	S	K	H	P	Y	I	A	J	E	C	M	B	P	P	V	G	M	A	T	R	O	V	A	C	T	N	E	T	G
U	X	T	M	N	L	P	E	P	U	S	A	V	F	M	A	R	R	Y	Y	P	M	F	Z	A	I	L	H	R	P
Q	H	Y	A	D	R	V	K	A	I	A	Z	E	Q	S	L	L	G	V	G	O	J	G	G	U	X	T	M	Y	U
Y	A	J	Z	L	A	X	F	G	J	D	G	Y	I	Y	M	Z	H	A	M	U	G	Q	I	F	Q	E	N	E	
O	I	D	Q	S	Y	D	E	R	O	S	G	O	U	Z	E	S	F	V	Y	X	L	O	K	Y	O	K	L	E	I
Z	V	Z	A	R	G	U	L	Q	Z	G	E	L	I	A	H	P	X	G	P	X	U	P	R	P	G	I	I	L	C
A	W	C	Y	I	M	T	Y	Y	K	E	L	J	L	M	V	G	K	Z	W	G	A	L	E	P	R	Y	W	I	B
U	S	U	B	B	U	G	T	F	H	X	F	W	Z	K	W	N	W	L	W										

Die Beaufort-Chiffre beruht auf dem in Tabelle 4.21 dargestellten Beaufort-Quadrat. Es sieht genau wie das Vigenère-Quadrat in Tabelle 4.6 aus, nur dass man hier mit dem Buchstaben „Z“ in der ersten Zeile beginnt und dann rückwärts durch das englische Alphabet geht. Von Zeile zu Zeile wird diese Reihenfolge der Buchstaben um eine Position nach rechts verschoben. Die Ver- und Entschlüsselung funktioniert genau wie bei der Vigenère-Chiffre, nur dass man statt des Vigenère-Quadrats nun das Beaufort-Quadrat verwendet.

- (a) Angenommen, ein Klartext wurde mittels der Vigenère-Chiffre mit einem bestimmten Schlüssel verschlüsselt. Wie würde man diesen Schlüssel modifizieren, um mittels der Beaufort-Chiffre denselben Schlüsseltext aus diesem Klartext zu erhalten?
- (b) Verifiziere deine Vermutung aus Übung 4.6(a). Vergleiche zu diesem Zweck die beiden Schlüsseltexte, die aus demselben Klartext erhalten wurden, mittels der Vigenère-Chiffre für einen Schlüssel und mittels der Beaufort-Chiffre mit dem entsprechend modifizierten Schlüssel. Verwende dazu den Klartext aus Übung 4.5, falls es dir gelungen war, den Schlüsseltext in Tabelle 4.20 zu entschlüsseln; andernfalls wähle selbst einen beliebigen Klartext.

Aufgabe 4.7 (a) Wende den CBC-Modus auf die folgenden Blockchiffren an:

- Die Vigenère-Chiffre mit Schlüssel ELLA und dem Klartext in Tabelle 4.7 aus Beispiel 4.10. Der Anfangsvektor aus \mathbb{Z}_{26}^4 ist $\mathbf{c}_0 = \text{ALLE}$.
- Die Vigenère-Chiffre mit Schlüssel PAULA und dem Klartext in Tabelle 4.12. Der Anfangsvektor aus \mathbb{Z}_{26}^5 ist $\mathbf{c}_0 = \text{ALUAP}$.
- Die Hill-Chiffre mit Schlüssel $A = \begin{pmatrix} 3 & 4 \\ 7 & 2 \end{pmatrix}$ und dem Klartext in Tabelle 4.8 aus Beispiel 4.13. Der Anfangsvektor ist $\mathbf{c}_0 = \square B$.

Tabelle 4.21. Beaufort-Quadrat: Klartext „S“ wird mit Schlüssel „H“ als „A“ verschlüsselt

Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A
A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B
B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C
C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D
D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E
E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F
F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G
G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H
H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I
I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J
J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K
K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L
L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M
M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N
N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O
O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P
P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q
Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R
R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S
S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T
T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U
U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V
V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W
W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X
X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y
Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z

- Die Hill-Chiffre mit Schlüssel $A = \begin{pmatrix} 4 & 3 \\ 2 & 7 \end{pmatrix}$ und dem Klartext, der dem Schlüsseltext in Tabelle 4.19 aus Übung 4.4(d) entspricht. Der Anfangsvektor ist $\mathbf{c}_0 = \text{FZ}$.

Hinweis: Für die Vigenère-Verschlüsselungen wird die im CBC-Modus verwendete Operation \oplus durch zeichenweise Addition modulo 26 ersetzt. Ebenso wird bei Hill-Verschlüsselungen \oplus durch zeichenweise Addition modulo 29 ersetzt.

- (b) Wende sowohl den CFB-Modus als auch den OFB-Modus auf die Permutationschiffre mit Alphabet $\Sigma = \{0, 1\}$, Blocklänge 6, Schlüssel $\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 1 & 2 & 5 & 3 & 4 \end{pmatrix}$ und Anfangsvektor $\mathbf{z}_0 = 101001$ an. Der Klartext ist

$$m = 100110 \ 111100 \ 001100 \ 010100.$$

Wähle $k = 3$ als die kürzere Blocklänge.

- (c) Wiederhole Übung 4.7(b) mit der kürzeren Blocklänge $k = 2$.
 (d) Verschlüssele mittels der Permutationschiffre mit Alphabet $\Sigma = \{0, 1\}$, Blocklänge 3 und Schlüssel $\pi = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}$ den Klartext

$$m = 001\ 100\ 111\ 010$$

im ECB-, CBC-, CFB- und OFB-Modus. Wird dabei ein Anfangsvektor benötigt, so sei dieser 111. Im CFB- und OFB-Modus sei die kürzere Blocklänge $k = 2$.

Aufgabe 4.8 Betrachte die auf einem linearen Rückkopplungsschieberegister beruhende Stromchiffre aus Beispiel 4.21.

- (a) Sei $n = 6$. Fixiere die Koeffizienten $a_2 = a_3 = a_6 = 0$ und $a_1 = a_4 = a_5 = 1$ in (4.7). Wähle den Schlüssel $\mathbf{k} = (1, 1, 0, 0, 1, 1)$. Verschlüssele die Nachricht

$$m = 101110001101101010011010111001.$$

- (b) Führe einen Known-Plaintext-Angriff aus, um diese Stromchiffre zu brechen.

Hinweis: Dieser Angriff ähnelt der Kryptoanalyse der affin-linearen Blockchiffren, z.B. der Hill-Chiffre aus Beispiel 4.16. Alle Operationen in dieser Stromchiffre sind linear. Kennt man also einen Klartext und einen zugehörigen Schlüsseltext, so kann man ein System linearer Gleichungen lösen, um die Werte der n unbekannten Koeffizienten in der linearen Rekursion (4.7) zu bestimmen.

Aufgabe 4.9 Betrachte die folgende Stromchiffre, die eine der Ideen der berüchtigten Verschlüsselungsmaschine *Enigma* umsetzt, welche die Deutsche Wehrmacht während des Zweiten Weltkriegs verwendete. Der Schlüsselraum ist \mathbb{Z}_{26} . Für einen festen Schlüssel $k \in \mathbb{Z}_{26}$ und für jedes $i \geq 1$ wird der Schlüsselstrom s erzeugt, indem sein i -tes Element bestimmt wird durch

$$s_i = (k + i - 1) \bmod 26.$$

Sei π eine feste Permutation von \mathbb{Z}_{26} . Ist $s \in \mathbb{Z}_{26}$ das aktuelle Element des Schlüsselstroms und ist x der aktuelle Klartextbuchstabe, so verwendet die Verschlüsselungsfunktion E_s , die von \mathbb{Z}_{26} in \mathbb{Z}_{26} abbildet, sowohl π als auch s wie folgt:

$$E_s(x) = \pi((x + s) \bmod 26).$$

Ebenso verwendet die Entschlüsselungsfunktion D_s , die auch von \mathbb{Z}_{26} in \mathbb{Z}_{26} abbildet, sowohl s als auch die inverse Permutation π^{-1} zur Entschlüsselung des aktuellen Schlüsseltextsymbols y :

$$D_s(y) = (\pi^{-1}(y) - s) \bmod 26.$$

- (a) Beweise, dass dies ein Kryptosystem ist, d.h., zeige, dass (4.1) erfüllt ist.
(b) Angenommen, die Permutation π von \mathbb{Z}_{26} ist durch

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 & 25 \\ 11 & 8 & 6 & 1 & 3 & 4 & 5 & 9 & 10 & 2 & 7 & 0 & 14 & 12 & 20 & 13 & 25 & 21 & 15 & 17 & 24 & 18 & 16 & 22 & 19 & 23 \end{pmatrix}$$

gegeben. Der folgende Schlüsseltext wurde mittels der obigen Stromchiffre mit π erzeugt:

FRRMXCBEWMJWDDH TKO UACYKUK QAMT ASVZWO

Finde den verwendeten Schlüssel durch erschöpfende Suche im Schlüsselraum, bestimme den vollständigen Schlüsselstrom und entschlüssele den Schlüsseltext.

Aufgabe 4.10 Ändere die in Beispiel 4.23 gegebenen Wahrscheinlichkeiten so, dass sich ein Kryptosystem ergibt, das perfekte Geheimhaltung garantiert.

Aufgabe 4.11 Beweise, dass die Verschiebungschiffre perfekte Geheimhaltung garantiert, falls die Schlüssel gleichverteilt sind und falls ein neuer zufälliger Schlüssel jeweils für jeden neuen Klartextbuchstaben verwendet wird.

Aufgabe 4.12 Vervollständige den Beweis von Satz 4.31.

Aufgabe 4.13 Beweise Satz 4.33 und Korollar 4.34.

Aufgabe 4.14 Bestimme in der Arithmetik modulo 2 die Inverse der Matrix

$$A = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}.$$

Aufgabe 4.15 Bestimme die Determinante $\det B$ der Matrix

$$B = \begin{pmatrix} 3 & 1 & 2 \\ 2 & 3 & 1 \\ 1 & 2 & 3 \end{pmatrix}.$$

Problem 4.1 (Berechnen der Determinante)

- (a) Entwirf einen Algorithmus in Pseudocode zur Berechnung der Determinante einer gegebenen Matrix über \mathbb{Z}_m . Implementiere deinen Algorithmus in einer Programmiersprache deiner Wahl.
- (b) Kann man die Inverse einer gegebenen Matrix effizient berechnen?

Problem 4.2 (Redundanz einer Sprache und Unizitätsabstand)

Sei L eine natürliche Sprache, wie etwa Holländisch oder Englisch, mit dem Alphabet Σ . Intuitiv misst die *Entropie von L* , bezeichnet mit \mathcal{H}_L , die Entropie pro Buchstabe von L . Wäre L eine zufällige Sprache, in der jeder Buchstabe mit derselben Wahrscheinlichkeit und unabhängig von seinem Kontext auftritt, dann hätte L die Entropie $\log \|\Sigma\|$. Natürliche Sprachen sind jedoch nicht zufällig. Den Anteil der „zusätzlichen“ oder „überflüssigen“ Buchstaben nennt man die *Redundanz von L* und bezeichnet diese mit \mathcal{R}_L . Definiere diese Begriffe formal durch

$$\mathcal{H}_L = \lim_{n \rightarrow \infty} \frac{\mathcal{H}(\mathbf{M}^n)}{n} \quad \text{und} \quad \mathcal{R}_L = 1 - \frac{\mathcal{H}_L}{\log \|\Sigma\|},$$

wobei \mathbf{M}^n eine Zufallsvariable ist, die gemäß der Wahrscheinlichkeiten verteilt ist, mit der die Wörter der Länge n (also alle n -Gramme) in der Sprache L auftreten.

- (a) Sei $\mathbf{S} = (M, C, K, \mathcal{E}, \mathcal{D})$ ein Kryptosystem, wobei $\|M\| = \|C\|$ gilt und die Schlüssel gleichverteilt sind. Sei L die Sprache, in der Klartexte geschrieben werden. Für einen gegebenen Schlüsseltext der Länge n kann ein Kryptoanalytiker, der einen Ciphertext-only-Angriff ausführt, womöglich gewisse Schlüssel ausschließen, aber viele mögliche Schlüssel bleiben übrig, von denen nur einer der korrekte ist. Mögliche, aber inkorrekte Schlüssel heißen *falsche* Schlüssel. Sei \tilde{s}_n die erwartete Anzahl falscher Schlüssel für einen gegebenen Schlüsseltext der Länge n .

Finde eine untere Schranke für \tilde{s}_n in Abhängigkeit von der Redundanz von L .

Hinweis: Verwende die Begriffe der Entropie und der bedingten Entropie.

- (b) Definiere den *Unizitätsabstand* von \mathbf{S} als den Wert $n_0 \in \mathbb{N}$, für den \tilde{s}_{n_0} null wird, d.h., als die mittlere Größe von Schlüsseltexten, die nötig ist, um den korrekten Schlüssel eindeutig zu bestimmen, vorausgesetzt, man hat hinreichende Berechnungszeit zur Verfügung. Gib eine Approximation des Unizitätsabstands n_0 an, die auf deiner Formel für \tilde{s}_n beruht.

Problem 4.3 (Ein Schlüsseltext von Edgar Allan Poe)

Unter Verwendung einer monoalphabetischen Chiffre erzeugte Edgar Allan Poe den folgenden englischen Schlüsseltext in seiner Kurzgeschichte „The Gold-Bug“ (Random House, Inc., 1965):

53†‡†305))6*;4826)4‡.)4‡);806*;48†8¶60))85;1‡(;:‡*8
 †83(88)5*†;46(;88*96*?;8)*‡(;485);5*†2:/*‡(;4956*2(5
 -4)8¶8;4069285);)6†8)4‡‡;1(‡9;48081;8:8‡1;48†85;
 4)485†528806*81(‡9;48;(88;4(‡?34;48)4‡;161;:188;‡?;

Bestimme den Schlüssel und entschlüssele den Schlüsseltext. Gehe mit deiner Lösung dieses Rätsels los und finde den versteckten Schatz und alle Juwelen von Golconda.

4.5 Zusammenfassung und bibliographische Notizen

Allgemeine Bemerkungen: Zu den Klassikern in der Kryptologie gehören die Bücher von Goldreich [Gol99, Gol01], Salomaa [Sal96], Stinson [Sti05] und Welsh [Wel98]. Das Buch von Schneier [Sch96] ist eine überaus umfassende Kollektion von nahezu sämtlichen Begriffen und Konzepten, die in der Kryptologie bekannt sind.

Singh [Sin99] hat ein sehr amüsantes, leicht lesbares, faszinierendes Buch über die Geschichte der Kryptographie geschrieben. Eine ältere, aber immer noch wertvolle Quelle ist Kahns Buch [Kah67]. Ohne hier eine vollständige Liste anzustreben, seien die Texte von Bauer [Bau00a, Bau00b], Beutelspacher et al. [Beu94, BSW01, Beu02], Buchmann [Buc01a, Buc01b] und Luby [Lub96] erwähnt. Mehr

über den Entwurf und die Analyse von Stromchiffren findet man in Rueppels Buch [Rue86]. Micciancio und Goldwasser [MG02] geben einen Überblick über die Komplexität von Gitterproblemen aus einer kryptographischen Perspektive. Bruß et al. [BEM⁺07] und Gisin et al. [GRTZ02] geben Einführungen in das Gebiet der Quantenkryptographie.

Zu den Überblicksartikeln, die in den Themenbereich dieses Buches fallen, gehören (in alphabethischer Reihenfolge) die von Beygelzimer et al. [BHHR99], Cai [Cai99], Feigenbaum [Fei92], Goldreich [Gol88], Goldwasser [Gol89], Kumar und Sivakumar [KS01], Nguyen und Stern [NS01], Rothe [Rot07b, Rot02] und Selman [Sel92].

Spezielle Bemerkungen: Die Anfänge der Kryptographie lassen sich bis in graue Vorzeiten und zu den Wurzeln der menschlichen Zivilisation zurückverfolgen. Beispielsweise berichtet Julius Cäsar in seinem Buch „De Bello Gallico“, dass er eine verschlüsselte Nachricht an Q. Tullius Cicero (den Bruder des berühmten Redners) übermittelte, der mit seiner Legion in den Gallischen Kriegen (58 bis 50 v.u.Z.) belagert wurde. Das verwendete System war monoalphabetisch und ersetzte lateinische Buchstaben durch griechische. Aus Cäsars Bericht geht jedoch nicht hervor, ob er tatsächlich die Verschiebungschiffre mit Schlüssel $k = 3$ verwendete. Diese Information wurde später von Suetonius geliefert.

Die Chiffre von Vigenère baut auf vorhergehenden Arbeiten des italienischen Mathematikers Leon Battista Alberti (1404 geboren), des deutschen Abtes Johannes Trithemius (1492 geboren) und des italienischen Wissenschaftlers Giovanni Porta (1535 geboren) auf, siehe Singh [Sin99]. Kasiskis Leistung, die Vigenère-Chiffre gebrochen zu haben, wird inzwischen nicht mehr nur ihm, sondern auch dem britischen Genie und Ekzentriker Charles Babbage zugeschrieben, der seine etwa um 1854 entstandene Arbeit allerdings nicht veröffentlichte. Mehr über Babbages Leben, Arbeit und seine genialen Erfindungen, einschließlich eines frühen Prototyps eines Computers, findet man in [Sin99].

Aus dem Beispiel in Tabelle 4.7 erfahren wir nicht nur, wie die Vigenère-Chiffre funktioniert, sondern auch, dass die Verwendung einer nicht sehr verbreiteten Sprache, wie etwa Ungarisch und Finnisch, die nicht autorisierte Entschlüsselung oft erschwert und somit zu einer erhöhten Sicherheit führt. Dies ist keine rein theoretische Beobachtung. Während des Zweiten Weltkriegs übermittelte die U.S. Navy wichtige Nachrichten mittels der Sprache der Navajos, eines Indianerstamms in Amerika. Der „Navajo-Code“ wurde nie von den japanischen Code-Brechern entschlüsselt, siehe [Sin99].

In Singhs Buch [Sin99] werden noch viele weitere spannende Begebenheiten aus der Geschichte der Kryptologie erzählt. Zum Beispiel wird dort ausführlich über den packenden Kampf zwischen den deutschen Kryptographen und den alliierten Kryptoanalytikern während des Zweiten Weltkriegs berichtet. Den Code-Brechern von Bletchley Park, unter welchen Alan Turing höchst erwähnenswerte Beiträge lieferte, gelang es schließlich, die deutsche *Enigma* zu brechen, siehe auch Bauer [Bau00a, Bau00b]. Die auf dem *Enigma*-Code beruhende Stromchiffre, die in

Übung 4.9 beschrieben wird, modifiziert eine ähnliche Stromchiffre aus dem Buch von Stinson [Sti05].

Dass „SCHAEUBLE“ ein Anagramm von „BELAUSCHE“ ist, wurde von Delbrouck [Del07] entdeckt (siehe Übung 4.1(c)).

Die Lösung von Problem 4.2, das die Entropie und die Redundanz natürlicher Sprachen sowie den Unizitätsabstand von Kryptosystemen behandelt, kann ebenfalls in Stinson [Sti05] gefunden werden. Stinson [Sti05] gibt eine Abschätzung von $\mathcal{H}_E \approx 1.25$ an, wobei E die englische Sprache bezeichnet. Das heißt, der mittlere Informationsgehalt englischer Texte – einschließlich des Buches, das hier übersetzt wird – ist etwa 1.25 Bit pro Buchstabe. Demnach hat Englisch eine Redundanz von $\mathcal{R}_E \approx 0.75$. Das bedeutet nicht, dass man dieses Buch auch dann noch lesen könnte, wenn auf je drei von vier Buchstaben verzichtet worden wäre.¹ Stattdessen bedeutet es, dass man diesen Text mit einem Codierungsalgorithmus auf etwa ein Viertel seiner ursprünglichen Länge komprimieren könnte, ohne dadurch irgendwelche Information zu verlieren.

Man könnte sich fragen, weshalb wir denn so redundant sprechen, wenn wir doch dieselbe Information in einer anderen, komprimierten, Sprache prägnanter – kürzer und bündiger – übermitteln könnten. Es ist erwähnenswert, dass die Redundanz natürlicher Sprachen nicht überflüssig ist, sie dient einem Zweck. Sie ermöglicht es uns nämlich, uns auch dann noch verständigen zu können, wenn Übertragungsfehler auftreten, sei es beim Gespräch in der Bar mit lautem, störendem Lärm im Hintergrund oder in der Vorlesung, wenn sich der Vorlesende unangebracht lautstark in den Vordergrund drängt. Da wir gerade davon sprechen, der Ausdruck „Es ist erwähnenswert, dass“ in einem früheren Satz dieses Absatzes hätte ohne größeren Informationsverlust einfach weggelassen werden können. Und ebenso die unnötige Einleitung „Da wir gerade davon sprechen“.

Die bahnbrechende Idee, den Begriff der Entropie als ein Maß der Information zu verwenden, ist Shannon [Sha49] zu verdanken, der auch Satz 4.24 bewies und so die perfekte Geheimhaltung von Kryptosystemen durch notwendige und hinreichende Bedingungen charakterisierte. Der Begriff der Entropie selbst ist allerdings älter als die Informationstheorie von Shannon. So spielt Entropie insbesondere in der Thermodynamik eine zentrale Rolle; beispielsweise wurde das Lemma von Gibbs [Gib75] ursprünglich in diesem Zusammenhang formuliert.

In den Kapiteln 7 und 8 werden wir uns wieder der Kryptographie zuwenden. Unser Hauptaugenmerk wird dann auf der Public-Key-Kryptographie und ihrer Beziehung zur Komplexitätstheorie liegen, weshalb die Präsentation modernerer symmetrischer Kryptosysteme in diesem Buch etwas stiefmütterlich behandelt wird, trotz ihrer fraglosen Bedeutung in der Praxis. Viele solcher Systeme beruhen auf überaus raffinierten Ideen. So besteht beispielsweise der *Data Encryption Standard* (kurz DES), eine der populärsten symmetrischen, monoalphabetischen Blockchiffren und in vielen Büchern zur Kryptographie zu finden, aus einer Reihe von ausgeklügelten Transformationen. Der DES wurde von IBM entwickelt und 1977 vom National Institute of Standards and Technology (NIST) als Standard empfohlen. Ein Problem

¹ H□□□s□□□h□□□h□□□□□□ w□□□ d□□ □i□□□ k□□□p□□□

dieses Systems ist seine Schlüssellänge, die nur 56 Bit beträgt und somit nach heutigen Maßstäben zu kurz ist. Der DES wurde 1999 durch erschöpfende Schlüsselsuche gebrochen. Mehr über die mathematischen Eigenschaften des DES, die vorwiegend aus empirischen Untersuchungen erhalten wurden, erfährt man aus der Arbeit von Kaliski, Rivest und Sherman [KRS88].

Ein Nachfolger dieser Blockchiffre, Triple-DES, hat eine Schlüssellänge von 112 Bit und ist somit sicherer. Für den *Advanced Encryption Standard* (kurz AES), welcher vom NIST als der aktuelle Verschlüsselungsstandard empfohlen wird, sei auf die Arbeit von Daemen und Rijmen [DR01] verwiesen.

Hierarchien über NP

*Wer reitet so spät durch Nacht und Schnee?
Es ist Gerd Wechsung mit seinem MEE;
Er hat die Formel wohl in dem Arm,
Er faßt sie sicher, er hält sie warm. –*

*Mein Phi, was birgst du so bang dein Gesicht? –
Siehst, Gerd, denn du den Erlkönig nicht?
Den Erlenkönig mit Kron und so? –
Mein Phi, das ist nur Theta-pe-zwo. –*

*,Du liebes Phi, komm, geh mit mir!
Gar schöne Spiele spiel ich mit dir;
Und bist du die kleinste Formel am End,
So macht meine Mutter dich äquivalent“. –*

*Mein Gerd, mein Gerd, und hörest du nicht,
Was Erlenkönig mir leise verspricht? –
Sei ruhig, bleibe ruhig, mein Phi!
Orakelmaschinen säuseln doch nie. –*

*,,Gehst, feines Phi, du mit mir schnell?
Meine Töchter erfragen dich parallel;
Meine Töchter sind schön und ganz ohne Makel
Und fragen nach dir ihr NP-Orakel“.*

*Mein Gerd, mein Gerd, und siehst du nicht dort
Erlkönigs Töchter am düstern Ort? –
Mein Phi, mein Phi, was ich da seh,
Das ist doch wieder bloß Theta-zwo-pe. –*

*,Ich liebe dich, mich reizt deine schöne Gestalt;
Und bist du nicht willig, so brauch ich Gewalt“. –
Mein Gerd, mein Gerd, jetzt faßt er mich an!
Erlkönig hat mir ein Leids getan! –*

*Gerd Wechsung grauset's, er reitet wie nie,
Er hält in Armen das ächzende Phi,
Erreicht den Hof mit Phi, so zart;
In seinen Armen das MEE war hart.*

*(Nach dem Gedicht „Erlkönig“
von Johann W. von Goethe, siehe auch [HW02, Rot04])*

In diesem Kapitel, das sich wieder der Komplexitätstheorie zuwendet, werden einige wichtige Hierarchien eingeführt, die auf NP basieren, wie etwa die boolesche Hierarchie über NP und die Polynomialzeit-Hierarchie. Es werden vollständige Probleme in den Stufen dieser Hierarchien identifiziert und die Beziehungen zwischen diesen Hierarchien sowie ihre Eigenschaften erkundet. Zu den Problemen, die vollständig für die Stufen der booleschen Hierarchie über NP sind, gehören zum Beispiel „exakte“ Varianten NP-vollständiger Optimierungsprobleme und kritische Graphprobleme. Als Beispiele für Probleme, die in den Stufen der Polynomialzeit-Hierarchie vollständig sind, werden Varianten von NP-vollständigen Problemen betrachtet, die sich durch eine feste Zahl alternierender längenbeschränkter Quantoren darstellen lassen. Im Zusammenhang damit wird der Begriff der alternierenden Turingmaschine eingeführt.

Außerdem wird in diesem Kapitel die Low-Hierarchie in NP eingeführt, die ein Maßstab für die Komplexität von NP-Problemen ist, die anscheinend weder in P noch

NP-vollständig sind. Als ein Beispiel solcher Probleme wird später, in Kapitel 6, gezeigt, dass das Graphisomorphie-Problem low ist.

5.1 Die boolesche Hierarchie über NP

In Abschnitt 3.5.3 wurde die NP-Vollständigkeit vieler Probleme gezeigt, darunter das Dreifärbbarkeitsproblem, 3-Colorability, und das Domatische-Zahl-Problem, DNP. Nun ist unser Ziel, die „exakten“ Varianten dieser beiden Probleme zu untersuchen und ihre exakte Komplexität zu bestimmen.

Wir beginnen mit dem exakten Domatische-Zahl-Problem. In Abschnitt 3.5.3 wurde DNP definiert als die Entscheidungsversion des Problems, die Knotenmenge eines gegebenen Graphen G in eine maximale Anzahl disjunkter dominierender Mengen zu zerlegen. Eine dominierende Menge ist eine Teilmenge D der Knotenmenge von G , so dass jeder Knoten in $V(G)$ entweder zu D gehört oder zu einem Knoten in D benachbart ist.

Was sind nun die „exakten“ Varianten von NP-vollständigen Optimierungsproblemen wie etwa DNP, und weshalb möchte man sie studieren? Das Domatische-Zahl-Problem ist zum Beispiel durch die Aufgabe motiviert, die Ressourcen in einem Computer-Netzwerk zu verteilen. Eine andere Aufgabe besteht darin, Sendeanlagen in einem Kommunikationsnetzwerk aufzustellen. Im letzteren Szenario sind n Städte durch Kommunikationskanäle miteinander verbunden. Eine *Sendergruppe* ist eine Teilmenge T der Städte, so dass jede Stadt aus T Nachrichten in jede Stadt des Netzwerks übertragen kann. Eine Sendergruppe ist nichts anderes als eine dominierende Menge im Netzwerkgraphen, und die domatische Zahl dieses Graphen ist die maximale Anzahl disjunkter Sendergruppen im Netzwerk.

Im erstgenannten Szenario betrachten wir ein Netzwerk aus n Computern. Die Ressourcen sollen im Netzwerk so verteilt werden, dass auf teure Dienste in der unmittelbaren Nachbarschaft eines jeden Knotens schnell zugegriffen werden kann. Hat jeder Knoten nur eine beschränkte Kapazität, so gibt es eine Schranke für die Anzahl der Ressourcen, die unterstützt werden können. Kann insbesondere jeder Knoten nur eine einzige Ressource zur Verfügung stellen, dann ist die maximale Anzahl der Ressourcen, die überhaupt unterstützt werden können, gleich der domatischen Zahl des Netzwerkgraphen.

Teure Ressourcen sollten nicht verschwendet werden. Sind ein Graph G und eine positive Zahl i gegeben, wie schwer ist es zu bestimmen, ob $\delta(G)$, die domatische Zahl von G , genau gleich i ist? Das heißt, was ist die exakte Komplexität des exakten Domatische-Zahl-Problems?

Definition 5.1 (Exaktes Domatische-Zahl-Problem). Für jede feste positive Zahl i ist das exakte Domatische-Zahl-Problem definiert durch

$$\text{Exact-}i\text{-DNP} = \{G \mid G \text{ ist ein Graph und } \delta(G) = i\}.$$

Fakt 5.2 Für jedes feste $i \geq 3$ ist Exact- i -DNP NP-hart.

Beweis. Die Aussage wird nur für den Fall $i = 3$ bewiesen; Übung 5.1 behandelt die Fälle $i > 3$.

Betrachten wir also, für $i = 3$, die im Beweis von Satz 3.58 konstruierte Reduktion f für $\text{3-Colorability} \leq_m^p \text{DNP}$. Diese Reduktion bildet einen gegebenen Graphen G auf das Paar $\langle H, 3 \rangle = f(G)$ ab, wobei H ein Graph ist, für den – wie in diesem Beweis gezeigt – die Implikationen (3.15) und (3.16) gelten:

$$\begin{aligned} G \in \text{3-Colorability} &\implies \delta(H) = 3; \\ G \notin \text{3-Colorability} &\implies \delta(H) = 2. \end{aligned}$$

Modifiziert man diese Reduktion leicht, indem man $\hat{f}(G) = H$ setzt, so erhält man eine Reduktion für $\text{3-Colorability} \leq_m^p \text{Exact-3-DNP}$. Folglich ist Exact-3-DNP NP-hart. \square

Ist $\text{Exact-}i\text{-DNP}$ sogar NP-vollständig? Das heißt, ist dieses Problem in NP? Ein naiver Versuch, Zugehörigkeit zu NP zu zeigen, misslingt wie folgt: Gegeben eine positive ganze Zahl i und ein Graph G , rate nichtdeterministisch eine Zerlegung der Knotenmenge von G in i paarweise disjunkte Mengen und verifiziere für jede geratene Zerlegung deterministisch, dass jede Menge in der Zerlegung eine dominierende Menge von G ist. Dieser Versuch klappt, falls $\delta(G) \leq i$ ist, denn der NP-Algorithmus prüft die andere Bedingung, die für die Gleichheit erforderlich ist: $\delta(G) \geq i$. Ist jedoch $\delta(G) > i$, so ist G nicht in $\text{Exact-}i\text{-DNP}$, doch der obige NP-Algorithmus akzeptiert fälschlicherweise G auf einem Berechnungspfad, der sich irrt. Was man also braucht, um $\text{Exact-}i\text{-DNP}$ zu akzeptieren, ist ein zusätzlicher Test, dass $\delta(G) > i$ nicht gilt. Dies ist aber ein coNP-Prädikat.

Es sieht also danach aus, dass der Beweis für die Mitgliedschaft von $\text{Exact-}i\text{-DNP}$ in NP, als oberer Schranke, schwer ist, wenn nicht unmöglich. Aber was ist denn dann die beste obere Schranke, die man für dieses Problem kennt? $\text{Exact-}i\text{-DNP}$ kann als Schnitt zweier Mengen geschrieben werden:

$$\text{Exact-}i\text{-DNP} = \{G \mid \delta(G) \geq i\} \cap \{G \mid \delta(G) \leq i\}.$$

Die erste Menge, $\{G \mid \delta(G) \geq i\}$, ist in NP, wohingegen $\{G \mid \delta(G) \leq i\}$, die zweite Menge, in coNP ist. Das heißt, $\text{Exact-}i\text{-DNP}$ ist der Schnitt einer NP-Menge mit einer coNP-Menge oder – anders gesagt – die Differenz zweier NP-Mengen. Durch solche „exakten“ Varianten NP-vollständiger Optimierungsprobleme motiviert, führten Papadimitriou und Yannakakis die Komplexitätsklasse DP ein, die genau die Differenzen von je zwei NP-Mengen enthält:

$$\text{DP} = \{A - B \mid A \text{ und } B \text{ sind in NP}\}.$$

Insbesondere ist $\text{Exact-}i\text{-DNP}$ in DP, denn $A - B = A \cap \overline{B}$. Und wieder kann man fragen: Ist dieses Problem DP-vollständig? Genauer, ist es DP-hart? Wir werden später sehen, dass die Antwort ja ist. Bevor wir jedoch die DP-Vollständigkeit des exakten Domatische-Zahl-Problems zeigen, stellen wir eine verallgemeinerte Version dieses Problems vor, und im Zusammenhang damit eine Verallgemeinerung von DP. Gegeben ein Graph G und eine Menge $M_k = \{i_1, i_2, \dots, i_k\}$ von k positiven ganzen Zahlen, wie schwer ist es, zu entscheiden, ob $\delta(G)$ exakt gleich einem i_j ist?

Definition 5.3 (Exaktes Domatische-Zahl-Problem (allgemeine Version)). Sei $M_k \subseteq \mathbb{N}$ eine Menge von k ganzen Zahlen. Definiere die allgemeine Version des exakten Domatische-Zahl-Problems durch

$$\text{Exact-}M_k\text{-DNP} = \{G \mid G \text{ ist ein Graph und } \delta(G) \in M_k\}.$$

Insbesondere schreiben wir $\text{Exact-}i\text{-DNP} = \{G \mid \delta(G) = i\}$ für jede einelementige Menge $M_1 = \{i\}$.

Analog zu den obigen Argumenten dafür, dass Exact- i -DNP in DP liegt, kann man zeigen, dass Exact- M_k -DNP als die Vereinigung von k Mengen in DP geschrieben werden kann. Diese Beobachtung motiviert eine Verallgemeinerung von DP: die boolesche Hierarchie über NP. Um diese Hierarchie zu definieren, verwenden wir die Symbole \wedge und \vee , welche den *komplexen Schnitt* bzw. die *komplexe Vereinigung* von Mengenklassen bezeichnen. Das heißt, für beliebige Mengenklassen \mathcal{C} und \mathcal{D} definieren wir:

$$\mathcal{C} \wedge \mathcal{D} = \{A \cap B \mid A \in \mathcal{C} \text{ und } B \in \mathcal{D}\};$$

$$\mathcal{C} \vee \mathcal{D} = \{A \cup B \mid A \in \mathcal{C} \text{ und } B \in \mathcal{D}\}.$$

Definition 5.4 (Boolesche Hierarchie über NP). Die boolesche Hierarchie über NP ist induktiv definiert durch:

$$\text{BH}_0(\text{NP}) = \text{P},$$

$$\text{BH}_1(\text{NP}) = \text{NP},$$

$$\text{BH}_2(\text{NP}) = \text{NP} \wedge \text{coNP},$$

$$\text{BH}_k(\text{NP}) = \text{BH}_{k-2}(\text{NP}) \vee \text{BH}_2(\text{NP}) \quad \text{für jedes } k \geq 3, \text{ und}$$

$$\text{BH}(\text{NP}) = \bigcup_{k \geq 0} \text{BH}_k(\text{NP}).$$

Es gilt $\text{DP} = \text{BH}_2(\text{NP}) = \text{NP} \wedge \text{coNP}$. Eine Warnung ist hier nötig: $\text{NP} \wedge \text{coNP}$ und $\text{NP} \cap \text{coNP}$ sind (höchstwahrscheinlich) *verschiedene* Komplexitätsklassen! Mengentheoretisch findet die *Schnittbildung* $\text{NP} \cap \text{coNP}$ der Mengenklassen NP und coNP auf einer anderen Stufe statt als die *komplexe Schnittbildung* $\text{NP} \wedge \text{coNP}$. Das heißt, $\text{NP} \wedge \text{coNP}$ ist die Menge der Schnittmengen von NP-Mengen mit coNP-Mengen, wohingegen $\text{NP} \cap \text{coNP}$ die Menge der Mengen ist, die sowohl in NP und in coNP liegen. Freilich, wenn etwa NP gleich P ist, so sind auch coNP, $\text{NP} \wedge \text{coNP}$, $\text{NP} \cap \text{coNP}$ und viele weitere Klassen gleich P, siehe Übung 5.3. Die Gleichheit $\text{NP} = \text{P}$ wird jedoch für höchst unwahrscheinlich gehalten.

Die obige Anmerkung über die komplexe Vereinigung und den komplexen Schnitt von Mengenklassen gilt ebenso für den co-Operator, dessen Anwendung auf eine beliebige Klasse \mathcal{C} die Klasse $\text{co}\mathcal{C} = \{\overline{L} \mid L \in \mathcal{C}\}$ hervorbringt, die Klasse der Komplemente von Mengen in \mathcal{C} . Beispielsweise ist coNP , die Klasse der Komplemente von NP-Mengen, etwas ganz Anderes als $\overline{\text{NP}} = \{L \mid L \notin \text{NP}\}$, das Komplement von NP, welches jede Menge enthält, die keine NP-Menge ist.

Definition 5.5 (Boolesche Hülle). Sei \mathcal{C} eine Klasse von Mengen, und seien A und B Mengen. Definiere die boolesche Hülle von \mathcal{C} , bezeichnet mit $BC(\mathcal{C})$, als die kleinste Klasse \mathcal{D} von Mengen, die \mathcal{C} enthält und unter den booleschen Operationen Vereinigung, Schnitt und Komplement abgeschlossen ist. Dabei sagen wir:

- \mathcal{D} ist abgeschlossen unter Vereinigung, falls gilt: Sind A und B in \mathcal{D} , so auch $A \cup B$;
- \mathcal{D} ist abgeschlossen unter Schnitt, falls gilt: Sind A und B in \mathcal{D} , so auch $A \cap B$;
- \mathcal{D} ist abgeschlossen unter Komplement, falls gilt: Ist A in \mathcal{D} , so auch \overline{A} .

Betrachte insbesondere $BC(NP)$, die boolesche Hülle von NP . Die ersten beiden Abschlusseigenschaften (unter Vereinigung und Schnitt) können ebenso durch $NP \vee NP = NP$ bzw. $NP \wedge NP = NP$ ausgedrückt werden, und sie gelten unmittelbar für NP ; siehe Übung 5.4. Ob jedoch auch die dritte Abschlusseigenschaft (unter Komplement) gilt, ist für NP eine große offene Frage, beinahe so schwierig und berücksichtigt wie die P-versus-NP-Frage. Die meisten Komplexitätstheoretiker glauben, dass $NP \neq coNP$ gilt, d.h., dass NP nicht unter Komplement abgeschlossen ist. Daher glaubt man auch, dass $NP \neq BC(NP)$ gilt; siehe Übung 5.5. Ein Beweis der Ungleichheit $NP \neq BC(NP)$ jedoch würde unmittelbar einen Beweis der Ungleichheit $NP \neq coNP$ nach sich ziehen, woraus wiederum sofort $P \neq NP$ folgen würde.

Mengenklassen, die unter Vereinigung und Schnitt abgeschlossen sind und die \emptyset und Σ^* enthalten, wie z.B. die Mengenklasse NP , nennt man *Mengenringe*, mit den Ringoperationen Vereinigung und Schnitt. Ein Mengenring, der zusätzlich auch unter Komplement abgeschlossen ist, heißt *boolesche Algebra*. Für jede boolesche Algebra \mathcal{A} gilt $BC(\mathcal{A}) = \mathcal{A}$. Hausdorff [Hau14] führte die boolesche Hierarchie über beliebigen Mengenringen als die „Vereinigung-von-Differenzen-Hierarchie“ ein, welche für den Mengenring NP in Definition 5.4 gegeben wird.¹ Daher nennt man diese Normalform der booleschen Hierarchie manchmal auch die *Hausdorff-Hierarchie*; in Abschnitt 5.9 findet man noch andere Normalformen der booleschen Hierarchie (über NP) sowie von booleschen Hierarchien über anderen Klassen als NP .

Eine dieser oben erwähnten anderen Normalformen der booleschen Hierarchie ist die „verschachtelte Differenzenhierarchie“, die Stufe für Stufe mit der „Vereinigung-von-Differenzen-Hierarchie“ aus Definition 5.4 übereinstimmt. Satz 5.6 unten sagt, dass diese beiden Normalformen der booleschen Hierarchie über beliebigen Mengenringen Stufe für Stufe äquivalent sind, und auch, dass beide die boolesche Hülle des zugrunde liegenden Mengenrings umfassen. Ist die zugrunde liegende Mengenklasse jedoch kein Ring, so sind die entsprechenden (stufenweisen) Äquivalenzen nicht so klar. Für bestimmte Normalformen der booleschen Hierarchie (die hier nicht genauer erläutert werden sollen) zeigten L. Hemaspandra und Rothe, dass der Abschluss der zugrunde liegenden Mengenklasse unter Schnitt bereits genügt, um die boolesche Hülle der Klasse zu umfassen; siehe die Diskussion in Abschnitt 5.9. Ein Beispiel einer Klasse, die zwar unter Schnitt, aber vermutlich nicht unter Vereinigung abgeschlossen ist, ist die Klasse UP ; siehe Übung 5.6.

¹ Setzt man in Definition 5.4 für NP irgendeine Mengenklasse \mathcal{C} ein, so definiert man die boolesche Hierarchie über \mathcal{C} als $BH(\mathcal{C}) = \bigcup_{k \geq 0} BH_k(\mathcal{C})$, deren Stufen die Klassen $BH_k(\mathcal{C})$ sind.

Wenn wir verschachtelte Differenzen von Mengen schreiben, legen wir per Konvention fest, dass zur besseren Lesbarkeit die Klammern weggelassen werden dürfen:

$$A_1 - A_2 - \cdots - A_{k-1} - A_k = A_1 - (A_2 - (\cdots - (A_{k-1} - A_k) \cdots)),$$

auch wenn das Bilden von Mengendifferenzen keine assoziative Operation ist.

Satz 5.6. Für jeden Mengenring \mathcal{C} gelten die folgenden beiden Aussagen:

1. Die verschachtelte Differenzenhierarchie über \mathcal{C} stimmt Stufe für Stufe mit der Vereinigung-von-Differenzen-Hierarchie über \mathcal{C} überein:

$$\text{BH}_k(\mathcal{C}) = \left\{ L \left| \begin{array}{l} L = A_1 - A_2 - \cdots - A_k \text{ für Mengen } A_i \text{ in } \mathcal{C}, \\ 1 \leq i \leq k, \text{ wobei } A_k \subseteq A_{k-1} \subseteq \cdots \subseteq A_1 \end{array} \right. \right\}.$$

2. $\text{BH}(\mathcal{C}) = \text{BC}(\mathcal{C})$.

Beweis. Beide Aussagen können durch elementare Mengentransformationen gezeigt werden; siehe Übung 5.7. \square

Korollar 5.7. 1. Die verschachtelte Differenzenhierarchie über NP stimmt Stufe für Stufe mit der Vereinigung-von-Differenzen-Hierarchie über NP überein:

$$\text{BH}_k(\text{NP}) = \left\{ L \left| \begin{array}{l} L = A_1 - A_2 - \cdots - A_k \text{ für Mengen } A_i \text{ in NP,} \\ 1 \leq i \leq k, \text{ wobei } A_k \subseteq A_{k-1} \subseteq \cdots \subseteq A_1 \end{array} \right. \right\}.$$

2. $\text{BH}(\text{NP}) = \text{BC}(\text{NP})$.

Satz 5.8 gibt die Inklusionsbeziehungen zwischen den Stufen der booleschen Hierarchie über NP an, welche unmittelbar aus Definition 5.4 folgen. Abbildung 5.1 stellt diese Inklusionsstruktur als ein Hasse-Diagramm dar. Das heißt, ist eine Klasse \mathcal{C} in einer Klasse \mathcal{D} enthalten, so wird dies durch eine aufsteigende Linie von \mathcal{C} zu \mathcal{D} angezeigt. Unvergleichbare Klassen – also Klassen, für die weder die Inklusion $\mathcal{C} \subseteq \mathcal{D}$ noch die Inklusion $\mathcal{D} \subseteq \mathcal{C}$ bekannt ist – sind nicht durch eine Linie verbunden. Abbildung 5.2 zeigt dieselbe Inklusionsstruktur als ein Venn-Diagramm, wobei BH_1 statt $\text{BH}_1(\text{NP})$ usw. geschrieben wird, um die Lesbarkeit zu erhöhen. Hier sind dunklere Klassen in helleren Klassen enthalten, und unvergleichbare Klassen haben dieselbe Graustufe. Von keiner der Inklusionen ist bekannt, ob sie echt ist oder nicht.

Satz 5.8. Für jedes $k \geq 0$ gilt

$$\text{BH}_k(\text{NP}) \subseteq \text{BH}_{k+1}(\text{NP}) \text{ und } \text{coBH}_k(\text{NP}) \subseteq \text{coBH}_{k+1}(\text{NP}) \quad (5.1)$$

und somit $\text{BH}_k(\text{NP}) \cup \text{coBH}_k(\text{NP}) \subseteq \text{BH}_{k+1}(\text{NP})$.

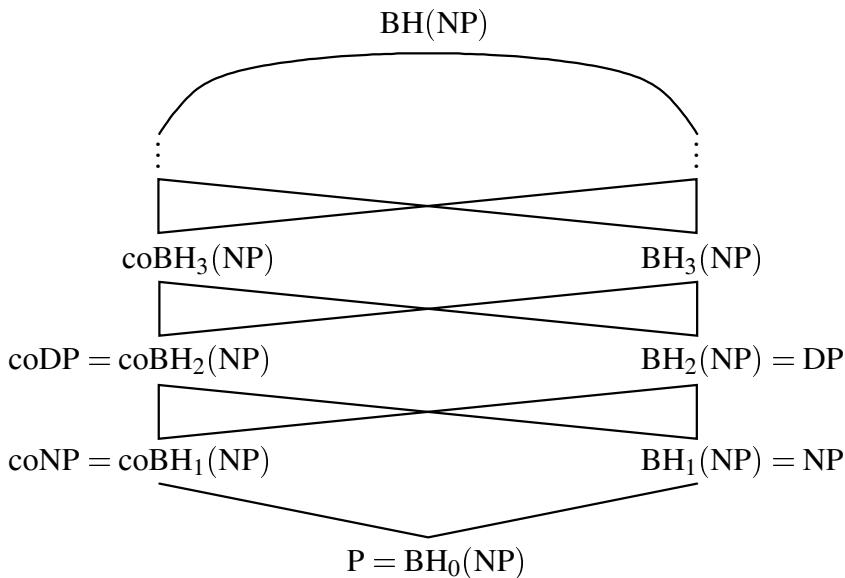


Abb. 5.1. Boolesche Hierarchie über NP (Hasse-Diagramm)

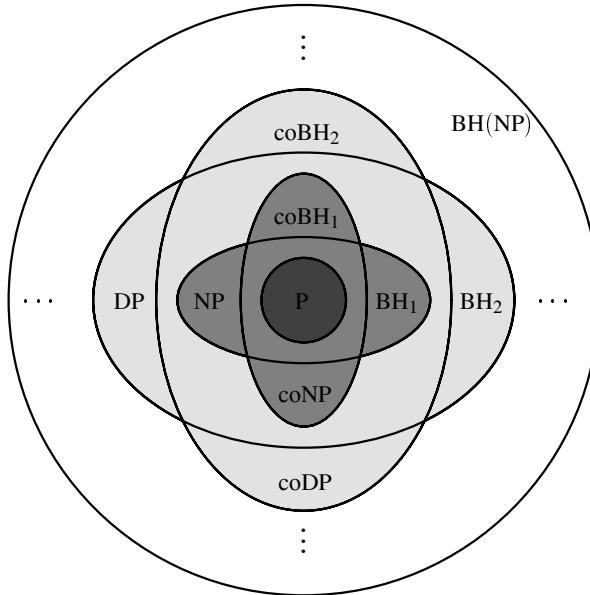
Story 5.9 Stell dir die boolesche Hierarchie als ein Gebäude vor; ein gewaltiges, prachtvolles, grandioses Paar von Zwillingsstürmen, deren Stockwerke die Stufen $BH_k(NP)$ bzw. $coBH_k(NP)$ sind. Wie viele Stockwerke gibt es? Ist $BH_k(NP)$ eine echte Hierarchie, die aus einer unendlichen Anzahl von tatsächlich unterschiedlichen Stufen besteht? Oder ist sie endlich? Niemand weiß es. Wie so oft in der Komplexitätstheorie ist dies eine offene Forschungsfrage, die eng mit der P-versus-NP-Frage zusammenhängt.

Nun stell dir einen Attentäter vor, der durch den Boolean Hierarchy Tower stürmt und das 71. Stockwerk mit Handgranaten und Sprengstoffen so schlimm beschädigt, dass es vollständig zerstört wird. Es ist klar, was mit dem Gebäude passiert: Es kollabiert. Satz 5.10 zeigt, dass dies ganz genau das ist, was in diesem Fall mit der booleschen Hierarchie geschieht: Kollabiert eine ihrer Stufen auf die nächst-niedrigere Stufe, dann kollabiert die gesamte Hierarchie nach unten auf diese spezifische, endliche Stufe. Selbst wenn eine der Klassen $BH_k(NP)$ mit ihrer komplementären Klasse im anderen Zwillingssturm, $coBH_k(NP)$, übereinstimmen sollte, tritt dieselbe Konsequenz ein. Diese Eigenschaft einer Komplexitätshierarchie nennt man die „Kollapseseigenschaft nach oben“ (upward collapse) oder, synonym dazu, die „Separationseigenschaft nach unten“ (downward separation).

Satz 5.10. 1. Für jedes $k \geq 0$ gilt: Ist $BH_k(NP) = BH_{k+1}(NP)$, dann ist

$$BH_k(NP) = coBH_k(NP) = BH_{k+1}(NP) = coBH_{k+1}(NP) = \dots = BH(NP).$$

2. Für jedes $k \geq 1$ gilt: Ist $BH_k(NP) = coBH_k(NP)$, dann ist

**Abb. 5.2.** Boolesche Hierarchie über NP (Venn-Diagramm)

$$\text{BH}_k(\text{NP}) = \text{coBH}_k(\text{NP}) = \text{BH}_{k+1}(\text{NP}) = \text{coBH}_{k+1}(\text{NP}) = \cdots = \text{BH}(\text{NP}).$$

Beweis. Um die erste Aussage zu beweisen, nehmen wir an, dass $\text{BH}_k(\text{NP}) = \text{BH}_{k+1}(\text{NP})$ für ein festes $k \geq 0$ gilt. Wir zeigen durch Induktion über n :

$$(\forall n \geq k) [\text{BH}_n(\text{NP}) = \text{coBH}_n(\text{NP}) = \text{BH}_{n+1}(\text{NP})]. \quad (5.2)$$

Der Induktionsanfang, $n = k$, folgt unmittelbar aus (5.1) in Satz 5.8 und der Annahme $\text{BH}_k(\text{NP}) = \text{BH}_{k+1}(\text{NP})$:

$$\text{coBH}_k(\text{NP}) \subseteq \text{BH}_{k+1}(\text{NP}) = \text{BH}_k(\text{NP}),$$

woraus sofort $\text{coBH}_k(\text{NP}) = \text{BH}_k(\text{NP})$ folgt.

Die Induktionsvoraussetzung sagt, dass (5.2) für ein $n \geq k$ gilt. Wir müssen zeigen, dass $\text{BH}_{n+1}(\text{NP}) = \text{BH}_{n+2}(\text{NP})$. Mit einem zum Induktionsanfang analogen Argument folgt daraus $\text{BH}_{n+1}(\text{NP}) = \text{coBH}_{n+1}(\text{NP})$.

Sei X eine Menge in $\text{BH}_{n+2}(\text{NP})$. Dann gibt es Mengen A_1, A_2, \dots, A_{n+2} in NP mit $A_{n+2} \subseteq A_{n+1} \subseteq \cdots \subseteq A_1$ und $X = A_1 - A_2 - \cdots - A_{n+2}$.

Sei $Y = A_2 - A_3 - \cdots - A_{n+2}$. Somit ist $Y \in \text{BH}_{n+1}(\text{NP})$. Nach der Induktionsvoraussetzung ist Y in $\text{BH}_n(\text{NP}) = \text{BH}_{n+1}(\text{NP})$ enthalten und lässt sich daher als $Y = B_1 - B_2 - \cdots - B_n$ darstellen, für geeignete NP-Mengen B_1, B_2, \dots, B_n mit $B_n \subseteq B_{n-1} \subseteq \cdots \subseteq B_1$. Nach unserer Wahl der Mengen A_i , gilt $Y \subseteq A_1$, woraus $Y \cap A_1 = Y$ folgt. Daher gilt

$$Y = (B_1 \cap A_1) - (B_2 \cap A_1) - \cdots - (B_n \cap A_1).$$

Jede der Mengen $B_i \cap A_1$ ist in NP, da NP unter Schnitt abgeschlossen ist; siehe Übung 5.4(a). Es folgt, dass

$$X = A_1 - Y = A_1 - (B_1 \cap A_1) - (B_2 \cap A_1) - \cdots - (B_n \cap A_1),$$

eine Menge in $\text{BH}_{n+1}(\text{NP})$ ist, wobei

$$(B_n \cap A_1) \subseteq (B_{n-1} \cap A_1) \subseteq \cdots \subseteq (B_1 \cap A_1) \subseteq A_1.$$

Damit ist der Induktionsbeweis abgeschlossen und (5.2) bewiesen.

Um die zweite Aussage zu zeigen, nehmen wir an, dass $\text{BH}_k(\text{NP}) = \text{coBH}_k(\text{NP})$ für ein festes $k \geq 1$ gilt. Wir zeigen, dass aus dieser Annahme $\text{BH}_{k+1}(\text{NP}) = \text{BH}_k(\text{NP})$ folgt, womit wir die zweite Aussage auf die erste Aussage des Satzes reduziert haben.

Sei $X = A_1 - A_2 - \cdots - A_{k+1}$ eine Menge in $\text{BH}_{k+1}(\text{NP})$, wobei die Mengen A_1, A_2, \dots, A_{k+1} in NP sind und die Inklusionen $A_{k+1} \subseteq A_k \subseteq \cdots \subseteq A_1$ erfüllen.

Folglich ist $Y = A_2 - A_3 - \cdots - A_{k+1}$ eine Menge in $\text{BH}_k(\text{NP})$. Nach unserer Annahme ist Y in $\text{coBH}_k(\text{NP}) = \text{BH}_k(\text{NP})$. Somit ist \overline{Y} eine Menge in $\text{BH}_k(\text{NP})$. Seien B_1, B_2, \dots, B_k Mengen in NP mit $\overline{Y} = B_1 - B_2 - \cdots - B_k$ und $B_k \subseteq B_{k-1} \subseteq \cdots \subseteq B_1$. Wieder liegt jede der Mengen $A_1 \cap B_i$, $1 \leq i \leq k$, in NP, denn NP ist unter Schnitt abgeschlossen. Außerdem gilt $A_1 \cap B_k \subseteq A_1 \cap B_{k-1} \subseteq \cdots \subseteq A_1 \cap B_1$. Es folgt, dass

$$X = A_1 - Y = A_1 \cap \overline{Y} = (A_1 \cap B_1) - (A_1 \cap B_2) - \cdots - (A_1 \cap B_k)$$

eine Menge in $\text{BH}_k(\text{NP})$ ist, womit wie gewünscht $\text{BH}_{k+1}(\text{NP}) = \text{BH}_k(\text{NP})$ gezeigt ist. Das im Beweis der ersten Aussage des Satzes gegebene Argument kann nun angewandt werden, um den Kollaps $\text{BH}_k(\text{NP}) = \text{BH}(\text{NP})$ zu zeigen. \square

Kehren wir nun zum exakten Domatische-Zahl-Problem zurück. Wie erwähnt liegt *Exact- i -DNP* für jedes feste i in DP. Folglich ist für jedes $k \geq 1$ und für jede Menge M_k von k positiven ganzen Zahlen das Problem *Exact- M_k -DNP* in $\text{BH}_{2k}(\text{NP})$. Gibt es *untere Schranken* für *Exact- M_k -DNP*, die mit diesen oberen Schranken $\text{BH}_{2k}(\text{NP})$ übereinstimmen? Mit anderen Worten, ist *Exact- M_k -DNP* vollständig für $\text{BH}_{2k}(\text{NP})$? Es wurde bereits festgestellt, dass die Antwort ja ist.

Das folgende Lemma von Wagner [Wag87a] gibt hinreichende Bedingungen für den Nachweis der $\text{BH}_i(\text{NP})$ -Härte für $i \geq 1$ an. Ersetzt man „ungerade“ durch „gerade“ in (5.3) von Lemma 5.11, so erhält man analoge hinreichende Bedingungen für den Nachweis der $\text{coBH}_i(\text{NP})$ -Härte.

Wagners Technik zeigt ganz allgemein, wie man aus unteren Schranken für NP (nämlich aus der NP-Härte) untere Schranken (also Härte) für Klassen oberhalb von NP gewinnen kann. Abschnitt 5.3 wird eine andere Bedingung von Wagner angeben, die denen aus Lemma 5.11 sehr ähnlich ist und hinreichend für den Nachweis der Härte für eine Klasse ist, die noch größer als die boolesche Hierarchie ist. Anwendungen der Wagner-Technik werden in Abschnitt 5.9 behandelt.

Lemma 5.11 (Wagner). Seien A ein NP-vollständiges und B ein beliebiges Problem, und sei $k \geq 1$ fest gewählt. Angenommen, es gibt eine polynomialzeit-berechenbare Funktion f , so dass für alle Wörter $x_1, x_2, \dots, x_k \in \Sigma^*$ mit der Eigenschaft, dass $x_j \in A$ aus $x_{j+1} \in A$ für jedes j mit $1 \leq j < k$ folgt, die Äquivalenz

$$\|\{i \mid x_i \in A\}\| \text{ ist ungerade} \iff f(\langle x_1, x_2, \dots, x_k \rangle) \in B \quad (5.3)$$

gilt. Dann ist B ein $\text{BH}_k(\text{NP})$ -hartes Problem.

Beweis. Fixiere ein NP-vollständiges Problem A und ein $k \geq 1$ und betrachte ein beliebiges Problem B . Es genügt, den Satz nur für gerade k zu zeigen, der Fall ungerader Zahlen k kann analog bewiesen werden. Sei also $k = 2m$ für ein $m \geq 1$. Angenommen, eine Funktion $f \in \text{FP}$ erfüllt die Äquivalenz (5.3) für alle Wörter $x_1, x_2, \dots, x_k \in \Sigma^*$ mit der Eigenschaft, dass $x_j \in A$ aus $x_{j+1} \in A$ für jedes j mit $1 \leq j < k$ folgt.

Sei X eine beliebige Menge in $\text{BH}_k(\text{NP})$. Nach Satz 5.6 gibt es NP-Mengen Y_1, Y_2, \dots, Y_k , so dass $Y_k \subseteq Y_{k-1} \subseteq \dots \subseteq Y_1$ gilt und

$$X = Y_1 - Y_2 - \dots - Y_k = \bigcup_{i=1}^m (Y_{2i-1} \cap \overline{Y_{2i}}). \quad (5.4)$$

Da A NP-vollständig ist, gibt es k Reduktionen r_1, r_2, \dots, r_k in FP , so dass für jedes $x \in \Sigma^*$ gilt: $x \in Y_j$ genau dann, wenn $r_j(x) \in A$ für jedes j mit $1 \leq j \leq k$.

Da andererseits $Y_k \subseteq Y_{k-1} \subseteq \dots \subseteq Y_1$ gilt, folgt $r_j(x) \in A$ aus $r_{j+1}(x) \in A$ für jedes j mit $1 \leq j < k$. Nach (5.3) und (5.4) ergibt sich für jedes $x \in \Sigma^*$:

$$\begin{aligned} x \in X &\stackrel{(5.4)}{\iff} \|\{i \mid r_i(x) \in A\}\| \text{ ist ungerade} \\ &\stackrel{(5.3)}{\iff} f(\langle r_1(x), r_2(x), \dots, r_k(x) \rangle) \in B. \end{aligned}$$

Somit gilt $X \leq_m^p B$ mittels $f \in \text{FP}$. Da X eine beliebige Menge in $\text{BH}_k(\text{NP})$ ist, folgt die $\text{BH}_k(\text{NP})$ -Härte von B wie gewünscht. \square

Satz 5.12 wendet Wagners hinreichende Bedingung aus Lemma 5.11 auf das exakte Domatische-Zahl-Problem an: Für jede fixierte Menge M_k , die k nicht unmittelbar aufeinander folgende Zahlen nicht kleiner als $4k + 1$ enthält, ist $\text{Exact-}M_k\text{-DNP}$ $\text{BH}_{2k}(\text{NP})$ -vollständig. Für $k = 1$ sagt Korollar 5.13, dass insbesondere das Problem Exact-5-DNP DP-vollständig ist.

Satz 5.12. Für ein festes $k \geq 1$ sei $M_k = \{4k + 1, 4k + 3, \dots, 6k - 1\}$. Dann ist $\text{Exact-}M_k\text{-DNP}$ ein $\text{BH}_{2k}(\text{NP})$ -vollständiges Problem.

Korollar 5.13. Exact-5-DNP ist DP-vollständig.

Beweis von Satz 5.12. Wie bereits gesehen, ist $\text{Exact-}M_k\text{-DNP}$ in $\text{BH}_{2k}(\text{NP})$ enthalten. Es bleibt die $\text{BH}_{2k}(\text{NP})$ -Härte von $\text{Exact-}M_k\text{-DNP}$ zu beweisen. Zu diesem Zweck wird Lemma 5.11 mit 3-Colorability als dem NP-vollständigen Problem A und mit $\text{Exact-}M_k\text{-DNP}$ als dem Problem B aus diesem Lemma angewandt.

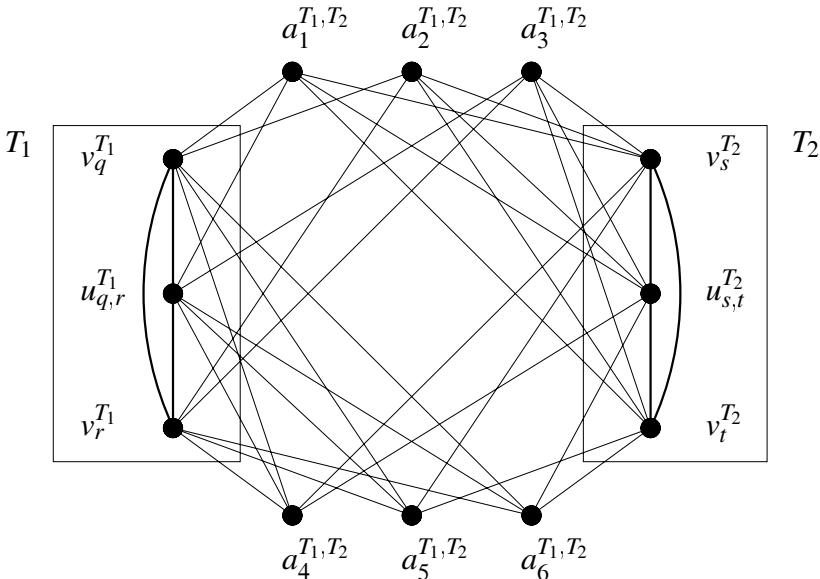


Abb. 5.3. Exact-5-DNP ist DP-vollständig: Verbindung der zwei Dreiecke T_1 und T_2

Fixiere beliebige $2k$ Graphen G_1, G_2, \dots, G_{2k} , die so geordnet sind, dass für jedes j mit $1 \leq j < 2k$ gilt: Ist G_{j+1} 3-färbbar, so ist auch G_j 3-färbbar. Nach dem Beweis von Satz 3.56, welcher eine Reduktion von 3-SAT auf 3-Colorability angibt, dürfen wir Folgendes annehmen:

- Für die chromatische Zahl $\chi(G_j)$ eines jeden Graphen G_j gilt $3 \leq \chi(G_j) \leq 4$.
- Keiner der Graphen G_j enthält isolierte Knoten.

Löse Übung 5.10, um zu verstehen, weshalb diese Annahmen ohne Beschränkung der Allgemeinheit vorausgesetzt werden können.

Im Beweis von Satz 3.58 wurde eine Reduktion $g \in \text{FP}$ von 3-Colorability auf DNP angegeben, die einen gegebenen Graphen G auf das Paar $\langle H, 3 \rangle = g(G)$ abbildet. Dabei ist H ein Graph, der die Implikationen (3.15) und (3.16) erfüllt:

$$\begin{aligned} G \in \text{3-Colorability} &\implies \delta(H) = 3; \\ G \notin \text{3-Colorability} &\implies \delta(H) = 2. \end{aligned}$$

Wendet man g auf jeden der Graphen G_j an, wobei $1 \leq j \leq 2k$, so erhält man $2k$ Graphen $H_j = g(G_j)$, die alle die Implikationen (3.15) und (3.16) erfüllen. Folglich ist $\delta(H_j) \in \{2, 3\}$ für jedes j , und es folgt $\delta(H_j) = 3$ aus $\delta(H_{j+1}) = 3$.

Definiere eine polynomialzeit-berechenbare Funktion f , die die gegebenen Graphen G_1, G_2, \dots, G_{2k} in einen Graphen H transformiert, so dass die Äquivalenz (5.3) aus Lemma 5.11 gilt. Der Graph H wird dabei so aus den Graphen H_1, H_2, \dots, H_{2k} konstruiert, dass $\delta(H) = \sum_{j=1}^{2k} \delta(H_j)$ gilt.

Bevor die allgemeine Konstruktion angegeben wird, betrachten wir zunächst den Spezialfall $k = 1$. Für $k = 1$ sind zwei Graphen gegeben, $H_1 = g(G_1)$ und $H_2 = g(G_2)$. Konstruiere einen Teilgraphen, der H_1 und H_2 wie folgt verbindet. Es ist dafür nützlich, sich an die Konstruktion aus dem Beweis von Satz 3.58 zu erinnern. Sei also T_1 mit $V(T_1) = \{v_q^{T_1}, u_{q,r}^{T_1}, v_r^{T_1}\}$ eines der Dreiecke in H_1 , und sei T_2 mit $V(T_2) = \{v_s^{T_2}, u_{s,t}^{T_2}, v_t^{T_2}\}$ eines der Dreiecke in H_2 . Verbinde T_1 und T_2 mittels der Kanten und neuen Knoten, die in Abbildung 5.3 zu sehen sind. Genauer gesagt, füge sechs neue Knoten $a_1^{T_1,T_2}, a_2^{T_1,T_2}, \dots, a_6^{T_1,T_2}$ hinzu und füge die neuen Kanten ein, die in dieser Abbildung durch dünne Linien dargestellt sind.

Verbinde jedes Paar von Dreiecken in H_1 und H_2 durch disjunkte Kopien des Teilgraphen aus Abbildung 5.3. Der resultierende Graph sei H . Für jeden Graphen G ist $\delta(G) \leq \min\deg(G) + 1$, wobei $\min\deg(G)$ den Minimalgrad der Knoten von G bezeichnet. (Der *Grad eines Knoten* v in einem Graphen G ist die Anzahl der aus v auslaufenden Kanten.) Da der Grad eines jeden Knoten a_i in dem Teilgraphen aus Abbildung 5.3 genau 5 ist, gilt $\delta(H) \leq 6$, egal, welche domatischen Zahlen H_1 und H_2 haben. Es wird nun gezeigt, dass $\delta(H) = \delta(H_1) + \delta(H_2)$ gilt.

Seien $D_1, D_2, \dots, D_{\delta(H_1)}$ $\delta(H_1)$ paarweise disjunkte Mengen, die H_1 dominieren, und seien $D_{\delta(H_1)+1}, D_{\delta(H_1)+2}, \dots, D_{\delta(H_1)+\delta(H_2)}$ $\delta(H_2)$ paarweise disjunkte Mengen, die H_2 dominieren. Unterscheide die folgenden drei Fälle.

Fall 1: $\delta(H_1) = \delta(H_2) = 3$. Betrachte ein festes D_j , wobei $1 \leq j \leq 3$. Da H_1 von D_j dominiert wird, hat jedes Dreieck T_1 von H_1 genau einen Knoten in D_j . Sei T_1 ein solches Dreieck mit $V(T_1) = \{v_q^{T_1}, u_{q,r}^{T_1}, v_r^{T_1}\}$ und, sagen wir, $V(T_1) \cap D_j = \{v_q^{T_1}\}$; die anderen Fälle sind ganz analog.

Betrachte nun für jedes Dreieck T_2 von H_2 , sagen wir konkret für T_2 mit $V(T_2) = \{v_s^{T_2}, u_{s,t}^{T_2}, v_t^{T_2}\}$, die sechs Knoten $a_1^{T_1,T_2}, a_2^{T_1,T_2}, \dots, a_6^{T_1,T_2}$, die T_1 und T_2 wie in Abbildung 5.3 verbinden. Genau einer dieser Knoten, nämlich $a_3^{T_1,T_2}$, ist nicht mit $v_q^{T_1}$ benachbart. Füge für jedes T_2 solche fehlenden Knoten zu D_j hinzu und definiere

$$\hat{D}_j = D_j \cup \{a_3^{T_1,T_2} \mid T_2 \text{ ist ein Dreieck von } H_2\}.$$

Da jeder Knoten von H_2 zu einem Dreieck T_2 von H_2 gehört und da $a_3^{T_1,T_2}$ zu einem jeden Knoten in T_2 benachbart ist, wird H_2 von \hat{D}_j dominiert. Außerdem dominiert \hat{D}_j , welches D_j enthält, den Graphen H_1 , und weil $v_q^{T_1}$ zu einem jeden Knoten $a_i^{T_1,T_2}$ außer zu $a_3^{T_1,T_2}$ für jedes Dreieck T_2 von H_2 benachbart ist, dominiert \hat{D}_j jeden Knoten in den neu hinzugefügten Teilgraphen von H . Folglich wird der ganze Graph H von \hat{D}_j dominiert.

Mit einem symmetrischen Argument zeigt man, dass jede H_2 dominierende Menge D_j , $4 \leq j \leq 6$, zu einer Menge \hat{D}_j erweitert werden kann, die den ganzen Graphen H dominiert. Nach Konstruktion sind die Mengen \hat{D}_j mit $1 \leq j \leq 6$ paarweise disjunkt. Somit gilt $\delta(H) = 6 = \delta(H_1) + \delta(H_2)$.

Fall 2: $\delta(H_1) = 3$ und $\delta(H_2) = 2$. Wie in Fall 1 können die Knoten in geeigneter Weise auf die fünf gegebenen Mengen D_1, D_2, \dots, D_5 verteilt werden, so

dass man fünf paarweise disjunkte Mengen $\hat{D}_1, \hat{D}_2, \dots, \hat{D}_5$ erhält, die jeweils den gesamten Graphen H dominieren. Es folgt $5 \leq \delta(H) \leq 6$. Zu zeigen bleibt $\delta(H) \neq 6$.

Nimm für einen Widerspruch an, es würde $\delta(H) = 6$ gelten. Betrachte in Abbildung 5.3 den Teilgraphen zwischen je zwei Dreiecken T_1 und T_2 in H_1 bzw. in H_2 . T_1 hat die Knoten $V(T_1) = \{v_q^{T_1}, u_{q,r}^{T_1}, v_r^{T_1}\}$. Die einzige Möglichkeit (abgesehen vom Umbenennen der dominierenden Mengen), den Graphen H in sechs dominierende Mengen zu zerlegen, etwa in E_1, E_2, \dots, E_6 , besteht darin, den Mengen E_i die Knoten von T_1 , von H_2 und von den neu hinzugefügten Teilgraphen, die mit T_1 verbunden sind, wie folgt zuzuweisen:

- $\{v_q^{T_1}\} \cup \{a_3^{T_1, T_2} | T_2 \text{ ist ein Dreieck in } H_2\}$ zu E_1 ;
- $\{u_{q,r}^{T_1}\} \cup \{a_2^{T_1, T_2} | T_2 \text{ ist ein Dreieck in } H_2\}$ zu E_2 ;
- $\{v_r^{T_1}\} \cup \{a_1^{T_1, T_2} | T_2 \text{ ist ein Dreieck in } H_2\}$ zu E_3 ;
- $\{v_s^{T_2}, a_6^{T_1, T_2} | T_2 \text{ ist ein Dreieck in } H_2\}$ zu E_4 ;
- $\{u_{s,t}^{T_2}, a_5^{T_1, T_2} | T_2 \text{ ist ein Dreieck in } H_2\}$ zu E_5 ;
- $\{v_t^{T_2}, a_4^{T_1, T_2} | T_2 \text{ ist ein Dreieck in } H_2\}$ zu E_6 .

Also müssen alle Knoten von H_2 auf die drei dominierenden Mengen E_4, E_5 und E_6 verteilt werden, was eine Zerlegung von H_2 in drei dominierende Mengen induziert. Dies widerspricht aber der Fallannahme $\delta(H_2) = 2$. Es folgt $\delta(H) = 5 = \delta(H_1) + \delta(H_2)$.

Fall 3: $\delta(H_1) = \delta(H_2) = 2$. Wie in den vorherigen beiden Fällen können die Knoten in geeigneter Weise auf die vier gegebenen Mengen D_1, D_2, D_3 und D_4 verteilt werden, so dass man eine Zerlegung von $V(H)$ in vier paarweise disjunkte Mengen $\hat{D}_1, \hat{D}_2, \hat{D}_3$ und \hat{D}_4 erhält, die jeweils den gesamten Graphen H dominieren. Es folgt $4 \leq \delta(H) \leq 6$. Mit demselben Argument wie im Fall 2 überlegt man sich, dass $\delta(H) \neq 6$ gilt. Es bleibt zu zeigen, dass $\delta(H) \neq 5$ gilt.

Nimm für einen Widerspruch an, es würde $\delta(H) = 5$ gelten. Betrachte in Abbildung 5.3 den Teilgraphen zwischen je zwei Dreiecken T_1 und T_2 in H_1 bzw. in H_2 . Die Knotenmenge von H sei in fünf dominierende Mengen zerlegt, E_1, E_2, \dots, E_5 .

Zunächst zeigen wir, dass weder T_1 noch T_2 zwei Knoten haben kann, die zur selben dominierenden Menge gehören. Angenommen, dies wäre doch so. Seien also etwa $v_q^{T_1}$ und $u_{q,r}^{T_1}$ beide in E_1 , und sei $v_r^{T_1}$ in E_2 (alle anderen Fälle können analog behandelt werden). Dann müssen die Knoten $v_s^{T_2}, u_{s,t}^{T_2}$ und $v_t^{T_2}$ aus T_2 den anderen drei dominierenden Mengen zugeordnet werden, E_3, E_4 und E_5 , denn andernfalls würde eine der Mengen E_i nicht sämtliche neu hinzugefügten Knoten $a_j^{T_1, T_2}, 1 \leq j \leq 6$, dominieren. Da T_1 mit jedem Dreieck in H_2 mittels eines solchen Teilgraphen verbunden ist, zeigt dasselbe Argument wie oben, dass $V(H_2)$ in drei dominierende Mengen zerlegt werden kann, was wiederum der Fallannahme $\delta(H_2) = 2$ widerspricht.

Folglich sind die Knoten von T_1 drei verschiedenen dominierenden Mengen zugeordnet, etwa den Mengen E_1, E_2 und E_3 . Dann muss jedes Dreieck T_2 von H_2 einen seiner Knoten in E_4 , einen in E_5 , und einen in einer der Mengen E_1, E_2

oder E_3 haben. Wieder induziert dies eine Zerlegung von H_2 in drei dominierende Mengen, was der Annahme $\delta(H_2) = 2$ widerspricht. Es folgt $\delta(H) \neq 5$, also $\delta(H) = 4 = \delta(H_1) + \delta(H_2)$.

Nach Konstruktion folgt $\delta(H_1) = 3$ aus $\delta(H_2) = 3$. Demnach kann der Fall „ $\delta(H_1) = 2$ und $\delta(H_2) = 3$ “ nicht eintreten. Die Fallunterscheidung ist vollständig.

Definiere $f(\langle G_1, G_2 \rangle) = H$. Offenbar kann f in Polynomialzeit berechnet werden und erfüllt (5.3) nach der obigen Fallunterscheidung:

$$\begin{aligned} & G_1 \in 3\text{-Colorability} \text{ und } G_2 \notin 3\text{-Colorability} \\ \iff & \delta(H_1) = 3 \text{ und } \delta(H_2) = 2 \\ \iff & \delta(H) = \delta(H_1) + \delta(H_2) = 5 \\ \iff & f(\langle G_1, G_2 \rangle) = H \in \text{Exact-5-DNP}. \end{aligned}$$

Aus Lemma 5.11 mit $k = 1$ folgt nun, dass Exact-5-DNP DP-vollständig ist, was Korollar 5.13 unmittelbar beweist.

Um den allgemeinen Fall zu beweisen, sei $k \geq 1$ fest gewählt. Gegeben sind die Graphen H_1, H_2, \dots, H_{2k} , die aus den Graphen G_1, G_2, \dots, G_{2k} konstruiert wurden. Verallgemeinere die oben angegebene Konstruktion des Graphen H wie folgt.

Für jede feste Folge T_1, T_2, \dots, T_{2k} von Dreiecken, wobei T_i zu H_i gehört, sind $6k$ neue Knoten a_1, a_2, \dots, a_{6k} hinzuzufügen. Dabei werden für jedes i mit $1 \leq i \leq 2k$ die drei neuen Knoten $a_{1+3(i-1)}, a_{2+3(i-1)}$ und a_{3i} mit dem Dreieck T_i assoziiert. So wie T_1 und T_2 mittels der Knoten a_1, a_2 und a_3 in Abbildung 5.3 verbunden sind, verbinde nun T_i , für jedes i mit $1 \leq i \leq 2k$, mit jedem T_j , wobei $1 \leq j \leq 2k$ und $i \neq j$, mittels desselben Teilgraphen, der aus den drei neuen, mit T_i assoziierten Knoten $a_{1+3(i-1)}, a_{2+3(i-1)}$ und a_{3i} besteht. Dann ist der Grad von a_i genau $6k - 1$ für jedes i , also ist $\delta(H) \leq 6k$. Ein Argument analog zu dem in der obigen Fallunterscheidung zeigt dann $\delta(H) = \sum_{j=1}^{2k} \delta(H_j)$. Es folgt:

$$\begin{aligned} & \|\{i \mid G_i \in 3\text{-Colorability}\}\| \text{ ist ungerade} \\ \iff & (\exists i : 1 \leq i \leq k) [\chi(G_1) = \dots = \chi(G_{2i-1}) = 3 \text{ und} \\ & \quad \chi(G_{2i}) = \dots = \chi(G_{2k}) = 4] \end{aligned}$$

$$\iff (\exists i : 1 \leq i \leq k) [\delta(H_1) = \dots = \delta(H_{2i-1}) = 3 \text{ und} \\ \quad \delta(H_{2i}) = \dots = \delta(H_{2k}) = 2]$$

$$\iff (\exists i : 1 \leq i \leq k) \left[\delta(H) = \sum_{j=1}^{2k} \delta(H_j) = 3(2i-1) + 2(2k-2i+1) \right]$$

$$\iff (\exists i : 1 \leq i \leq k) [\delta(H) = 4k + 2i - 1]$$

$$\iff \delta(H) \in \{4k+1, 4k+3, \dots, 6k-1\}$$

$$\iff f(\langle G_1, G_2, \dots, G_{2k} \rangle) = H \in \text{Exact-}M_k\text{-DNP}.$$

Die Funktion f erfüllt also (5.3). Nach Lemma 5.11 ist Exact- M_k -DNP ein $\text{BH}_{2k}(\text{NP})$ -hartes Problem und somit $\text{BH}_{2k}(\text{NP})$ -vollständig. \square

Im Gegensatz zu Korollar 5.13, gemäß welchem Exact-5-DNP ein DP-vollständiges Problem ist, liegt Exact-2-DNP in coNP; siehe Übung 5.8(a). Nach Satz 5.10 kann Exact-2-DNP also nicht DP-vollständig sein, außer die boolesche Hierarchie über NP würde kollabieren; siehe Übung 5.8(c). Die genaue Komplexität der Probleme Exact-3-DNP und Exact-4-DNP ist derzeit nicht bekannt: Sie sind beide coNP-hart und in DP enthalten, jedoch ist nicht bekannt, ob sie für eine dieser beiden Klassen vollständig sind.

Die zweite Anwendung von Lemma 5.11 betrifft das exakte Färbbarkeitsproblem. Zur Erinnerung: $\chi(G)$ bezeichnet die chromatische Zahl des Graphen G , d.h., die kleinste Zahl von Farben, die nötig sind, um die Knoten von G so zu färben, dass keine zwei benachbarten Knoten dieselbe Farbe erhalten. Nach Satz 3.56 aus Abschnitt 3.5.3 ist es NP-vollständig zu bestimmen, ob $\chi(G) \leq 3$ für einen gegebenen Graphen G gilt.

Definition 5.14 (Exaktes Färbbarkeitsproblem). Definiere für jede feste positive ganze Zahl i das exakte Färbbarkeitsproblem durch

$$\text{Exact-}i\text{-Colorability} = \{G \mid G \text{ ist ein Graph und } \chi(G) = i\}.$$

Definition 5.15 (Exaktes Färbbarkeitsproblem (allgemeine Version)). Es sei $M_k \subseteq \mathbb{N}$ eine Menge von k ganzen Zahlen. Definiere die allgemeine Version des exakten Färbbarkeitsproblems durch

$$\text{Exact-}M_k\text{-Colorability} = \{G \mid G \text{ ist ein Graph und } \chi(G) \in M_k\}.$$

Insbesondere schreiben wir $\text{Exact-}i\text{-Colorability} = \{G \mid \chi(G) = i\}$ für jede ein-elementige Menge $M_1 = \{i\}$.

Sei $M_k = \{6k + 1, 6k + 3, \dots, 8k - 1\}$. Eine einfache Anwendung von Wagners Technik zeigt, dass $\text{Exact-}M_k\text{-Colorability}$ für jedes $k \geq 1$ $\text{BH}_{2k}(\text{NP})$ -vollständig ist. Folglich ist für den Spezialfall $k = 1$ $\text{Exact-7-Colorability}$ ein DP-vollständiges Problem; siehe Übung 5.11. Im Gegensatz dazu zeigt die folgende Behauptung, dass für den Spezialfall $k = 1$ das Problem $\text{Exact-3-Colorability}$ in NP liegt. Nach Satz 5.10 kann $\text{Exact-3-Colorability}$ also nicht DP-vollständig sein, außer die boolesche Hierarchie über NP würde auf ihre erste Stufe kollabieren.

Behauptung 5.16. Sei $k \geq 1$ fest gewählt, und sei M_k eine beliebige Menge von k positiven ganzen Zahlen inklusive 3. Dann ist $\text{Exact-}M_k\text{-Colorability}$ in $\text{BH}_{2k-1}(\text{NP})$ und somit nicht $\text{BH}_{2k}(\text{NP})$ -vollständig, außer wenn die boolesche Hierarchie über NP kollabiert.

Beweis. Siehe Übung 5.12. □

Wieder gibt es eine Lücke für den präzisen Schwellwert i (nämlich zwischen $i = 4$ und $i = 7$), für den das Problem $\text{Exact-}i\text{-Colorability}$ von NP zur DP-Vollständigkeit springt. Um diese Lücke zu schließen, werden wir in Korollar 5.21

unten zeigen, dass die kleinste Zahl von Farben, die man zum Nachweis der DP-Vollständigkeit braucht, tatsächlich genau vier ist: Exact-4-Colorability ist DP-vollständig. Dieses Resultat ergibt sich aus der allgemeineren Aussage über die Härte des allgemeinen exakten Färbbarkeitsproblems für höhere Stufen der booleschen Hierarchie, siehe Satz 5.20 unten.

Um unser Ziel zu erreichen, wenden wir wieder Lemma 5.11 an, und wir benutzen zwei verschiedene Reduktionen von 3-SAT auf 3-Colorability mit bestimmten nützlichen Eigenschaften, die in den folgenden beiden Lemmata festgehalten werden. Die erste Reduktion ist die Standardreduktion von 3-SAT auf 3-Colorability, die im Beweis von Satz 3.56 angegeben wurde. Die zweite Reduktion geht auf Guruswami und Khanna [GK04, GK00] zurück.² Intuitiv sagt ihr Resultat, dass es „NP-hart ist, einen 3-färbaren Graphen mit höchstens vier Farben zu färben“, und zwar im Sinne von (5.7) und (5.8) in Lemma 5.18.

Lemma 5.17. *Es gibt eine Funktion $\sigma \in \text{FP}$, die 3-SAT auf 3-Colorability \leq_m^p -reduziert und die folgenden beiden Eigenschaften hat:*

$$\varphi \in \text{3-SAT} \implies \chi(\sigma(\varphi)) = 3; \quad (5.5)$$

$$\varphi \notin \text{3-SAT} \implies \chi(\sigma(\varphi)) = 4. \quad (5.6)$$

Lemma 5.18 (Guruswami und Khanna; cf. Beweis von Theorem 1.1 in [GK04]). *Es gibt eine Funktion $\rho \in \text{FP}$, die 3-SAT auf 3-Colorability \leq_m^p -reduziert und die folgenden beiden Eigenschaften hat:*

$$\varphi \in \text{3-SAT} \implies \chi(\rho(\varphi)) = 3; \quad (5.7)$$

$$\varphi \notin \text{3-SAT} \implies \chi(\rho(\varphi)) = 5. \quad (5.8)$$

Beweisskizze. Die Guruswami–Khanna-Reduktion, nennen wir sie ρ , ist die Komposition zweier aufeinander folgender Reduktionen: zuerst eine Reduktion von 3-SAT auf das Problem IS, und danach von IS auf 3-Colorability. Zur Erinnerung: Beim IS-Problem wird für einen gegebenen Graphen G und eine gegebene Zahl $m \geq 0$ gefragt, ob die Größe einer maximalen unabhängigen Menge von G (d.h., einer maximalen Teilmenge der Knotenmenge von G , in der keine zwei Knoten benachbart sind) mindestens m ist.

Wir beschränken uns auf eine grobe Skizze der sehr raffinierten Konstruktion von Guruswami und Khanna, welche u.a. baumartige Strukturen und verschiedene Arten von Teilgraphen zu ihrer Verknüpfung verwendet. Zunächst betrachten wir die Standardreduktion von 3-SAT auf IS, die im Beweis von Satz 3.54 angegeben ist (siehe Abbildung 3.3). Dort wird aus der gegebenen booleschen Formel φ mit m Klauseln ein Graph G konstruiert, der aus m Dreiecken besteht (d.h., aus m Cliques der Größe jeweils 3), so dass jedes Dreieck einer Klausel von φ entspricht und die Knoten von je zwei verschiedenen Dreiecken genau dann durch eine Kante verbunden sind, wenn sie in den entsprechenden Klauseln ein Literal und seine Negation repräsentieren.

² Ursprünglich war die Reduktion von Guruswami und Khanna nicht von Fragen motiviert, die die Härte des exakten Färbbarkeitsproblems betreffen, sondern von Fragen, die mit der Approximierbarkeit der chromatischen Zahl von Graphen zu tun haben.

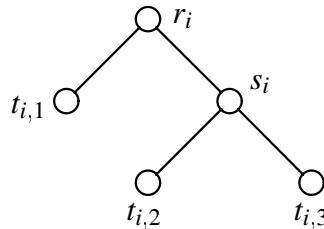


Abb. 5.4. Baumartige Struktur S_i in der Guruswami–Khanna-Reduktion

Bezeichne die m Dreiecke mit T_1, T_2, \dots, T_m . Einem jeden T_i in G entspricht eine baumartige Struktur S_i , die in Abbildung 5.4 dargestellt ist, wobei die drei „Blätter“ $t_{i,1}, t_{i,2}$ und $t_{i,3}$ den drei Eckknoten des Dreiecks T_i entsprechen. Jeder einzelne „Knoten“ der baumartigen Strukturen hat die Form des in Abbildung 5.5 gezeigten Grundgitters. Ein solches (3×3) -Gitter hat zusätzliche Kanten, so dass die Knoten in jeder Zeile und in jeder Spalte des Gitters eine 3-Clique induzieren. Die drei Knoten in der ersten Spalte eines solchen Grundgitters, welche als die „Grundknoten“ bezeichnet werden, werden eigentlich von sämtlichen Grundgittern in einer jeden baumartigen Struktur verwendet. Da die Grundknoten eine 3-Clique bilden, muss ihnen jede legale Färbung drei verschiedene Farben zuweisen, nennen wir diese Farben 1, 2 und 3.

Das Verbindungsmuster zwischen den fünf Grundgittern einer baumartigen Struktur sieht man in Abbildung 5.6, welche einen „Elternknoten“ und seine beiden „Kinder“ zeigt, die jeweils aus den neun Knoten des Grundgitters bestehen. Außerdem gibt es noch zwei weitere Dreiecke. Jeder Knoten in den Grundgittern und in den Dreiecken wird mit einem Zahlentripel markiert. Die einfache Regel ist, dass je zwei Knoten genau dann durch eine Kante verbunden werden, wenn ihre Markierungstripel in keiner Koordinate übereinstimmen. Abbildung 5.6 zeigt dieses Muster für die „Knoten“ $r_i, t_{i,1}$ und s_i ; ein analoges Muster gilt für die „Knoten“ $s_i, t_{i,2}$ und $t_{i,3}$.

Bevor wir beschreiben, wie die baumartigen Strukturen S_i miteinander verbunden sind, erklären wir kurz die intuitive Idee dieser Konstruktion. Jede Färbung von S_i

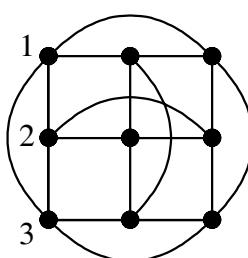


Abb. 5.5. Grundgitter in der Guruswami–Khanna-Reduktion

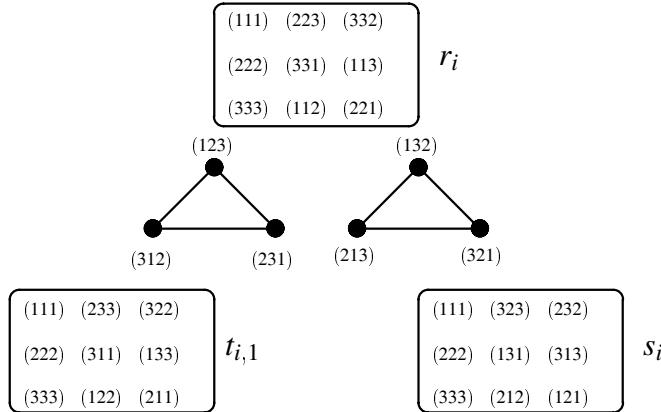


Abb. 5.6. Verbindungsmuster zwischen den Grundgittern einer baumartigen Struktur

wählt bestimmte „Knoten“ von S_i aus. Ein „Knoten“ in einem S_i wird genau dann gewählt, wenn wenigstens eine der drei Zeilen in seinem Grundgitter Farben erhält, die einer geraden Permutation von $\{1, 2, 3\}$ entsprechen, d.h., wenn die erste Zeile die Farben 1, 2, 3 von links nach rechts hat oder die zweite Zeile die Farben 2, 3, 1 oder die dritte Zeile die Farben 3, 1, 2.

Unser Ziel ist es, zu garantieren, dass jede legale 4-Färbung von S_i wenigstens eines der drei „Blätter“ wählt, $t_{i,1}$, $t_{i,2}$ oder $t_{i,3}$. Die Idee ist, die Wahl der Wurzel r_i zu erzwingen und anschließend zu zeigen, dass wenn ein innerer „Knoten“ gewählt wird, r_i oder s_i , dann muss auch mindestens eines seiner beiden Kinder gewählt werden. Diese Eigenschaft wird in dem folgenden Schlüssellemma ausgedrückt. Der detaillierte Beweis von Lemma 5.19 wird dem Leser als Problem 5.2 überlassen.

Lemma 5.19. *Jede legale 4-Färbung von S_i wählt wenigstens eines der „Blätter“ von S_i aus, also $t_{i,1}$, $t_{i,2}$ oder $t_{i,3}$.*

Schließlich spezifizieren wir noch, wie man die „Blätter“ unterschiedlicher baumartiger Strukturen S_i und S_j , $i \neq j$, miteinander verbindet. Für jedes Paar von Knoten, $t_{i,k}$ und $t_{j,\ell}$, die benachbarten Knoten im Ausgangsgraphen G entsprechen, werden geeignete Teilgraphen eingefügt, die verhindern sollen, dass diese beiden „Blätter“ gleichzeitig gewählt werden. (Das ist nötig, da andernfalls eine 4-Färbung des konstruierten Graphen implizieren würde, dass der Ausgangsgraph G eine unabhängige Menge der Größe m hat, im Widerspruch zu unserem Ziel, (5.8) zu zeigen.)

Zwei Typen von Teilgraphen werden zur Verbindung der „Blätter“ in unterschiedlichen S_i und S_j , $i \neq j$, benutzt. Der erste Typ ist in Abbildung 5.7 zu sehen. Sein Zweck ist, zu verhindern, dass beide „Blätter“ wegen *derselben* Zeile gleichzeitig gewählt werden; etwa weil in einer 4-Färbung beide „Blätter“ eine dritte Zeile haben, die mit 3, 1, 2 gefärbt ist. Der zweite Teilgraphentyp ist in Abbildung 5.8 dargestellt. Sein Zweck ist zu verhindern, dass beide „Blätter“ wegen *verschiedener* Zeilen gleichzeitig gewählt werden; etwa weil in einer 4-Färbung die dritte Zeile von $t_{i,k}$ mit

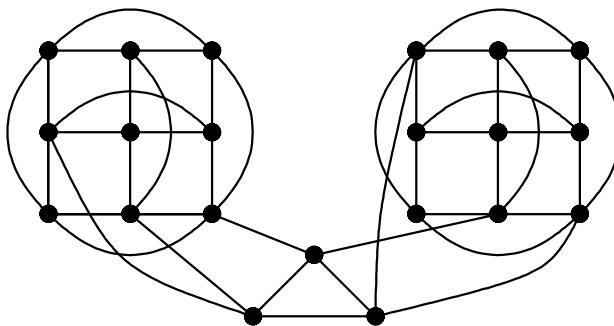


Abb. 5.7. Verbindung der „Blätter“ wegen *derselben* Zeile

3, 1, 2 und die erste Zeile von $t_{j,\ell}$ mit 1, 2, 3 gefärbt ist. Man beachte jedoch, dass es eine legale 3-Färbung des Graphen gibt, wann immer die dritte Zeile von $t_{i,k}$ mit 3, 2, 1 und die erste Zeile von $t_{j,\ell}$ mit 1, 3, 2 gefärbt ist, d.h., wann immer $t_{i,k}$ und $t_{j,\ell}$ nicht wegen dieser beiden verschiedenen Zeilen gewählt werden.

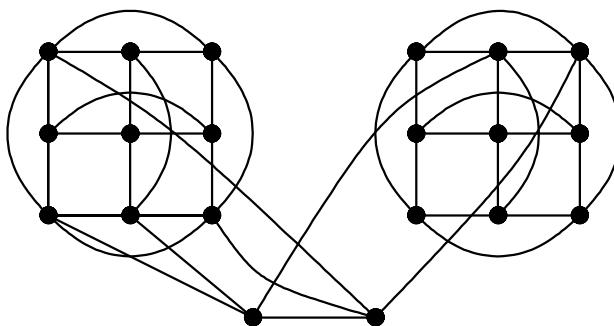


Abb. 5.8. Verbindung der „Blätter“ wegen *verschiedener* Zeilen

Damit ist die Reduktion ρ vollständig beschrieben, die die Formel φ über den Graphen G in den Graphen $H = \rho(\varphi)$ überführt. Wir verzichten auf die Erläuterung der Details im cleveren Korrektheitsbeweis von Guruswami und Khanna. Eine grobe Skizze der Idee sollte aus den obigen Erklärungen klar sein, und die genauen Details seien dem interessierten Leser als Problem 5.2 überlassen.

Die obige Konstruktion garantiert die folgenden Implikationen:

- $\varphi \in 3\text{-SAT} \implies \chi(\rho(\varphi)) = 3$, und
- $\varphi \notin 3\text{-SAT} \implies \chi(\rho(\varphi)) = 5$.

Anders gesagt hat der Graph H niemals eine chromatische Zahl von genau vier, egal, ob φ erfüllbar ist oder nicht. Somit sind (5.7) und (5.8) bewiesen. \square

Satz 5.20. Für ein festes $k \geq 1$ sei $M_k = \{3k + 1, 3k + 3, \dots, 5k - 1\}$. Dann ist Exact- M_k -Colorability ein $\text{BH}_{2k}(\text{NP})$ -vollständiges Problem.

Für $k = 1$ hat Satz 5.20 insbesondere das folgende Korollar. Sowohl der Satz als auch sein Korollar können mit Wagners Technik bewiesen werden, unter Verwendung von Lemma 5.17 und Lemma 5.18.

Korollar 5.21. Exact-4-Colorability ist DP-vollständig.

Beweis von Satz 5.20. Sei $k \geq 1$ beliebig, aber fest gewählt. Wende Lemma 5.11 an, wobei 3-SAT das NP-vollständige Problem A aus diesem Lemma ist. Das Problem B aus diesem Lemma ist Exact- M_k -Colorability mit $M_k = \{3k + 1, 3k + 3, \dots, 5k - 1\}$.

Sei σ die Standardreduktion von 3-SAT auf 3-Colorability gemäß Lemma 5.17, und sei ρ die Guruswami–Khanna-Reduktion gemäß Lemma 5.18, die ebenfalls 3-SAT auf 3-Colorability reduziert.

Definiere die folgende Operation auf Graphen. Für zwei gegebene knotendisjunkte Graphen A und B ist $A \bowtie B$ der Graph mit der folgenden Knoten- bzw. Kan tenmenge:

$$\begin{aligned} V(A \bowtie B) &= V(A) \cup V(B); \\ E(A \bowtie B) &= E(A) \cup E(B) \cup \{\{a, b\} \mid a \in V(A) \text{ und } b \in V(B)\}. \end{aligned}$$

Offenbar gilt $\chi(A \bowtie B) = \chi(A) + \chi(B)$, und \bowtie ist assoziativ; siehe Übung 5.13.

Gegeben seien $2k$ boolesche Formeln $\varphi_1, \varphi_2, \dots, \varphi_{2k}$, so dass $\varphi_j \in 3\text{-SAT}$ aus $\varphi_{j+1} \in 3\text{-SAT}$ für jedes j mit $1 \leq j < 2k$ folgt. Definiere $2k$ Graphen H_1, H_2, \dots, H_{2k} wie folgt: Es seien $H_{2i-1} = \rho(\varphi_{2i-1})$ und $H_{2i} = \sigma(\varphi_{2i})$ für jedes i mit $1 \leq i \leq k$. Nach (5.5), (5.6), (5.7) und (5.8) folgt aus den Lemmata 5.17 und 5.18:

$$\chi(H_j) = \begin{cases} 3 & \text{falls } 1 \leq j \leq 2k \text{ und } \varphi_j \in 3\text{-SAT} \\ 4 & \text{falls } j = 2i \text{ für ein } i \in \{1, 2, \dots, k\} \text{ und } \varphi_j \notin 3\text{-SAT} \\ 5 & \text{falls } j = 2i - 1 \text{ für ein } i \in \{1, 2, \dots, k\} \text{ und } \varphi_j \notin 3\text{-SAT}. \end{cases} \quad (5.9)$$

Definiere den Graphen G_i , für jedes i mit $1 \leq i \leq k$, als die disjunkte Vereinigung der Graphen H_{2i-1} und H_{2i} . Somit gilt $\chi(G_i) = \max\{\chi(H_{2i-1}), \chi(H_{2i})\}$, für jedes i , $1 \leq i \leq k$. Unsere Reduktion f ist vollständig beschrieben, wenn wir nun noch definieren:

$$f(\langle \varphi_1, \varphi_2, \dots, \varphi_{2k} \rangle) = G,$$

wobei der Graph G sich als $G = G_1 \bowtie G_2 \bowtie \dots \bowtie G_k$ ergibt. Folglich gilt:

$$\chi(G) = \sum_{i=1}^k \chi(G_i) = \sum_{i=1}^k \max\{\chi(H_{2i-1}), \chi(H_{2i})\}. \quad (5.10)$$

Es folgt aus dieser Konstruktion, dass gilt:

$$\begin{aligned}
& \|\{i \mid \varphi_i \in 3\text{-SAT}\}\| \text{ ist ungerade} \\
\iff & (\exists i : 1 \leq i \leq k) [\varphi_1, \dots, \varphi_{2i-1} \in 3\text{-SAT} \text{ und } \varphi_{2i}, \dots, \varphi_{2k} \notin 3\text{-SAT}] \\
\stackrel{(5.9), (5.10)}{\iff} & (\exists i : 1 \leq i \leq k) \left[\begin{array}{l} \sum_{j=1}^k \chi(G_j) = 3(i-1) + 4 + 5(k-i) \\ = 5k - 2i + 1 \end{array} \right] \\
\stackrel{(5.10)}{\iff} & \chi(G) \in M_k = \{3k+1, 3k+3, \dots, 5k-1\} \\
\iff & f(\langle \varphi_1, \varphi_2, \dots, \varphi_{2k} \rangle) = G \in \text{Exact-}M_k\text{-Colorability}.
\end{aligned}$$

Somit ist (5.3) erfüllt. Nach Lemma 5.11 ist $\text{Exact-}M_k\text{-Colorability}$ also $\text{BH}_{2k}(\text{NP})$ -vollständig, wie gewünscht. \square

Im Abschnitt 5.9 werden weitere Anwendungen der Technik von Wagner zum Nachweis unterer Schranken (d.h. der $\text{BH}_k(\text{NP})$ -Härte) für die Stufen der booleschen Hierarchie über NP erwähnt.

5.2 Die Polynomialzeit-Hierarchie

Genau wie die boolesche Hierarchie ist die Polynomialzeit-Hierarchie eine auf NP beruhende Hierarchie von Komplexitätsklassen. Die Stufen dieser Hierarchie können auf zwei äquivalente Weisen charakterisiert werden: erstens durch alternierende längenbeschränkte \exists - und \forall -Quantoren; zweitens durch übereinander gestapelte NP-Orakelmaschinen, die jeweils auf NP-Orakel zugreifen.

Wir beginnen mit der Quantorendarstellung für die erste Stufe der Hierarchie, also für NP selbst. NP-Algorithmen bestehen typischerweise aus zwei Phasen: einer nichtdeterministischen Ratephase, in der potenzielle Problemlösungen geraten werden, gefolgt von einer deterministischen Überprüfungsphase, in der die Korrektheit der geratenen Lösungen verifiziert wird. Beide Phasen sind polynomialzeitbeschränkt.

Problemlösungen \mathbf{w} , die in deterministischer Polynomialzeit verifiziert werden können, nennt man auch „Zeugen“ oder „Zertifikate“, denn sie bezeugen (oder zertifizieren) die Zugehörigkeit einer gegebenen Probleminstanz zu einem gegebenen NP-Problem. Zur Illustration betrachten wir die folgenden Beispiele von NP-Problemen.

Beispiel 5.22 (Zeugen für positive Instanzen von NP-Problemen).

- Betrachte das NP-vollständige Problem SOS. Die auf der Hand liegende NP-Maschine für SOS geht so vor: Gegeben eine Instanz $\langle s_1, s_2, \dots, s_n, T \rangle$, rate nichtdeterministisch eine Lösung $\mathbf{w} \in \{0, 1\}^n$, $\mathbf{w} = (w_1, w_2, \dots, w_n)$, berechne für jede geratene Lösung \mathbf{w} deterministisch die Summe $\sum_{i=1}^n w_i s_i$ und akzeptiere die Eingabe genau dann, wenn $\sum_{i=1}^n w_i s_i = T$.
- Betrachte das NP-Problem GI. Die auf der Hand liegende NP-Maschine für GI geht so vor: Gegeben eine Instanz $\langle G, H \rangle$, wobei G und H ungerichtete Graphen

mit jeweils n Knoten sind, rate nichtdeterministisch eine Lösung $\pi \in \mathfrak{S}_n$ (d.h., eine Permutation der Knoten von G) und überprüfe deterministisch für jede geratene Lösung π , ob sie ein Isomorphismus (d.h., eine kantenerhaltende Bijektion) zwischen den Knoten von G und H ist oder nicht.

Kurz erwähnt seien noch weitere Beispiele: Lösungen des Cliqueproblems für eine gegebene Instanz $\langle G, k \rangle$ sind (geeignet codierte) Cliques der Größe k des Graphen G ; Lösungen des Erfüllbarkeitsproblems sind erfüllende Wahrheitswertbelegungen der gegebenen Formel; Lösungen des 3-DM-Problems sind tripartite Matchings der gegebenen 3-DM-Instanz; und so weiter.

Polynomialzeit-beschränkte nichtdeterministische Ratephasen entsprechen somit der polynomiell längenbeschränkten existenziellen Quantifizierung. Alle NP-Probleme sind von dieser Form: A ist genau dann in NP (mittels einer NP-Maschine M), wenn A aus genau den Eingabewörtern x besteht, für die ein polynomiell längenbeschränkter, in Polynomialzeit überprüfbarer Zeuge w (bezüglich M) existiert. Anders gesagt sind Zeugen nichts anderes als akzeptierende Berechnungspfade von $M(x)$, geeignet als Binärwörter der Länge $p(|x|)$ für ein $p \in \text{IPol}$ codiert. Das i -te Bit eines Zeugen w entspricht der i -ten nichtdeterministischen Verzweigung von $M(x)$ entlang des Berechnungspfades w .

Definition 5.23 (Zeugenmenge). Sei $A \in \text{NP}$, und sei M eine NP-Maschine, die A in der Zeit $p \in \text{IPol}$ akzeptiert. Definiere für jedes x der Länge n die Zeugenmenge für „ $x \in A$ “ bezüglich M durch

$$\text{Wit}_M(x) = \{w \in \{0,1\}^{p(n)} \mid w \text{ ist ein akzeptierender Berechnungspfad von } M(x)\}.$$

Offenbar ist x genau dann in A , wenn $\text{Wit}_M(x)$ nicht leer ist.

Satz 5.24 ist das Komplexitätstheoretische Analogon des Projektionssatzes aus der Berechenbarkeitstheorie, welcher sagt, dass eine jede gegebene Menge genau dann rekursiv aufzählbar ist, wenn sie die Projektion einer entscheidbaren Menge ist; siehe die zweite Aussage von Satz 2.20 in Abschnitt 2.2. Satz 5.24 kann mit den oben gemachten Bemerkungen leicht bewiesen werden. Der formale Beweis wird dem Leser als Übung 5.16 überlassen.

Satz 5.24. Eine Menge A ist genau dann in NP, wenn es eine Menge B in P und ein Polynom p gibt, so dass für alle $x \in \Sigma^*$ gilt:

$$x \in A \iff (\exists w) [|w| \leq p(|x|) \text{ und } \langle x, w \rangle \in B]. \quad (5.11)$$

Für polynomiell längenbeschränkte Quantoren verwenden wir die folgende Bezeichnung.

Definition 5.25 (Polynomiell längenbeschränkter Quantor). Für jedes Prädikat B , für jedes Polynom p und für jedes Wort x sei

$$\begin{aligned} (\exists^p y) [B(x, y)] &\iff (\exists y) [|y| \leq p(|x|) \text{ und } B(x, y)]; \\ (\forall^p y) [B(x, y)] &\iff (\forall y) [|y| \leq p(|x|) \text{ impliziert } B(x, y)]. \end{aligned}$$

Nun wenden wir uns dem oben erwähnten Zugang zur Polynomialzeit-Hierarchie über NP-Orakelmaschinen zu. Orakel-Turingmaschinen wurden formal in Definition 2.22 von Abschnitt 2.2 eingeführt. Insbesondere sind sie ein geeignetes Modell zur Realisierung bestimmter Suchtechniken, etwa der *Binärsuche* oder der *Präfixsuche*, wie das folgende Beispiel illustriert.

Beispiel 5.26 (Präfixsuche mit einer Orakel-Turingmaschine). Das Graphisomorphie-Problem GI wurde in Definition 2.49 von Abschnitt 2.4 eingeführt. Seien G und H zwei gegebene Graphen mit jeweils $n \geq 1$ Knoten. Die Menge $\text{Iso}(G, H)$ der Isomorphismen zwischen G und H enthält alle Lösungen (oder Zeugen) von „ $\langle G, H \rangle \in \text{GI}$ “ bezüglich der Standard-NPTM für GI. Es gilt:

$$\text{Iso}(G, H) \neq \emptyset \iff \langle G, H \rangle \in \text{GI}. \quad (5.12)$$

Wir möchten den lexikographisch kleinsten Graphisomorphismus in $\text{Iso}(G, H)$ finden, falls $\langle G, H \rangle \in \text{GI}$; andernfalls soll „ $\langle G, H \rangle \notin \text{GI}$ “ dadurch angezeigt werden, dass das leere Wort ε zurückgegeben wird. Das heißt, wir wollen die durch

$$f(G, H) = \begin{cases} \min\{\pi \mid \pi \in \text{Iso}(G, H)\} & \text{if } \langle G, H \rangle \in \text{GI} \\ \varepsilon & \text{if } \langle G, H \rangle \notin \text{GI}. \end{cases}$$

definierte Funktion f berechnen. Dabei wird das Minimum bezüglich der lexikographischen Ordnung auf \mathfrak{S}_n genommen, die folgendermaßen definiert ist. Eine Permutation $\pi \in \mathfrak{S}_n$ wird als ein Wort $\pi(1)\pi(2)\cdots\pi(n)$ der Länge n über dem Alphabet $[n] = \{1, 2, \dots, n\}$ aufgefasst, und für $\sigma, \tau \in \mathfrak{S}_n$ schreibt man $\sigma < \tau$ genau dann, wenn es ein $j \in [n]$ gibt, so dass $\sigma(i) = \tau(i)$ für alle $i < j$ und $\sigma(j) < \tau(j)$ gilt. Sind die Permutationen σ und τ beispielsweise durch $\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 1 & 5 & 2 \end{pmatrix}$ und $\tau = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 2 & 1 & 5 \end{pmatrix}$ gegeben, so ist $\sigma = 34152 < 34215 = \tau$, denn sie stimmen in den ersten beiden Positionen überein und unterscheiden sich in der dritten, wobei $\sigma(3) = 1 < 2 = \tau(3)$.

Streicht man Paare $(i, \sigma(i))$ aus einer Permutation $\sigma \in \mathfrak{S}_n$, so erhält man eine *partielle Permutation*, die ebenfalls als ein Wort über $[n] \cup \{\ast\}$ aufgefasst werden kann, wobei \ast eine undefinierte Position anzeigt. Ein *Präfix der Länge k von $\sigma \in \mathfrak{S}_n$* , $k \leq n$, ist eine partielle Permutation von σ , die jedes Paar $(i, \sigma(i))$ mit $i \leq k$ enthält, aber keines der Paare $(i, \sigma(i))$ mit $i > k$. Ist insbesondere $k = 0$, dann ist das leere Wort ε der (eindeutige) Präfix der Länge 0 von σ , und ist $k = n$, dann ist die totale Permutation σ der (eindeutige) Präfix der Länge n von sich selbst. Ist beispielsweise $\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 1 & 5 & 2 \end{pmatrix}$, so ist $\tau = \begin{pmatrix} 1 & 3 & 5 \\ 3 & 1 & 2 \end{pmatrix}$ eine partielle Permutation von σ , und $\pi = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 4 & 1 \end{pmatrix}$ ist ein Präfix der Länge 3 von σ . Als ein Wort über $[n] \cup \{\ast\}$ hat die partielle Permutation τ die Gestalt $\tau = 3\ast1\ast2$. Für Präfixe wie $\pi = 341\ast\ast = 341$ können die Platzhalter \ast auch weggelassen werden.

Ist π ein Präfix der Länge $k < n$ von $\sigma \in \mathfrak{S}_n$ und ist $w = i_1 i_2 \cdots i_{|w|}$ ein Wort über $[n]$ der Länge $|w| \leq n - k$, wobei keines der i_j in π vorkommt, dann bezeichnet πw die partielle Permutation, die π um die Paare

$$(k+1, i_1), (k+2, i_2), \dots, (k+|w|, i_{|w|})$$

erweitert. Ist zusätzlich $\sigma(k+j) = i_j$ für $1 \leq j \leq |w|$, so ist πw auch ein Präfix von σ . Ist beispielsweise $\pi = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 4 & 1 \end{pmatrix}$ ein Prefix von $\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 1 & 5 & 2 \end{pmatrix}$, so wird π durch ein jedes

```

 $N^{\text{Pre-Iso}}(G, H) \{$ 
    if ( $\langle G, H, \varepsilon \rangle \notin \text{Pre-Iso}$ ) return  $\varepsilon$ ;
    else {
         $\pi := \varepsilon$ ;
         $j := 0$ ;
        while ( $j < n$ ) { // G und H haben beide  $n$  Knoten
             $i := 1$ ;
            while ( $\langle G, H, \pi i \rangle \notin \text{Pre-Iso}$ ) {  $i := i + 1$ ; }
             $\pi := \pi i$ ;
             $j := j + 1$ ;
        }
    }
    return  $\pi$ ;
}

```

Abb. 5.9. Präfixsuche nach dem kleinsten Graphisomorphismus in $\text{Iso}(G, H)$

der Wörter $w_1 = 2$, $w_2 = 5$, $w_3 = 25$ und $w_4 = 52$ erweitert, aber nur $\pi w_2 = 3415$ und $\pi w_4 = 34152$ sind Präfixe von σ , die π erweitern.

Für je zwei Graphen G und H sei die Menge der Präfixe der Isomorphismen in $\text{Iso}(G, H)$ definiert durch

$$\text{Pre-Iso} = \{ \langle G, H, \pi \rangle \mid (\exists w \in [n]^*) [w = i_1 i_2 \cdots i_{n-|\pi|} \text{ und } \pi w \in \text{Iso}(G, H)] \}.$$

Zu beachten ist, dass für $n \geq 1$ das leere Wort ε keine Permutation in S_n codiert. $\text{Iso}(G, H) = \emptyset$ gilt genau dann, wenn $\langle G, H, \varepsilon \rangle \notin \text{Pre-Iso}$, was nach (5.12) genau dann der Fall ist, wenn $\langle G, H \rangle \notin \text{GI}$. Unter Verwendung von Pre-Iso als Orakelmenge berechnet die DPOTM N in Abbildung 5.9 die Funktion f durch Präfixsuche. Folglich ist f in $\text{FP}^{\text{Pre-Iso}}$. Es ist nicht schwer zu zeigen, dass Pre-Iso eine Menge in NP ist; siehe Übung 5.17(a). Weil Pre-Iso in NP ist, ist f demnach in FP^{NP} ; siehe auch Übung 5.17(b) und Problem 5.3.

Der Algorithmus in Abbildung 5.9 beschreibt eine Turing-Reduktion auf die Orakelmenge Pre-Iso. Polynomialzeit-beschränkte Turing-Reduktionen können polynomialiell viele Fragen stellen, wohingegen polynomialzeit-beschränkte Many-one-Reduktionen lediglich eine Frage stellen dürfen. Darüber hinaus akzeptieren \leq_T^p -Reduktionen die gegebene Eingabe genau dann, wenn diese Frage positiv beantwortet wird, eine recht starke Einschränkung, unter der \leq_T^p -Reduktionen nicht leiden. Die Turing-Reduzierbarkeit ist daher viel flexibler als die Many-one-Reduzierbarkeit.

Definition 5.27 (Turing-Reduzierbarkeit und -Vollständigkeit). Sei $\Sigma = \{0, 1\}$ ein festes Alphabet, und seien A und B Mengen von Wörtern über Σ . \mathcal{C} sei eine beliebige Komplexitätsklasse.

1. Definiere die polynomialzeit-beschränkte Turing-Reduzierbarkeit, bezeichnet mit \leq_T^p , wie folgt: $A \leq_T^p B$ genau dann, wenn es eine deterministische polynomialzeit-beschränkte Orakel-Turingmaschine (kurz DPOTM) M gibt, so dass $A = L(M^B)$.

2. Definiere die nichtdeterministische polynomialzeit-beschränkte Turing-Reduzierbarkeit, bezeichnet mit \leq_T^{NP} , wie folgt: $A \leq_T^{\text{NP}} B$ genau dann, wenn es eine nichtdeterministische polynomialzeit-beschränkte Orakel-Turingmaschine (kurz **NPOTM**) M gibt, so dass $A = L(M^B)$.
3. Eine Menge B ist \leq_T^p -hart für \mathcal{C} , falls $A \leq_T^p B$ für jede Menge $A \in \mathcal{C}$ gilt.
4. Eine Menge B ist \leq_T^p -vollständig für \mathcal{C} , falls $B \leq_T^p$ -hart für \mathcal{C} ist und $B \in \mathcal{C}$.
5. \mathcal{C} heißt abgeschlossen unter der \leq_T^p -Reduzierbarkeit (kurz \leq_T^p -abgeschlossen), falls für je zwei Mengen A und B aus $A \leq_T^p B$ und $B \in \mathcal{C}$ folgt, dass $A \in \mathcal{C}$. Der Begriff der \leq_T^{NP} -Abgeschlossenheit von \mathcal{C} ist analog definiert. Die \leq_T^p -Hülle von \mathcal{C} und die \leq_T^{NP} -Hülle von \mathcal{C} sind jeweils wie folgt definiert:

$$\begin{aligned} P^{\mathcal{C}} &= \{A \mid (\exists B \in \mathcal{C}) [A \leq_T^p B]\}; \\ NP^{\mathcal{C}} &= \{A \mid (\exists B \in \mathcal{C}) [A \leq_T^{\text{NP}} B]\}. \end{aligned}$$

Die folgende Behauptung fasst einige grundlegende Eigenschaften der oben definierten Reduzierbarkeiten zusammen. Der Beweis von Behauptung 5.28 wird dem Leser als Übung 5.18 überlassen. Als Erklärung für die erste Aussage von Behauptung 5.28: Die Relationen \leq_T^p und \leq_T^{NP} können als Mengen von Paaren von Mengen aufgefasst werden, d.h., $\leq_T^p = \{(A, B) \mid A \leq_T^p B\}$ und $\leq_T^{\text{NP}} = \{(A, B) \mid A \leq_T^{\text{NP}} B\}$. Die letzte Aussage von Behauptung 5.28 gilt analog zu Satz 3.41. Sie sagt, dass NP auf L kollabiert, falls \leq_T^{NP} und \leq_m^{\log} auf NP übereinstimmen. Anders als \leq_m^{\log} , \leq_m^p und \leq_T^p ist \leq_T^{NP} beweisbar keine transitive Relation.

- Behauptung 5.28.**
1. $\leq_m^{\log} \subseteq \leq_m^p \subseteq \leq_T^p \subseteq \leq_T^{\text{NP}}$.
 2. Die Relation \leq_T^p ist reflexiv und transitiv, aber nicht antisymmetrisch.
 3. Die Relation \leq_T^{NP} ist reflexiv, aber weder antisymmetrisch noch transitiv.
 4. P und PSPACE sind \leq_T^p -abgeschlossen, d.h., $P^P = P$ und $P^{\text{PSPACE}} = \text{PSPACE}$.
 5. $NP^P = NP$ und $NP^{\text{PSPACE}} = \text{PSPACE}$.
 6. Sei $A \leq_T^p B$. Ist $A \leq_T^p$ -hart für eine Komplexitätsklasse \mathcal{C} , so auch B .
 7. Ist $L \neq NP$, so gibt es Mengen A und B in NP, so dass $A \leq_T^{\text{NP}} B$, aber $A \not\leq_m^{\log} B$.

Nach Behauptung 5.28 gilt $P^P = P$ und $NP^P = NP$. Somit kann die Berechnungskraft weder von P noch von NP durch Orakelmengen in P erhöht werden. Was ist mit Orakelmengen in NP? Ist NP gleich P^{NP} oder sogar gleich NP^{NP} ? Beide Gleichheiten werden für höchst unwahrscheinlich gehalten. Das heißt, NP ist höchstwahrscheinlich weder \leq_T^p -abgeschlossen noch \leq_T^{NP} -abgeschlossen. Jedoch ist NP abgeschlossen unter polynomialzeit-beschränkten positiven Turing-Reduktionen; siehe Übung 5.19.

Beim naiven Versuch, $P^{NP} = NP$ zu beweisen, würde man die Orakelfragen an das NP-Orakel B direkt simulieren. Angenommen, die Eingabe x wird bei der P^B -Berechnung abgelehnt und eine Orakelfrage, etwa q_i , gehört zu B . Dann ist „ja“ die korrekte Antwort auf die Orakelfrage „ $q_i \in B$ “? Die Simulation der NP-Maschine M für B bei Eingabe q_i kann sowohl akzeptierende Berechnungspfade (welche korrekt sind, da q_i in B ist) und (inkorrekte) ablehnende Berechnungspfade liefern. Die ablehnenden Pfade „wissen“ jedoch nicht, dass q_i in B ist. Ebensowenig „wissen“ sie, dass sie selbst eigentlich inkorrekt sind, denn sie können nicht „sehen“, was

auf den anderen Pfaden von $M(q_i)$ geschieht und ob es etwa einen akzeptierenden Pfad gibt. Daher kann es passieren, dass solch ein inkorrekt ablehnender Pfad von $M(q_i)$ fälschlicherweise zu einer Akzeptierung der Eingabe x führt, was zeigt, dass der naive Versuch, $P^{\text{NP}} = \text{NP}$ zu beweisen, fehlschlägt. Tatsächlich ist NP höchstwahrscheinlich verschieden von P^{NP} und von NP^{NP} . Diese Beobachtung motiviert die folgende Definition, mit der die Polynomialzeit-Hierarchie eingeführt wird.

Definition 5.29 (Polynomialzeit-Hierarchie). Die Polynomialzeit-Hierarchie ist induktiv wie folgt definiert:

$$\begin{aligned}\Delta_0^P &= \Sigma_0^P = \Pi_0^P = P; \\ \Delta_{i+1}^P &= P^{\Sigma_i^P}, \quad \Sigma_{i+1}^P = \text{NP}^{\Sigma_i^P}, \text{ und } \Pi_{i+1}^P = \text{co}\Sigma_{i+1}^P \quad \text{für } i \geq 0; \\ \text{PH} &= \bigcup_{k \geq 0} \Sigma_k^P.\end{aligned}$$

Insbesondere gilt $\Delta_1^P = P^{\Sigma_0^P} = P^P = P$ und $\Sigma_1^P = \text{NP}^{\Sigma_0^P} = \text{NP}^P = \text{NP}$ und $\Pi_1^P = \text{co}\Sigma_1^P = \text{coNP}$.

Satz 5.30. 1. Für jedes $i \geq 0$ ist $\Sigma_i^P \cup \Pi_i^P \subseteq \Delta_{i+1}^P \subseteq \Sigma_{i+1}^P \cap \Pi_{i+1}^P$.

2. $\text{PH} \subseteq \text{PSPACE}$.

3. Jede der Klassen Δ_i^P , Σ_i^P , Π_i^P und PH ist \leq_m^p -abgeschlossen. Die Δ_i^P -Stufen der Polynomialzeit-Hierarchie sind sogar unter \leq_T^p -Reduktionen abgeschlossen.

Beweis. 1. Für jede Klasse \mathcal{C} gilt $\mathcal{C} \subseteq P^{\mathcal{C}}$, da \leq_T^p nach Behauptung 5.28 reflexiv ist: Ist A in \mathcal{C} , so ist $A = L(M^A)$ für eine DPOTM M , also ist A in $P^{\mathcal{C}}$. Folglich gilt $\Sigma_i^P \subseteq P^{\Sigma_i^P} = \Delta_{i+1}^P$. Weiter folgt aus $\Delta_{i+1}^P = \text{co}\Delta_{i+1}^P$, dass $\Pi_i^P = \text{co}\Sigma_i^P \subseteq \Delta_{i+1}^P$. Schließlich gilt $\Delta_{i+1}^P = P^{\Sigma_i^P} \subseteq \text{NP}^{\Sigma_i^P} = \Sigma_{i+1}^P$ und $\Delta_{i+1}^P = \text{co}\Delta_{i+1}^P \subseteq \text{co}\Sigma_{i+1}^P = \Pi_{i+1}^P$.

2. Durch Induktion über i wird gezeigt:

$$(\forall i \geq 0) [\Sigma_i^P \subseteq \text{PSPACE}]. \quad (5.13)$$

Der Induktionsanfang, $i = 0$, ist trivial: $\Sigma_0^P = P \subseteq \text{PSPACE}$. Die Induktionsvoraussetzung sagt, dass (5.13) für ein $i \geq 0$ gilt: $\Sigma_i^P \subseteq \text{PSPACE}$. Dann gilt

$$\Sigma_{i+1}^P = \text{NP}^{\Sigma_i^P} \subseteq \text{NP}^{\text{PSPACE}} \subseteq \text{PSPACE},$$

wobei die letzte Inklusion, die auch in der fünften Aussage von Behauptung 5.28 festgestellt wurde, analog zur Inklusion $\text{NP} \subseteq \text{PSPACE}$ in Satz 3.27 plus einer direkten PSPACE -Simulation der Orakelfragen gezeigt werden kann.

3. Der Beweis dieser Aussage wird dem Leser als Übung 5.20 überlassen. \square

Satz 5.30 beschreibt die Inklusionsbeziehungen zwischen den Klassen der Polynomialzeit-Hierarchie. Abbildung 5.10 zeigt diese Inklusionsstruktur als ein Hasse-Diagramm. Wieder wird in dieser Abbildung das Enthaltensein einer Klasse \mathcal{C} in einer Klasse \mathcal{D} durch eine aufwärts gerichtete Linie von \mathcal{C} nach \mathcal{D} angedeutet, und unvergleichbare Klassen sind nicht miteinander verbunden. Abbildung 5.11 stellt diese

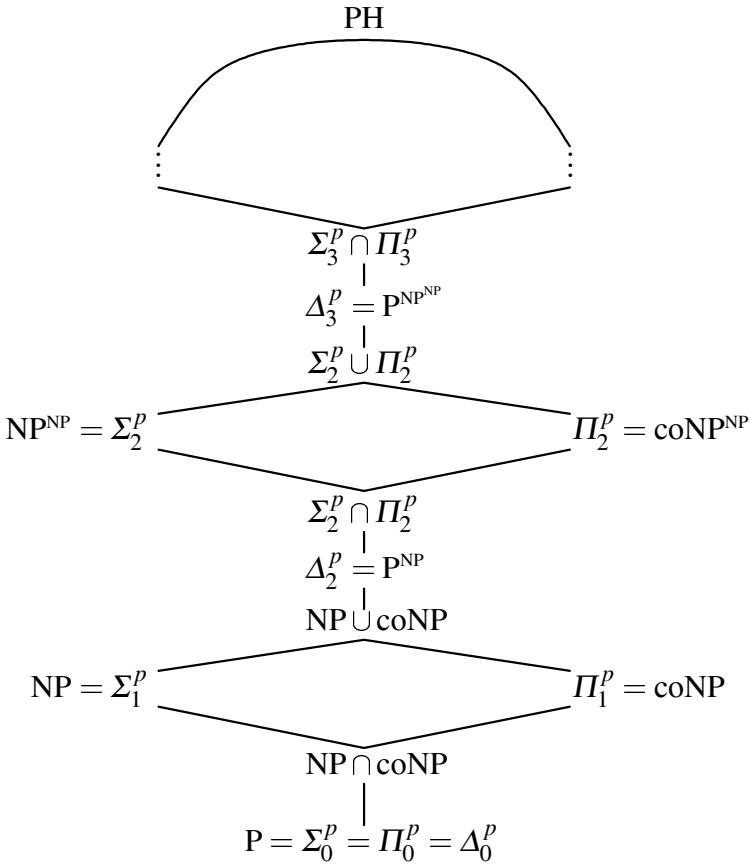


Abb. 5.10. Polynomialzeit-Hierarchie (Hasse-Diagramm)

Inklusionsstruktur als ein Venn-Diagramm dar. Dabei sind dunklere Klassen in helleren Klassen enthalten, und unvergleichbare Klassen haben denselben Grauton. Von keiner dieser Inklusionen weiß man, ob sie echt ist.

Der folgende Satz verallgemeinert die in Satz 5.24 formulierte Darstellung von NP mittels eines polynomiell längenbeschränkten Existenzquantors zu einer Darstellung der Stufen der Polynomialzeit-Hierarchie mittels polynomiell längenbeschränkter, alternierender Existenz- und Allquantoren.

Satz 5.31. Für jedes $i \geq 0$ ist A genau dann in Σ_i^P , wenn es eine Menge B in P und ein Polynom p gibt, so dass für jedes $x \in \Sigma^*$ gilt:

$$x \in A \iff (\exists^p w_1) (\forall^p w_2) \cdots (\mathfrak{Q}^p w_i) [\langle x, w_1, w_2, \dots, w_i \rangle \in B], \quad (5.14)$$

wobei $\mathfrak{Q}^p = \exists^p$, falls i ungerade ist, und $\mathfrak{Q}^p = \forall^p$, falls i gerade ist.

Beweis. Der Satz wird durch Induktion über i bewiesen. Der Induktionsanfang $i = 0$ ist trivial (und der Fall $i = 1$ gilt gemäß Satz 5.24). Die Induktionsvoraussetzung

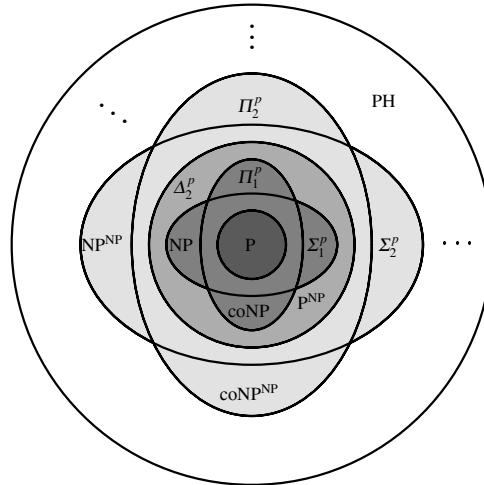


Abb. 5.11. Polynomialzeit-Hierarchie (Venn-Diagramm)

sagt, dass die Aussage von Satz 5.31 für ein $i \geq 0$ gilt. Zu zeigen ist, dass diese Aussage dann ebenso für $i + 1$ gilt.

Für die Richtung von links nach rechts sei A eine Menge in $\Sigma_{i+1}^P = \text{NP}^{\Sigma_i^P}$. Sei M eine NPOTM, die A in der Zeit $q \in \text{Pol}$ akzeptiert, und sei $C \in \Sigma_i^P$ die Orakelmenge für M , d.h., $A = L(M^C)$. Definiere eine Menge D wie folgt:

$$D = \left\{ \langle x, u, v, w \rangle \middle| \begin{array}{l} w \in \text{Wit}_{M^{(\cdot)}}(x), u = \langle u_1, u_2, \dots, u_k \rangle, v = \langle v_1, v_2, \dots, v_\ell \rangle, \\ \text{wobei } u \text{ die Fragen auf Pfad } w \text{ mit Antwort „ja“ enthält} \\ \text{und } v \text{ die Fragen auf Pfad } w \text{ mit Antwort „nein“ enthält} \end{array} \right\}.$$

Offenbar ist D in P . Aus der Definition von D folgt:

$$\begin{aligned} x \in A &\iff M^C \text{ akzeptiert } x & (5.15) \\ &\iff (\exists^q w) [w \in \text{Wit}_{M^C}(x)] \\ &\iff (\exists^q w) (\exists^q u) (\exists^q v) [u = \langle u_1, u_2, \dots, u_k \rangle \wedge v = \langle v_1, v_2, \dots, v_\ell \rangle \\ &\quad \wedge \langle x, u, v, w \rangle \in D \wedge u_1, u_2, \dots, u_k \in C \wedge v_1, v_2, \dots, v_\ell \notin C]. \end{aligned}$$

Definiere die Mengen

$$\begin{aligned} C_{\text{ja}} &= \{u \mid u = \langle u_1, u_2, \dots, u_k \rangle \text{ und } u_1, u_2, \dots, u_k \in C\}; \\ C_{\text{nein}} &= \{v \mid v = \langle v_1, v_2, \dots, v_\ell \rangle \text{ und } v_1, v_2, \dots, v_\ell \notin C\}. \end{aligned}$$

Da $C \in \Sigma_i^P$, $k \leq q(|x|)$ und Σ_i^P unter Paarung abgeschlossen ist, ist C_{ja} in Σ_i^P . Ebenso ist C_{nein} in Π_i^P , da $\overline{C} \in \Pi_i^P$, $\ell \leq q(|x|)$ und Π_i^P unter Paarung abgeschlossen ist; siehe Übung 5.21.

Nach Induktionsvoraussetzung gibt es für $C_{\text{ja}} \in \Sigma_i^P$ und $C_{\text{nein}} \in \Pi_i^P$ Mengen E und F in P und Polynome r und s , so dass gilt:

$$u \in C_{ja} \iff (\exists^r y_1) (\forall^r y_2) \cdots (\mathfrak{Q}^r y_i) [\langle u, y_1, y_2, \dots, y_i \rangle \in E]; \quad (5.16)$$

$$v \in C_{nein} \iff (\forall^s z_1) (\exists^s z_2) \cdots (\overline{\mathfrak{Q}}^s z_i) [\langle v, z_1, z_2, \dots, z_i \rangle \in F], \quad (5.17)$$

wobei $\mathfrak{Q}^r = \exists^r$ und $\overline{\mathfrak{Q}}^s = \forall^s$, falls i ungerade ist, und $\mathfrak{Q}^r = \forall^r$ und $\overline{\mathfrak{Q}}^s = \exists^s$, falls i gerade ist. Setzt man die rechten Seiten der Äquivalenzen (5.16) und (5.17) in (5.15) ein, so erhält man:

$$\begin{aligned} x \in A \iff & (\exists^q w) (\exists^q u) (\exists^q v) [\langle x, u, v, w \rangle \in D \wedge \\ & (\exists^r y_1) (\forall^r y_2) \cdots (\mathfrak{Q}^r y_i) [\langle u, y_1, y_2, \dots, y_i \rangle \in E] \wedge \\ & (\forall^s z_1) (\exists^s z_2) \cdots (\overline{\mathfrak{Q}}^s z_i) [\langle v, z_1, z_2, \dots, z_i \rangle \in F]]. \end{aligned} \quad (5.18)$$

Zieht man nun abwechselnd die Quantoren aus den unteren beiden Zeilen der Äquivalenz (5.18) nach vorn und fasst man aufeinander folgende Quantoren gleichen Typs zu einem Quantor desselben Typs zusammen, so ergibt sich:

$$\begin{aligned} x \in A \iff & \underbrace{(\exists^q w) (\exists^q u) (\exists^q v)}_{\text{zusammenfassen zu } (\exists^p w_1)} (\underbrace{\exists^r y_1}_{(\forall^p w_2)} \underbrace{(\forall^r y_2) (\forall^s z_1)}_{(\mathfrak{Q}^p w_i)} \cdots \underbrace{(\mathfrak{Q}^r y_i) (\mathfrak{Q}^s z_{i-1})}_{(\mathfrak{Q}^p w_{i+1})} \underbrace{(\overline{\mathfrak{Q}}^s z_i)}_{(\overline{\mathfrak{Q}}^p w_{i+1})} \\ & [\langle x, u, v, w \rangle \in D \wedge \langle u, y_1, y_2, \dots, y_i \rangle \in E \wedge \langle v, z_1, z_2, \dots, z_i \rangle \in F] \\ \iff & (\exists^p w_1) (\forall^p w_2) \cdots (\overline{\mathfrak{Q}}^p w_{i+1}) [\langle x, w_1, w_2, \dots, w_{i+1} \rangle \in B], \end{aligned} \quad (5.19)$$

wobei $p = \max\{3q+r, r+s\} + c$ ein Polynom ist, das von den Polynomen q, r und s und einer Konstanten c abhängt, die von der Paarung von Wörtern beim Zusammenfassen der Quantoren herröhrt. Der Zusammenfassung der Quantoren gemäß wird nun eine Menge B geeignet so definiert, dass gilt:

$$\begin{aligned} & \langle x, w_1, w_2, \dots, w_{i+1} \rangle \in B \\ \iff & \langle x, u, v, w \rangle \in D \wedge \langle u, y_1, y_2, \dots, y_i \rangle \in E \wedge \langle v, z_1, z_2, \dots, z_i \rangle \in F. \end{aligned}$$

Da D, E und F in P liegen, ist auch B eine Menge in P . Gemäß der Äquivalenz (5.19) erfüllt A die Darstellung (5.14) für $i+1$. Der Induktionsbeweis ist damit für die Richtung von links nach rechts abgeschlossen.

Umgekehrt seien nun für die Richtung von rechts nach links eine Menge B in P und ein Polynom p gegeben, so dass A folgendermaßen dargestellt werden kann:

$$A = \{x \mid (\exists^p w_1) (\forall^p w_2) \cdots (\overline{\mathfrak{Q}}^p w_{i+1}) [\langle x, w_1, w_2, \dots, w_{i+1} \rangle \in B]\},$$

wobei $\mathfrak{Q}^p = \exists^p$, falls i gerade ist, und $\mathfrak{Q}^p = \forall^p$, falls i ungerade ist. Definiere eine Menge C wie folgt:

$$C = \{\langle x, w_1 \rangle \mid |w_1| \leq p(|x|) \wedge (\forall^p w_2) \cdots (\overline{\mathfrak{Q}}^p w_{i+1}) [\langle x, w_1, w_2, \dots, w_{i+1} \rangle \in B]\}.$$

Somit gilt:

$$x \in A \iff (\exists^p w_1) [\langle x, w_1 \rangle \in C].$$

Nach Induktionsvoraussetzung ist die Menge C in Π_i^P ; also ist ihr Komplement, \overline{C} , in Σ_i^P . Sei M eine NPOTM, die mit dem Orakel \overline{C} die Menge A folgendermaßen akzeptiert: Bei Eingabe x

- rät M nichtdeterministisch ein Wort w_1 mit $|w_1| \leq p(|x|)$,
- fragt für jedes geratene w_1 das Orakel nach dem Paar $\langle x, w_1 \rangle$ und
- akzeptiert x auf diesem Pfad genau dann, wenn die Antwort „nein“ ist.

Somit ist $A = L(M^{\bar{C}})$. Es folgt $A \in \text{NP}^{\Sigma_i^p} = \Sigma_{i+1}^p$, womit der Induktionsbeweis abgeschlossen ist. \square

Korollar 5.32. Für jedes $i \geq 0$ ist A genau dann in Π_i^p , wenn es eine Menge B in P und ein Polynom p gibt, so dass für jedes $x \in \Sigma^*$ gilt:

$$x \in A \iff (\forall^p w_1) (\exists^p w_2) \cdots (\mathfrak{Q}^p w_i) [\langle x, w_1, w_2, \dots, w_i \rangle \in B],$$

wobei $\mathfrak{Q}^p = \forall^p$, falls i ungerade ist, und $\mathfrak{Q}^p = \exists^p$, falls i gerade ist.

Der folgende Satz zeigt, dass die Polynomialzeit-Hierarchie genau wie die booleschen Hierarchie die „Kollapsierung nach oben“ (*upward collapse*) besitzt (siehe Satz 5.10): Fällt eine ihrer Stufen auf die vorhergehende Stufe, dann kollabiert die ganze Hierarchie auf diese spezielle endliche Stufe. Dieselbe Konsequenz tritt selbst dann ein, wenn eine der Klassen Σ_i^p mit ihrer Komplementklasse, Π_i^p , übereinstimmt.

Satz 5.33. 1. Gilt $\Sigma_i^p = \Sigma_{i+1}^p$ für irgendein $i \geq 0$, so folgt

$$\Sigma_i^p = \Pi_i^p = \Delta_{i+1}^p = \Sigma_{i+1}^p = \Pi_{i+1}^p = \cdots = \text{PH}.$$

2. Gilt $\Sigma_i^p = \Pi_i^p$ für irgendein $i \geq 1$, so folgt

$$\Sigma_i^p = \Pi_i^p = \Delta_{i+1}^p = \Sigma_{i+1}^p = \Pi_{i+1}^p = \cdots = \text{PH}.$$

Beweis. Zunächst zeigen wir, dass die Voraussetzung der ersten Aussage die der zweiten impliziert. Angenommen, es gilt $\Sigma_i^p = \Sigma_{i+1}^p$ für irgendein $i \geq 0$. Dann gilt

$$\Pi_i^p \subseteq \Sigma_{i+1}^p = \Sigma_i^p,$$

woraus $\Sigma_i^p = \Pi_i^p$ folgt; siehe Übung 5.22.

Nun sei $\Sigma_i^p = \Pi_i^p$ für irgendein $i \geq 1$ angenommen.³ Wir zeigen, dass dies $\Sigma_i^p = \Sigma_{i+1}^p$ impliziert. Sei A eine beliebige Menge in Σ_{i+1}^p . Nach Satz 5.31 gibt es eine Menge $B \in \text{P}$ und ein Polynom p , so dass für jedes $x \in \Sigma^*$ gilt:

$$x \in A \iff (\exists^p w_1) (\forall^p w_2) \cdots (\mathfrak{Q}^p w_{i+1}) [\langle x, w_1, w_2, \dots, w_{i+1} \rangle \in B],$$

wobei $\mathfrak{Q}^p = \exists^p$, falls i gerade ist, und $\mathfrak{Q}^p = \forall^p$, falls i ungerade ist. Definiere eine Menge C durch

³ Nach Definition gilt $\Sigma_0^p = \text{P} = \Pi_0^p$ unmittelbar, aber $\text{P} = \Sigma_0^p = \Sigma_1^p = \text{NP}$ ist sehr zweifelhaft. An welcher Stelle genau funktioniert das Argument in diesem Beweis nicht, wenn man $\Sigma_0^p = \Pi_0^p \implies \Sigma_0^p = \Sigma_1^p$ zeigen wollte? Siehe Übung 5.23.

$$C = \left\{ \langle x, w_1 \rangle \middle| \begin{array}{l} |w_1| \leq p(|x|) \wedge (\forall^p w_2) (\exists^p w_3) \cdots \\ (\mathfrak{Q}^p w_{i+1}) [\langle x, w_1, w_2, \dots, w_{i+1} \rangle \in B] \end{array} \right\}.$$

Nach Satz 5.31 ist C in $\Pi_i^p = \Sigma_i^p$. Wieder nach Satz 5.31 gibt es für jedes $C \in \Sigma_i^p$ eine Menge $D \in P$ und ein Polynom q , so dass für jedes $x \in \Sigma^*$ gilt:

$$C = \left\{ \langle x, w_1 \rangle \middle| \begin{array}{l} |w_1| \leq q(|x|) \wedge (\exists^q w_2) (\forall^q w_3) \cdots \\ (\mathfrak{Q}^q w_{i+1}) [\langle x, w_1, w_2, \dots, w_{i+1} \rangle \in D] \end{array} \right\},$$

wobei $\mathfrak{Q}^q = \forall^q$, falls i gerade ist, und $\mathfrak{Q}^q = \exists^q$, falls i ungerade ist. Folglich gilt:

$$x \in A \iff \underbrace{(\exists^p w_1) (\exists^q w_2)}_{\text{zusammenfassen zu } (\exists^r w)} (\forall^q w_3) \cdots (\mathfrak{Q}^q w_{i+1}) [\langle x, w_1, w_2, \dots, w_{i+1} \rangle \in D].$$

Fasst man die ersten beiden existenziellen Quantoren zu einem existenziellen Quantor zusammen, dessen Länge durch das Polynom $r = p + q$ beschränkt ist (wobei der dem Paaren geschuldete konstante Aufwand vernachlässigt wird), und wendet man nochmals Satz 5.31 an, so ergibt sich $A \in \Sigma_i^p$. Da A beliebig aus Σ_{i+1}^p gewählt war, ist damit $\Sigma_i^p = \Sigma_{i+1}^p$ gezeigt.

Eine einfache Induktion zeigt nun, dass jede Stufe Σ_k^p mit $k \geq i$ auf Σ_i^p kollabiert:

$$\Sigma_{i+2}^p = \text{NP}^{\Sigma_{i+1}^p} = \text{NP}^{\Sigma_i^p} = \Sigma_{i+1}^p = \Sigma_i^p,$$

und so weiter. □

Hat die Polynomialzeit-Hierarchie vollständige Mengen? Welche Probleme sind für die Σ_i^p -Stufen vollständig und welche für die Π_i^p -Stufen? Die Charakterisierung von Σ_i^p und Π_i^p durch längenbeschränkte Quantoren aus Satz 5.31 und Korollar 5.32 legt nahe, dass man gute Kandidaten für solche vollständigen Mengen erhält, indem man das Erfüllbarkeitsproblem geeignet verallgemeinert: Statt der Erfüllbarkeit von booleschen Formeln wird nun für *quantifizierte* boolesche Formeln getestet, ob sie wahr sind oder nicht, siehe dazu Definition 2.26 in Abschnitt 2.3.

Definition 5.34 (QBF-Problem). *Definiere das Problem*

$$\text{QBF} = \{F \mid F \text{ ist eine wahre geschlossene QBF}\}.$$

Im Abschnitt 2.3 wurde gezeigt, dass sich jede geschlossene QBF zu entweder wahr oder falsch auswerten lässt und dass jede QBF in eine äquivalente QBF in Pränexnormalform transformiert werden kann, siehe Definition 2.28 und Beispiel 2.29. Ebenso kann man aufeinander folgende Quantoren desselben Typs zu einem Quantor dieses Typs zusammenfassen, welcher dann eine Menge von Variablen quantifiziert. Durch Umbenennen der quantifizierten Variablen können die Variablenmengen nach den Quantoren paarweise disjunkt gemacht werden.

Ist die Anzahl der alternierenden Quantoren in der gegebenen QBF durch eine feste Konstante $i \geq 1$ beschränkt, so ergeben sich Spezialfälle von QBF, nämlich die Probleme Σ_i SAT und Π_i SAT. Dabei enthält Σ_i SAT genau die Formeln aus QBF, die in

Pränexnormalform sind und einen Präfix von i alternierenden Quantoren haben, der mit \exists beginnt. Π_i SAT enthält genau die Formeln aus QBF, die in Pränexnormalform sind und einen Präfix von i alternierenden Quantoren haben, der mit \forall beginnt.

Definition 5.35 (QBF-Problem mit beschränkter Zahl von Alternierungen).

1. Sei $i \geq 1$. Eine QBF F heißt Σ_i SAT-Formel, falls F geschlossen und von der Form

$$F = (\exists X_1) (\forall X_2) \cdots (\mathfrak{Q} X_i) H(X_1, X_2, \dots, X_i)$$

ist, wobei die X_j paarweise disjunkte Variablenmengen sind, $\mathfrak{Q} \in \{\exists, \forall\}$, die i Quantoren zwischen \exists und \forall alternieren und H eine boolesche Formel ohne Quantoren ist. Definiere das Problem

$$\Sigma_i\text{SAT} = \{F \mid F \text{ ist eine wahre } \Sigma_i\text{SAT-Formel}\}.$$

2. Sei $i \geq 1$. Eine QBF F heißt Π_i SAT-Formel, falls F geschlossen und von der Form

$$F = (\forall X_1) (\exists X_2) \cdots (\mathfrak{Q} X_i) H(X_1, X_2, \dots, X_i),$$

ist, wobei die X_j paarweise disjunkte Variablenmengen sind, $\mathfrak{Q} \in \{\exists, \forall\}$, die i Quantoren zwischen \forall und \exists alternieren und H eine boolesche Formel ohne Quantoren ist. Definiere das Problem

$$\Pi_i\text{SAT} = \{F \mid F \text{ ist eine wahre } \Pi_i\text{SAT-Formel}\}.$$

Satz 5.36. 1. QBF ist PSPACE-vollständig.

2. Für jedes $i \geq 1$ gilt: $\Sigma_i\text{SAT}$ ist Σ_i^P -vollständig und $\Pi_i\text{SAT}$ ist Π_i^P -vollständig.
3. Falls eine für PH vollständige Menge existiert, so kollabiert PH auf eine endliche Stufe: $\text{PH} = \Sigma_i^P = \Pi_i^P$ für ein i .

Beweis. 1. Die erste Aussage des Satzes kann mit der Methode gezeigt werden, die beim Beweis des Satzes von Savitch angewandt wurde (siehe den Beweis von Satz 3.29); siehe Übung 5.24. Alternativ dazu kann die erste Aussage des Satzes auch analog zum Beweis der zweiten Aussage des Satzes gezeigt werden.

2. Beim Beweis der zweiten Aussage folgt die Zugehörigkeit von $\Sigma_i\text{SAT}$ zu Σ_i^P bzw. von $\Pi_i\text{SAT}$ zu Π_i^P unmittelbar aus Satz 5.31 und Korollar 5.32.

Sei i ungerade. (Der Fall eines geraden i kann in ähnlicher Weise gezeigt werden.) Um die Σ_i^P -Härte von $\Sigma_i\text{SAT}$ zu zeigen, sei A eine Menge in Σ_i^P . Wieder nach Satz 5.31 gibt es eine Menge B in P und ein Polynom p , so dass für jedes $x \in \Sigma^*$:

$$x \in A \iff (\exists^p y_1) (\forall^p y_2) \cdots (\exists^p y_i) [\langle x, y_1, y_2, \dots, y_i \rangle \in B]. \quad (5.20)$$

Sei M eine deterministische Polynomialzeit-Turingmaschine, die B entscheidet, d.h., $L(M) = B$. Nach dem Satz von Cook (siehe Satz 3.49) kann die Berechnung von M bei einer Eingabe der Form $\langle x, y_1, y_2, \dots, y_i \rangle$ als eine boolesche Formel

$\varphi_M(X, Y_1, Y_2, \dots, Y_i, Z)$ codiert werden, wobei $X \cup \bigcup_{j=1}^i Y_j$ die Menge der EingabevARIABLEN und Z die Menge aller übrigen Variablen in der Cook-Reduktion ist. Dabei entspricht jede Variable in $X \cup \bigcup_{j=1}^i Y_j$ einem Bit in den Wörtern x, y_1, y_2, \dots, y_i der Eingabe, und haben diese Bits die Werte 1 oder 0, so werden die entsprechenden Variablen wahr oder falsch gesetzt. Insbesondere ergibt das Setzen aller Bits in x eine boolesche Formel $\varphi_{\langle M, x \rangle}$, die nur von den Variablen aus $Z \cup \bigcup_{j=1}^i Y_j$ abhängt.

Nach dem Satz von Cook gilt für jede Eingabe $\langle x, y_1, y_2, \dots, y_i \rangle$:

$$\begin{aligned} \langle x, y_1, y_2, \dots, y_i \rangle \in B &\iff \varphi_{\langle M, x, y_1, y_2, \dots, y_i \rangle}(Z) \in \text{SAT} \\ &\iff (\exists^q z) [\varphi_{\langle M, x, y_1, y_2, \dots, y_i, z \rangle} \text{ ist wahr}], \end{aligned} \quad (5.21)$$

wobei q ein Polynom in $|x|$ ist, das die Anzahl der Variablen in Z beschränkt; siehe den Beweis von Satz 3.49. Die Äquivalenzen (5.20) und (5.21) implizieren nun für jedes $x \in \Sigma^*$:

$$\begin{aligned} x \in A &\iff (\exists^p y_1) (\forall^p y_2) \cdots (\exists^p y_i) [\langle x, y_1, y_2, \dots, y_i \rangle \in B] \\ &\iff (\exists^p y_1) (\forall^p y_2) \cdots \underbrace{(\exists^p y_i)}_{(\exists^r z_1)} \underbrace{(\exists^q z)}_{[\varphi_{\langle M, x, y_1, y_2, \dots, y_i, z \rangle} \text{ ist wahr}]}, \end{aligned} \quad (5.22)$$

wobei $r = p + q$ ein Polynom ist. Sei $V = Y_i \cup Z$ die Menge, die alle Variablen aus Y_i und Z enthält. Die Abbildung eines gegebenen Wortes x auf die Σ_i SAT-Formel

$$F_x = (\exists Y_1) (\forall Y_2) \cdots (\forall Y_{i-1}) (\exists V) \varphi_{\langle M, x \rangle}(Y_1, Y_2, \dots, Y_{i-1}, V)$$

definiert die gesuchte \leq_m^p -Reduktion von A auf Σ_i SAT. Da die Cook-Reduktion polynomialzeit-berechenbar ist, kann F_x aus x in einer Zeit polynomiell in $|x|$ berechnet werden. Gemäß der Äquivalenz (5.22) ist x genau dann in A , wenn F_x in Σ_i SAT ist, womit die Σ_i^p -Vollständigkeit von Σ_i SAT gezeigt ist. Die Π_i^p -Vollständigkeit von Π_i SAT wird analog gezeigt.

3. Um die dritte Aussage des Satzes zu zeigen, sei L eine vollständige Menge für PH. Da L in PH ist, gibt es ein kleinstes i mit $L \in \Sigma_i^p$. Weil L hart für PH ist, gilt $X \leq_m^p L$ für jedes X in PH. Da $\Sigma_i^p \leq_m^p$ -abgeschlossen ist, folgt $X \in \Sigma_i^p$ für jedes $X \in \text{PH}$. Somit gilt $\text{PH} = \Sigma_i^p = \Pi_i^p$. \square

Korollar 5.37. $\text{PSPACE} = \Sigma_i^p$ genau dann, wenn QBF $\in \Sigma_i^p$.

5.3 Paralleler Zugriff auf NP

Abbildung 5.9 in Abschnitt 5.2 zeigte als Beispiel einer polynomialzeit-beschränkten Turing-Reduktion einen Algorithmus, der mit Präfixsuche die lexikographisch kleinste Lösung einer gegebenen GI-Instanz findet. Da die Anzahl aller Lösungen exponentiell in der Größe n der Eingabe ist, etwa 2^{n^c} für eine Konstante c , stellt dieser Algorithmus höchstens $\mathcal{O}(\log 2^{n^c}) = \mathcal{O}(n^c)$ Orakelfragen und terminiert somit in Polynomialzeit. Wenn die Größe des Lösungsraums selbst durch ein Polynom in

der Eingabegröße beschränkt ist, etwa durch n^k für eine Konstante k , dann ist eine Binärsuche (oder Präfixsuche) in der Zeit $\mathcal{O}(\log n^k) = \mathcal{O}(\log n)$ erfolgreich.

Zur Illustration betrachten wir das Problem, zu entscheiden, ob die Unabhängigkeitszahl eines gegebenen Graphen ungerade ist oder nicht, bzw. das Problem, die Unabhängigkeitszahlen zweier gegebener Graphen miteinander zu vergleichen. Man erinnere sich, dass eine unabhängige Menge eines Graphen eine Teilmenge I der Knotenmenge des Graphen ist, so dass keine zwei Knoten in I benachbart sind. Die Größe von I ist die Anzahl der Knoten in I .

Definition 5.38 (Unabhängigkeitszahl-Probleme). Für einen Graphen G ist die Unabhängigkeitszahl von G , bezeichnet mit $\alpha(G)$, die Größe einer maximalen unabhängigen Menge von G . Definiere die Probleme:

$$\text{IN-Odd} = \{G \mid G \text{ ist ein Graph, so dass } \alpha(G) \text{ ungerade ist}\};$$

$$\text{IN-Equ} = \{\langle G, H \rangle \mid G \text{ und } H \text{ sind Graphen mit } \alpha(G) = \alpha(H)\};$$

$$\text{IN-Geq} = \{\langle G, H \rangle \mid G \text{ und } H \text{ sind Graphen mit } \alpha(G) \geq \alpha(H)\}.$$

Natürlich gilt $\alpha(G) \leq n$ für jeden Graphen G mit n Knoten. Daher erfordert beispielsweise die Entscheidung, ob G in IN-Odd ist, nicht mehr als $\mathcal{O}(\log n)$ sequenzielle Turing-Fragen. Papadimitriou und Zachos [PZ83] führten die entsprechende Komplexitätsklasse ein, $\text{P}^{\text{NP}[\log]}$, die sich später auch als die Θ_2^P -Stufe der Polynomialzeit-Hierarchie einen Namen machte.

Definition 5.39. 1. Definiere $\text{P}^{\text{NP}[\log]}$ als die Klasse der Probleme A , die von einer DPOTM M mit Orakel $B \in \text{NP}$ entscheidbar sind, d.h., $A = L(M^B)$, wobei M bei jeder Eingabe der Länge n höchstens $\mathcal{O}(\log n)$ sequenzielle Turing-Fragen an B stellt.

2. Definiere $\Theta_i^P = \text{P}^{\Sigma_{i-1}^P[\log]}$ für jedes $i \geq 1$. Insbesondere ist $\Theta_2^P = \text{P}^{\text{NP}[\log]}$.

Nach Definition gilt $\Sigma_{i-1}^P \cup \Pi_{i-1}^P \subseteq \Delta_i^P \subseteq \Delta_i^P$ für jedes $i \geq 1$. Die obigen Kommentare zeigen unmittelbar, dass Θ_2^P eine obere Schranke für die Unabhängigkeitszahl-Probleme aus Definition 5.38 ist. Die Einzelheiten des Beweises dieser Aussage werden dem Leser als Übung 5.25(a) überlassen. Wir werden später (nämlich in Satz 5.44) sehen, dass Θ_2^P ebenfalls eine untere Schranke für diese Probleme liefert.

Behauptung 5.40. IN-Odd, IN-Equ und IN-Geq sind in Θ_2^P .

Es gibt etwa ein Dutzend von Charakterisierungen der Komplexitätsklasse $\Theta_2^P = \text{P}^{\text{NP}[\log]}$; siehe [Wag90]. Ganz wesentlich ist unter diesen die Charakterisierung von Θ_2^P als der Abschluss von NP unter der polynomialzeit-beschränkten Truth-table-Reduzierbarkeit.

Definition 5.41 (Truth-table-Reduzierbarkeit und -Vollständigkeit). Für ein Alphabet $\Sigma = \{0, 1\}$ seien A und B beliebige Mengen von Wörtern über Σ , und \mathcal{C} sei eine beliebige Komplexitätsklasse. Zur Erinnerung: Die charakteristische Funktion von B , bezeichnet mit c_B , ist definiert durch $c_B(q) = 1$, falls $q \in B$, und $c_B(q) = 0$, falls $q \notin B$.

1. Definiere die polynomialzeit-beschränkte Truth-table-Reduzierbarkeit, bezeichnet mit \leq_{tt}^p , wie folgt: $A \leq_{tt}^p B$ genau dann, wenn es eine Funktion $f \in FP$ gibt, so dass $f(x) = \langle \tau, q_1, q_2, \dots, q_k \rangle$ für jede Eingabe x , wobei q_1, q_2, \dots, q_k die dabei erzeugten Truth-table-Fragen sind und τ eine k -stellige boolesche Funktion (codiert etwa durch ihre Wahrheitstafel – englisch „truth table“) ist, und es gilt:

$$x \in A \iff \tau(c_B(q_1), c_B(q_2), \dots, c_B(q_k)) = 1.$$

Dabei ist sowohl die Anzahl k der Fragen als auch die Länge $|q_i|$ einer jeden Frage polynomiell in $|x|$, weil f in FP ist.

2. Eine Menge B ist \leq_{tt}^p -hart für \mathcal{C} , falls $A \leq_{tt}^p B$ für jedes $A \in \mathcal{C}$.
3. Eine Menge B ist \leq_{tt}^p -vollständig für \mathcal{C} , falls $B \leq_{tt}^p$ -hart für \mathcal{C} ist und $B \in \mathcal{C}$.
4. Der \leq_{tt}^p -Abschluss von \mathcal{C} ist definiert durch

$$P_{tt}^{\mathcal{C}} = \{A \mid (\exists B \in \mathcal{C}) [A \leq_{tt}^p B]\}.$$

Eine Klasse \mathcal{C} heißt abgeschlossen unter der \leq_{tt}^p -Reduzierbarkeit (kurz \leq_{tt}^p -abgeschlossen), falls $\mathcal{C} = P_{tt}^{\mathcal{C}}$.

Die Truth-table-Reduzierbarkeit ist flexibler als die Many-one-Reduzierbarkeit, jedoch weniger flexibel als die Turing-Reduzierbarkeit. Der wesentliche Unterschied zwischen den beiden letztgenannten Reduzierbarkeiten ist, dass die Orakelfragen in Turing-Reduktionen von den Antworten auf zuvor gestellte Fragen abhängen können – in diesem Sinn sind Turing-Reduktionen *adaptiv*. Im Gegensatz dazu sind Truth-table-Fragen *nichtadaptiv*, da sie alle berechnet werden, bevor irgendeine von ihnen gestellt wird, und dann werden sie alle parallel gestellt. Deshalb heißt der Truth-table-Zugriff auf ein Orakel auch *paralleler Orakelzugriff*. Insbesondere sagt man oft auch, dass die Komplexitätsklasse P_{tt}^{NP} den Begriff des „parallelen Zugriffs auf NP“ verkörpert. Korollar 5.55 wird die Gleichheit $P_{tt}^{NP} = \Theta_2^P$ zeigen.

Beispiel 5.42 (Truth-table-Reduzierbarkeit). Zur Illustration wird nun eine \leq_{tt}^p -Reduktion f von IN-Odd auf IS konstruiert. Die Menge $IS = \{\langle G, k \rangle \mid \alpha(G) \geq k\}$ ist NP-vollständig, siehe Satz 3.54. Da also IS in NP ist und IN-Odd \leq_{tt}^p IS gilt, folgt $IN-Odd \in \Theta_2^P$ aus Korollar 5.55, das in Abschnitt 5.4 gezeigt werden wird.

Gegeben ein Graph G mit n Knoten, definiere $f(G) = \langle \tau, q_1, q_2, \dots, q_n \rangle$, wobei $q_i = \langle G, i \rangle$ die i -te Frage ist, $1 \leq i \leq n$, und die boolesche Funktion τ definiert ist durch

$$\begin{aligned} \tau(b_1, b_2, \dots, b_n) = \\ \begin{cases} 1 & \text{falls } b_1 = \dots = b_i = 1 \text{ und } b_{i+1} = \dots = b_n = 0 \text{ für ein ungerades } i \\ 0 & \text{sonst,} \end{cases} \end{aligned}$$

wobei $b_i = c_{IS}(q_i)$ für $1 \leq i \leq n$, d.h., $b_i = 1 \iff \alpha(G) \geq i$.

Tabelle 5.1 illustriert diese \leq_{tt}^p -Reduktion für den Fall von vier Fragen, d.h., für einen Graphen mit vier Knoten. Lediglich die fettgedruckten Spalten der Tabelle sind für die Definition von τ relevant, da die den anderen Spalten entsprechenden Fälle nicht auftreten können. In den nicht fettgedruckten Spalten der Tabelle könnten also beliebige Werte von τ gewählt werden.

Tabelle 5.1. Beispiel einer \leq_m^p -Reduktion von IN-0dd auf IS mit vier Fragen

$b_1: „q_1 = \langle G, 1 \rangle \in IS?“$	0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
$b_2: „q_2 = \langle G, 2 \rangle \in IS?“$	0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
$b_3: „q_3 = \langle G, 3 \rangle \in IS?“$	0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
$b_4: „q_4 = \langle G, 4 \rangle \in IS?“$	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
$\tau(b_1, b_2, b_3, b_4)$	0 0 0 0 0 0 0 1 0 0 0 0 0 1 0

Das folgende Lemma gibt eine hinreichende Bedingung für Θ_2^P -Härte an. Dies ist ein weiteres Beispiel für die Anwendung der Wagner-Technik, um eine untere Schranke für ein Problem von NP-Härte zur Härte bezüglich einer Klasse oberhalb von NP anzuheben. Der Unterschied zu Lemma 5.11, welches eine analoge hinreichende Bedingung für den Nachweis der $BH_k(NP)$ -Härte eines Problems für jedes k angibt, besteht darin, dass der Wert von k in Lemma 5.43 nicht fixiert ist.

Lemma 5.43 (Wagner). *Seien A ein NP-vollständiges und B ein beliebiges Problem. Angenommen, es gibt eine polynomialzeit-berechenbare Funktion f, so dass für alle $k \geq 1$ und alle Wörter $x_1, \dots, x_k \in \Sigma^*$ mit der Eigenschaft, dass $x_j \in A$ aus $x_{j+1} \in A$ für jedes j mit $1 \leq j < k$ folgt, die Äquivalenz*

$$\|\{i \mid x_i \in A\}\| \text{ ist ungerade} \iff f(\langle x_1, x_2, \dots, x_k \rangle) \in B \quad (5.23)$$

gilt. Dann ist B ein Θ_2^P -hartes Problem.

Um die Anwendung von Lemma 5.43 zu illustrieren, wird nun gezeigt, dass jedes der drei oben definierten Unabhängigkeitszahl-Probleme Θ_2^P -vollständig ist.

Satz 5.44. IN-0dd, IN-Equ und IN-Geq sind Θ_2^P -vollständig.

Beweis. Die Zugehörigkeit dieser Probleme zu Θ_2^P wurde bereits in Behauptung 5.40 festgestellt. Unter Benutzung von Lemma 5.43 wird nun gezeigt, dass IN-Equ Θ_2^P -hart ist. Der Beweis der Θ_2^P -Härte von IN-0dd und IN-Geq ist ganz ähnlich; siehe Übung 5.25(b).

Um Lemma 5.43 anzuwenden, wähle 3-SAT als das NP-vollständige Problem A und die Menge $B = \text{IN-Equ}$ des Lemmas. Da Θ_2^P unter Komplementbildung abgeschlossen ist, genügt es, eine in Polynomialzeit berechenbare Funktion f so zu definieren, dass die Äquivalenz

$$\|\{\varphi_i \mid \varphi_i \in \text{3-SAT}\}\| \text{ ist gerade} \iff f(\langle \varphi_1, \dots, \varphi_{2k} \rangle) \in \text{IN-Equ} \quad (5.24)$$

für alle $k \geq 1$ und alle diejenigen booleschen Formeln $\varphi_1, \varphi_2, \dots, \varphi_{2k}$ (geeignet als Wörter in Σ^* codiert) erfüllt ist, für die $\varphi_i \in \text{3-SAT}$ aus $\varphi_{i+1} \in \text{3-SAT}$ folgt, $1 \leq i < 2k$. Sei g eine Reduktion, die 3-SAT auf IS \leq_m^p -reduziert, so dass die folgende Eigenschaft erfüllt ist: Für jede boolesche Formel φ ist $g(\varphi) = \langle G, \ell \rangle$, wobei G ein Graph ist und ℓ eine positive ganze Zahl (nämlich die Anzahl der Klauseln von φ), und es gilt:

$$\varphi \in 3\text{-SAT} \implies \alpha(G) = \ell; \quad (5.25)$$

$$\varphi \notin 3\text{-SAT} \implies \alpha(G) = \ell - 1. \quad (5.26)$$

Die Konstruktion von g mit den Eigenschaften (5.25) und (5.25) wird dem Leser als Übung 5.25(b) überlassen. (Die Reduktion für $3\text{-SAT} \leq_m^p \text{IS}$ im Beweis von Satz 3.54 hat diese gewünschten Eigenschaften nicht.)

Sei $k \geq 1$, und seien $\varphi_1, \varphi_2, \dots, \varphi_{2k}$ boolesche Formeln, für die $\varphi_i \in 3\text{-SAT}$ aus $\varphi_{i+1} \in 3\text{-SAT}$ für jedes $i < 2k$ folgt. Definiere $g(\varphi_i) = \langle G_i, \ell_i \rangle$ für jedes i mit $1 \leq i \leq 2k$. Es gilt $\|\{i \mid \varphi_i \in 3\text{-SAT}\}\| = m$ genau dann, wenn $\varphi_1, \varphi_2, \dots, \varphi_m \in 3\text{-SAT}$ und $\varphi_{m+1}, \varphi_{m+2}, \dots, \varphi_{2k} \notin 3\text{-SAT}$. Daraus folgt:

- Ist $\|\{i \mid \varphi_i \in 3\text{-SAT}\}\|$ gerade, so folgt für jedes $i \in \{1, 2, \dots, k\}$:

$$\varphi_{2i-1} \in 3\text{-SAT} \iff \varphi_{2i} \in 3\text{-SAT}.$$

Somit gilt für jedes $i \in \{1, 2, \dots, k\}$:

$$\alpha(G_{2i-1}) + \ell_{2i} = \alpha(G_{2i}) + \ell_{2i-1}.$$

- Ist $\|\{i \mid \varphi_i \in 3\text{-SAT}\}\|$ ungerade, so gibt es ein $i \in \{1, 2, \dots, k\}$ mit:

$$\begin{aligned} \varphi_{2i-1} \in 3\text{-SAT} \quad \text{und} \quad \varphi_{2i} \notin 3\text{-SAT} \quad \text{und} \\ (\forall j \in \{1, 2, \dots, k\}, j \neq i) [\varphi_{2j-1} \in 3\text{-SAT} \iff \varphi_{2j} \in 3\text{-SAT}]. \end{aligned}$$

Dies impliziert, dass für diese Zahl i gilt:

$$\begin{aligned} \alpha(G_{2i-1}) + \ell_{2i} - 1 &= \alpha(G_{2i}) + \ell_{2i-1}; \\ (\forall j \in \{1, 2, \dots, k\}, j \neq i) [\alpha(G_{2j-1}) + \ell_{2j} &= \alpha(G_{2j}) + \ell_{2j-1}]. \end{aligned}$$

Die *disjunkte Vereinigung* zweier (knotendisjunkter) Graphen G und H ist definiert als der Graph $G \cup H$ mit der Knotenmenge $V(G \cup H) = V(G) \cup V(H)$ und der Kantenmenge $E(G \cup H) = E(G) \cup E(H)$. Wichtig ist hier die einfache Feststellung, dass $\alpha(G \cup H) = \alpha(G) + \alpha(H)$. Ohne die Allgemeinheit zu beschränken, kann man annehmen, dass die Graphen G_1, G_2, \dots, G_{2k} paarweise knotendisjunkt sind. Definiere die Summe der ℓ_j und die disjunkte Vereinigung der G_j mit ungeradem (beziehungsweise mit geradem) Index j durch:

$$\begin{aligned} \ell_{\text{odd}} &= \sum_{1 \leq i \leq k} \ell_{2i-1} \quad \text{und} \quad \ell_{\text{even}} = \sum_{1 \leq i \leq k} \ell_{2i}; \\ G_{\text{odd}} &= \bigcup_{1 \leq i \leq k} G_{2i-1} \quad \text{und} \quad G_{\text{even}} = \bigcup_{1 \leq i \leq k} G_{2i}. \end{aligned}$$

Für $m > 0$ sei H_m der Graph, der aus m isolierten Knoten besteht, und H_m sei disjunkt zu G_{odd} und G_{even} . Dann ist $\alpha(H_m) = m$. Definiere die Reduktion:

$$f(\langle \varphi_1, \varphi_2, \dots, \varphi_{2k} \rangle) = \langle G_{\text{odd}} \cup H_{\ell_{\text{even}}}, G_{\text{even}} \cup H_{\ell_{\text{odd}}} \rangle.$$

Offenbar kann f in Polynomialzeit berechnet werden. Damit (5.24) erfüllt ist, bleibt zu zeigen, dass:

$$\|\{i \mid \varphi_i \in 3\text{-SAT}\}\| \text{ ist gerade} \iff \alpha(G_{\text{odd}} \cup H_{\ell_{\text{even}}}) = \alpha(G_{\text{even}} \cup H_{\ell_{\text{odd}}}).$$

Von links nach rechts:

$$\begin{aligned} & \|\{i \mid \varphi_i \in 3\text{-SAT}\}\| \text{ ist gerade} \\ \implies & (\forall i \in \{1, 2, \dots, k\}) [\alpha(G_{2i-1}) + \ell_{2i} = \alpha(G_{2i}) + \ell_{2i-1}] \\ \implies & \sum_{1 \leq i \leq k} (\alpha(G_{2i-1}) + \ell_{2i}) = \sum_{1 \leq i \leq k} (\alpha(G_{2i}) + \ell_{2i-1}) \\ \implies & \alpha(G_{\text{odd}}) + \ell_{\text{even}} = \alpha(G_{\text{even}}) + \ell_{\text{odd}} \\ \implies & \alpha(G_{\text{odd}} \cup H_{\ell_{\text{even}}}) = \alpha(G_{\text{even}} \cup H_{\ell_{\text{odd}}}). \end{aligned}$$

Von rechts nach links:

$$\begin{aligned} & \|\{i \mid \varphi_i \in 3\text{-SAT}\}\| \text{ ist ungerade} \\ \implies & (\exists i \in \{1, 2, \dots, k\}) [\alpha(G_{2i-1}) + \ell_{2i} - 1 = \alpha(G_{2i}) + \ell_{2i-1} \text{ und} \\ & (\forall j \in \{1, 2, \dots, k\}, j \neq i) [\alpha(G_{2j-1}) + \ell_{2j} = \alpha(G_{2j}) + \ell_{2j-1}]] \\ \implies & -1 + \sum_{1 \leq j \leq k} (\alpha(G_{2j-1}) + \ell_{2j}) = \sum_{1 \leq j \leq k} (\alpha(G_{2j}) + \ell_{2j-1}) \\ \implies & \alpha(G_{\text{odd}}) + \ell_{\text{even}} - 1 = \alpha(G_{\text{even}}) + \ell_{\text{odd}} \\ \implies & \alpha(G_{\text{odd}} \cup H_{\ell_{\text{even}}}) - 1 = \alpha(G_{\text{even}} \cup H_{\ell_{\text{odd}}}) \\ \implies & \alpha(G_{\text{odd}} \cup H_{\ell_{\text{even}}}) \neq \alpha(G_{\text{even}} \cup H_{\ell_{\text{odd}}}). \end{aligned}$$

Somit gilt (5.24). Nach Lemma 5.43 ist $\text{IN-Equ } \Theta_2^P$ -hart, was zu beweisen war. \square

Lemma 5.43 ist ein universales Werkzeug zur Fabrikation von Θ_2^P -Vollständigkeitsresultaten. Sei etwa **Favorite** unser Lieblingsproblem unter all den NP-vollständigen Problemen, in deren Definition ein Parameter (wie das k in IS gemäß Definition 3.52) eingeht. Definiere nun – analog zu **IN-Odd**, **IN-Equ** und **IN-Geq** aus Definition 5.38 – die zu **Favorite** gehörigen Varianten **Favorite-Odd**, **Favorite-Equ** und **Favorite-Geq** und wende die Wagner-Technik aus Lemma 5.43 an, um die Θ_2^P -Vollständigkeit dieser Varianten zu zeigen. In Abschnitt 5.9 werden weitere Anwendungen dieser Technik angegeben.

Natürlich sind Probleme wie zum Beispiel **Favorite-Odd**, **Favorite-Equ** und **Favorite-Geq** nicht unbedingt übermäßig natürlich. Es gibt jedoch auch ganz natürliche Probleme, deren Θ_2^P -Vollständigkeit mit dieser Technik gezeigt werden konnte. Einige davon haben mit der Komplexität von bestimmten Problemen für Wahlsysteme zu tun, die in einem als „*Computational Social Choice*“ bezeichneten Gebiet auftreten. Insbesondere wurde Lemma 5.43 erfolgreich angewandt, um für gewisse Wahlsysteme zu zeigen, dass das Problem, die Gewinner einer gegebenen Wahl zu bestimmen, Θ_2^P -vollständig ist. Der Rest dieses Abschnitts stellt ein solches Ergebnis vor, nämlich für das Wahlsystem von Young, und klassifiziert die Komplexität des entsprechenden Gewinnerproblems: Young-Gewinner zu bestimmen ist vollständig für parallelen Zugriff auf NP. Der Beweis dieses Resultats von Rothe, Spakowski und Vogel [RSV02, RSV03] wendet Lemma 5.43 nicht explizit

an. Jedoch wird es implizit verwendet, indem nämlich eine Reduktion von dem Problem IN-Geq konstruiert wird, welches nach Satz 5.44 Θ_2^P -vollständig ist, und der Beweis dieses Satzes ist eine Anwendung von Lemma 5.43.

Bevor wir uns der eigentlichen Komplexitätsanalyse zuwenden, soll eine kurze Abschweifung in die Social-Choice-Theorie einführen. Mehr über dieses faszinierende Gebiet und insbesondere über seinen Bezug zur *Computational Social Choice* erfährt man in den Übersichtsartikeln von E. und L. Hemaspaandra [HH00] und von Faliszewski et al. [FHHR], siehe auch Abschnitt 5.9.

5.3.1 Eine kurze Abschweifung in die Theorie der Wahlsysteme

Das folgende Zitat stammt vom Klappentext eines Buches von Saari [Saa01] über die Mathematik des Wählens:

„What does the 2000 U.S. Presidential Election have in common with selecting a textbook for a calculus course in your department? Was Ralph Nader’s influence on the election of George W. Bush greater than the now-famous chads? In Chaotic Elections!, Don Saari analyzes these questions, placing them in the larger context of voting systems in general. His analysis shows that the fundamental problems with the 2000 presidential election are not with the courts, recounts or defective ballots, but are caused by the very way Americans vote for president.“

(Aus „*Chaotic Elections!*“ von Don Saari, American Mathematical Society, 2001)

Anders gesagt: Bevor man irgendjemanden oder irgendetwas für den womöglich unerwünschten Ausgang einer Wahl verantwortlich macht, sollte man sich erst einmal das verwendete Wahlsystem und die zugrunde liegende Mathematik genau ansehen. Ist es ein „faires“ System? Kann es überhaupt „faire“ Wahlsysteme geben? Und was sind die grundlegenden Eigenschaften von Wahlsystemen?

Fragen wie diese werden seit Jahrhunderten in der Social-Choice-Theorie, in den Politik- und Wirtschaftswissenschaften und auch in der Mathematik untersucht. Motiviert durch Anwendungen der Theorie des Wählens und der Präferenzaggregation auf automatisierte Verfahren, wie sie beispielsweise bei elektronischen Wahlen, Multi-Agenten-Systemen,⁴ Page-Ranking-Algorithmen für Suchmaschinen im Internet, E-Kommerz und elektronischen Auktionen auftreten, spielen Wahlsysteme jedoch auch in der Informatik eine immer wichtigere Rolle.

In der Social-Choice-Theorie untersucht man Eigenschaften, die jedes „vernünftige“ Wahlsystem besitzen sollte. Dazu gehören etwa die Monotonität, das so genannte Pareto-Prinzip, die so genannte Unabhängigkeit irrelevanter Alternativen, und natürlich sollte ein Wahlsystem nicht diktatorisch sein. Eines der bemerkenswertesten Resultate der Social-Choice-Theorie ist das berühmte Unmöglichkeitstheorem

⁴ Das sind Systeme, die aus vielen voneinander unabhängigen Software-Programmen (den „Agenten“) bestehen, die miteinander kommunizieren und gemeinsam Aufgaben lösen. Dabei müssen sie manchmal gemeinsam Entscheidungen treffen, also eine Auswahl aus mehreren möglichen Alternativen. Da individuelle Agenten natürlich unterschiedliche, oft im Konflikt befindliche Präferenzen über diese Alternativen haben können, werden Wahlverfahren verwendet, um einen gemeinsamen Konsens zu erzielen, mit dem alle beteiligten Agenten „einverstanden“ sind. Diesen Prozess bezeichnet man als „Präferenzaggregation“.

von Arrow [Arr63], welches sagt, dass die gerade erwähnten vier Eigenschaften logisch inkonsistent sind, sobald es mindestens drei Kandidaten (oder Alternativen) gibt. Dies kann man so interpretieren, dass es kein „faires“ Wahlsystem geben kann, also kein Wahlsystem, das diesem Mindestanspruch an Fairness genügt.

Um Wahlsysteme formal zu beschreiben, benötigen wir Kandidaten (oder Alternativen) und Wähler (oder Agenten) sowie eine Regel, die sagt, wer eine Wahl gemäß der Wählerpräferenzen über die Kandidaten gewonnen hat. Eine Wahl stellt man dabei als ein *Präferenzprofil* dar, ein Paar $\langle C, V \rangle$ also, wobei C eine endliche Menge von Kandidaten ist und V eine endliche Multimenge⁵ der Wählerpräferenzen über C . Wir nehmen an, dass jeder Wähler strikte Präferenzen über die Kandidaten hat. Formal ist die Präferenzordnung eines jeden Wählers eine strikte (d.h. irreflexive und antisymmetrische), transitive und vollständige (d.h., alle Kandidaten kommen in jeder Wählerpräferenz vor) Relation auf C .

Ein *Wahlsystem* ist eine Regel, mittels derer bestimmt wird, wer eine Wahl gewonnen hat. Im Gegensatz zu so genannten *social choice functions* (kurz SCF), die üblicherweise Präferenzprofile auf vollständige Präferenzlisten abbilden,⁶ trennen Wahlsysteme nur die Gewinner von den Nichtgewinnern. Das heißt, ein Wahlsystem f bildet jedes gegebene Präferenzprofil $\langle C, V \rangle$ auf die Menge $f(\langle C, V \rangle) \subseteq C$ der Gewinner ab (die auf englisch als *choice set* bezeichnet wird).

Beispielsweise sagt die (einfache) *Majoritätsregel* (auch als *Mehrheitsentscheid* bezeichnet), dass ein Kandidat A einen Kandidaten B genau dann *siegt*, wenn eine echte Mehrheit der Wähler A gegenüber B den Vorzug gibt. Um gemäß der Majoritätsregel eine Wahl zu gewinnen, muss ein Kandidat jeden anderen Kandidaten im paarweisen Vergleich schlagen. Ein solcher Kandidat heißt der *Condorcet-Gewinner* der Wahl.

Beispiel 5.45 (Condorcet-Paradox). Im Jahr 1785 veröffentlichte Marie Jean Antoine Nicolas de Caritat, der Marquis de Condorcet, einen Aufsatz [Con85], in dem er feststellte, dass bei mindestens drei Kandidaten, etwa a , b und c , die Majoritätsregel Zyklen hervorbringen kann. Sein Beispiel besteht aus den folgenden drei Wählern:

$$\begin{aligned} \text{Wähler 1: } & a > b > c; \\ \text{Wähler 2: } & b > c > a; \\ \text{Wähler 3: } & c > a > b. \end{aligned}$$

In diesem Beispiel gilt: a schlägt b und b schlägt c , aber c schlägt a . Das heißt, auch wenn sich individuelle Wähler rational verhalten, was man oft durch eine transitive (also nichtzyklische) Präferenzordnung ausdrückt, kann sich die Gesellschaft als Ganzes irrational verhalten und Zyklen bilden. Condorcet-Gewinner müssen demnach nicht immer existieren. Dies ist als das Paradox von Condorcet bekannt.

⁵ Eine *Multimenge* ist eine Liste von Elementen, in der jedes Element mehr als einmal auftreten kann. Präferenzprofile werden als Multimengen dargestellt, da verschiedene Wähler natürlich dieselben Präferenzen über die Kandidaten haben können.

⁶ Anders als bei individuellen Präferenzen, die strikt sind, erlaubt man in einer solchen Präferenzliste auch Indifferenzen zwischen Kandidaten.

Das *Condorcet-Prinzip* sagt, dass die Gewinner einer Wahl nach der Majoritätsregel zu bestimmen sind. Eine *Condorcet-SCF* ist eine SCF, die das Condorcet-Prinzip in dem Sinn respektiert, dass der Condorcet-Gewinner gewählt wird, sofern er existiert. Condorcet-Gewinner sind, falls sie überhaupt existieren, stets eindeutig bestimmt. Viele Condorcet-SCF sind in der Social-Choice-Literatur vorgeschlagen worden; eine Überblick über die wichtigsten erhält man in Fishburns Arbeit [Fis77]. Jede Condorcet-SCF erweitert das Condorcet-Prinzip in einer Weise, die das beunruhigende Merkmal der Majoritätsregel (dass nämlich nicht immer ein Sieger feststeht) vermeidet. Im Folgenden konzentrieren wir uns auf das Wahlsystem von Young [You77], welches eine der in [Fis77] aufgeführten Condorcet-SCF ist.

5.3.2 Gewinnerproblem für Young-Wahlen

Der Zugang H. Youngs [You77] zur Erweiterung des Condorcet-Prinzips beruht auf einer Veränderung der Präferenzprofile. Seinem Vorschlag gemäß folgen wir dem Condorcet-Prinzip dann am treuesten, wenn wir fordern, dass eine Wahl von einem solchen Kandidaten gewonnen wird, der durch die Entfernung von möglichst wenigen Wählern zum Condorcet-Gewinner gemacht werden kann. Wir definieren nun die folgenden Entscheidungsprobleme, die beide mit dem Young-Wahlsystem assoziiert sind, und untersuchen dann ihre Komplexität.

Definition 5.46 (Young Winner und Young Ranking). *Definiere für jeden Kandidaten c in einem gegebenen Präferenzprofil $\langle C, V \rangle$ den Young-Score, bezeichnet mit $\text{YScore}(C, c, V)$, als die Größe einer größten Teilmultimenge von V , für die c ein Condorcet-Gewinner ist. Ein Young-Gewinner ist ein jeder Kandidat mit maximalem Young-Score. Definiere die Probleme:*

$$\begin{aligned} \text{YoungWinner} &= \left\{ \langle C, c, V \rangle \middle| \begin{array}{l} \langle C, V \rangle \text{ ist ein Präferenzprofil, } c \in C \text{ ist ein} \\ \text{ausgezeichneter Kandidat und für jedes } d \in C \\ \text{ist } \text{YScore}(C, c, V) \geq \text{YScore}(C, d, V) \end{array} \right\}; \\ \text{YoungRanking} &= \left\{ \langle C, c, d, V \rangle \middle| \begin{array}{l} \langle C, V \rangle \text{ ist ein Präferenzprofil und } c \text{ und } d \\ \text{sind ausgezeichnete Kandidaten in } C, \text{ so} \\ \text{dass } \text{YScore}(C, c, V) \geq \text{YScore}(C, d, V) \end{array} \right\}. \end{aligned}$$

Die Sätze 5.49 und 5.51 unten zeigen, dass die oben definierten Probleme Θ_2^P -hart sind. Dabei werden Reduktionen vom Problem Max-SetPacking-Geq angegeben. Zur Erinnerung: Definition 3.64 führte das mit Max-SetPacking-Geq verwandte Problem SetPacking ein, welches nach Satz 3.65 NP-vollständig ist.

Definition 5.47 (Maximum Set Packing Compare). *Für eine Familie \mathcal{S} von Mengen bezeichne $\kappa(\mathcal{S})$ die maximale Anzahl von paarweise disjunkten Mengen in \mathcal{S} . Definiere das Problem:*

Max-SetPacking-Geq

$$= \left\{ \langle U_1, U_2, \mathcal{S}_1, \mathcal{S}_2 \rangle \middle| \begin{array}{l} U_i \text{ ist eine endliche Menge und } \mathcal{S}_i \subseteq \mathfrak{P}(U_i), i \in \{1, 2\}, \\ \text{ist eine Familie nichtleerer Mengen mit } \kappa(\mathcal{S}_1) \geq \kappa(\mathcal{S}_2) \end{array} \right\}.$$

Lemma 5.48. Max-SetPacking-Geq ist Θ_2^P -vollständig.

Beweis. Es ist nicht schwer, eine Reduktion auf Max-SetPacking-Geq vom Problem IN-Geq anzugeben, das nach Satz 5.44 Θ_2^P -vollständig ist. Die Details der Reduktion werden dem Leser als Übung 5.25(c) überlassen. \square

Satz 5.49. YoungRanking ist Θ_2^P -vollständig.

Beweis von Satz 5.49. Dass YoungRanking, ebenso wie YoungWinner, in Θ_2^P liegt, ist leicht zu sehen. Die Details dieses Beweises werden als Übung 5.26 dem Leser überlassen.

Um die Θ_2^P -Härte zu zeigen, wird eine polynomialzeit-beschränkte Many-one-Reduktion vom Problem Max-SetPacking-Geq angegeben. Gegeben seien also zwei Mengen, $U_1 = \{x_1, x_2, \dots, x_m\}$ und $U_2 = \{y_1, y_2, \dots, y_n\}$, und zwei Familien \mathcal{S}_1 und \mathcal{S}_2 nichtleerer Teilmengen von U_1 bzw. U_2 . Wie oben erwähnt ist $\kappa(\mathcal{S}_i)$, $i \in \{1, 2\}$, die maximale Anzahl paarweise disjunkter Mengen in \mathcal{S}_i . Ohne Beschränkung der Allgemeinheit darf angenommen werden, dass $\kappa(\mathcal{S}_i) > 2$ gilt.

Das Ziel ist, ein Präferenzprofil $\langle C, V \rangle$ mit den ausgezeichneten Kandidaten c und d in C so zu konstruieren, dass gilt:

$$\text{YScore}(C, c, V) = 2 \cdot \kappa(\mathcal{S}_1) + 1; \quad (5.27)$$

$$\text{YScore}(C, d, V) = 2 \cdot \kappa(\mathcal{S}_2) + 1. \quad (5.28)$$

Zur Kandidatenmenge C gehören:

- die beiden ausgezeichneten Kandidaten c und d ,
- je ein Kandidat x_i für jedes Element x_i von U_1 ,
- je ein Kandidat y_i für jedes Element y_i von U_2 und
- zwei Hilfskandidaten, a und b .

Zu den Wählern V gehören:

- *Wähler, die \mathcal{S}_1 repräsentieren:* Für jede Menge $E \in \mathcal{S}_1$ gibt es einen einzelnen Wähler v_E folgender Art:
 - Nummeriere E als $\{e_1, e_2, \dots, e_{\|E\|}\}$ (wobei $E \subseteq \{x_1, x_2, \dots, x_m\}$, d.h., die Kandidaten e_i entstehen der Einfachheit halber durch Umbenennen) und nummeriere das Komplement $\bar{E} = U_1 - E$ als $\{\bar{e}_1, \bar{e}_2, \dots, \bar{e}_{m-\|E\|}\}$.
 - Zur leichteren Lesbarkeit der Präferenzordnungen werden die folgenden Bezeichnungen verwendet:

$$\begin{aligned} „\mathbf{E}“ &\text{ repräsentiert } „e_1 > e_2 > \dots > e_{\|E\|}“; \\ „\overline{\mathbf{E}}“ &\text{ repräsentiert } „\bar{e}_1 > \bar{e}_2 > \dots > \bar{e}_{m-\|E\|}“; \\ „\mathbf{U}_1“ &\text{ repräsentiert } „x_1 > x_2 > \dots > x_m“; \\ „\mathbf{U}_2“ &\text{ repräsentiert } „y_1 > y_2 > \dots > y_n“; \end{aligned}$$

- Der E entsprechende Wähler v_E hat die Präferenzordnung:

$$\mathbf{E} > a > c > \overline{\mathbf{E}} > \mathbf{U}_2 > b > d. \quad (5.29)$$

- Zusätzlich gibt es zwei Wähler mit der Präferenzordnung:

$$c > \mathbf{U}_1 > a > \mathbf{U}_2 > b > d. \quad (5.30)$$

- Schließlich gibt es $\|\mathcal{S}_1\| - 1$ Wähler mit der Präferenzordnung:

$$\mathbf{U}_1 > c > a > \mathbf{U}_2 > b > d. \quad (5.31)$$

- *Wähler, die \mathcal{S}_2 repräsentieren:* Der Fall \mathcal{S}_2 wird analog behandelt, wobei \mathcal{S}_1 , U_1 , x_i , c , a , E , e_j und \bar{e}_k die Rollen tauschen mit \mathcal{S}_2 , U_2 , y_i , d , b , F , f_j und \bar{f}_k . Genauer gibt es für jede Menge $F \in \mathcal{S}_2$ einen einzelnen Wähler v_F folgender Art:
 - Nummeriere F als $\{f_1, f_2, \dots, f_{\|F\|}\}$ (wobei $F \subseteq \{y_1, y_2, \dots, y_n\}$, d.h., die Kandidaten f_j entstehen wieder einfach durch Umbenennen) und nummeriere das Komplement $\bar{F} = U_1 - F$ als $\{\bar{f}_1, \bar{f}_2, \dots, \bar{f}_{n-\|F\|}\}$.
 - Zur leichteren Lesbarkeit der Präferenzordnungen werden, zusätzlich zu den oben eingeführten Abkürzungen, die folgenden Bezeichnungen verwendet:

„ \mathbf{F} “ repräsentiert „ $f_1 > f_2 > \dots > f_{\|F\|}$ “;
 „ $\bar{\mathbf{F}}$ “ repräsentiert „ $\bar{f}_1 > \bar{f}_2 > \dots > \bar{f}_{n-\|F\|}$ “.

- Der F entsprechende Wähler v_F hat die Präferenzordnung:

$$\mathbf{F} > b > d > \bar{\mathbf{F}} > \mathbf{U}_1 > a > c. \quad (5.32)$$

- Zusätzlich gibt es zwei Wähler mit der Präferenzordnung:

$$d > \mathbf{U}_2 > b > \mathbf{U}_1 > a > c. \quad (5.33)$$

- Schließlich gibt es $\|\mathcal{S}_2\| - 1$ Wähler mit der Präferenzordnung:

$$\mathbf{U}_2 > d > b > \mathbf{U}_1 > a > c. \quad (5.34)$$

Zu zeigen ist zunächst (5.27): $\text{YScore}(C, c, V) = 2 \cdot \kappa(\mathcal{S}_1) + 1$.

Seien $\kappa(\mathcal{S}_1)$ paarweise disjunkte Teilmengen $E_1, E_2, \dots, E_{\kappa(\mathcal{S}_1)} \in \mathcal{S}_1$ von U_1 gegeben. Betrachte die Teilmultimenge \hat{V} von V , zu der die folgenden Wähler gehören:

- Jeder der Menge E_i entsprechende Wähler v_{E_i} , $1 \leq i \leq \kappa(\mathcal{S}_1)$;
- die beiden in (5.30) angegebenen Wähler;
- $\kappa(\mathcal{S}_1) - 1$ Wähler der Form (5.31).

Offenbar gilt $\|\hat{V}\| = 2 \cdot \kappa(\mathcal{S}_1) + 1$. Da eine echte Mehrheit der Wähler in \hat{V} den Kandidaten c einem jeden anderen Kandidaten vorziehen, ist c Condorcet-Gewinner in $\langle C, \hat{V} \rangle$. Somit gilt

$$\text{YScore}(C, c, V) \geq 2 \cdot \kappa(\mathcal{S}_1) + 1.$$

Um umgekehrt zu zeigen, dass $\text{YScore}(C, c, V) \leq 2 \cdot \kappa(\mathcal{S}_1) + 1$ gilt, benötigen wir das folgende Lemma.

Lemma 5.50. Für λ mit $3 < \lambda \leq \|\mathcal{S}_1\| + 1$ sei V_λ eine beliebige Teilmultimenge von V , so dass V_λ genau λ Wähler der Form (5.30) oder (5.31) enthält und c Condorcet-Gewinner in $\langle C, V_\lambda \rangle$ ist. Dann enthält V_λ genau $\lambda - 1$ Wähler der Form (5.29) und keine Wähler der Form (5.32), (5.33) oder (5.34). Weiterhin repräsentieren die $\lambda - 1$ Wähler der Form (5.29) in V_λ paarweise disjunkte Mengen aus \mathcal{S}_1 .

Beweis von Lemma 5.50. Für festes λ sei V_λ wie im Lemma angegeben. Betrachte die Teilmultimenge von V_λ , die aus den λ Wählern der Form (5.30) oder (5.31) besteht. Einem jeden Kandidaten x_i , $1 \leq i \leq m$, wird von mindestens $\lambda - 2$ Wählern der Form (5.31) der Vorzug gegenüber c gegeben. Da c Condorcet-Gewinner in $\langle C, V_\lambda \rangle$ ist, gibt es für jedes x_i mindestens $\lambda - 1 > 2$ Wähler in V_λ , die c gegenüber x_i vorziehen. Nach Konstruktion müssen diese Wähler von der Form (5.29) oder (5.30) sein. Da es höchstens zwei Wähler der Form (5.30) gibt, existiert mindestens ein Wähler der Form (5.29); sei \tilde{v} ein solcher Wähler. Da die Wähler der Form (5.29) \mathcal{S}_1 repräsentieren und \mathcal{S}_1 nur nichtleere Mengen enthält, gibt es einen Kandidaten x_j , den \tilde{v} gegenüber c vorzieht. Insbesondere muss der Condorcet-Gewinner c den Kandidaten x_j in $\langle C, V_\lambda \rangle$ schlagen und benötigt daher mehr als $(\lambda - 2) + 1$ Stimmen der Form (5.29) oder (5.30). Es gibt höchstens zwei Wähler der Form (5.30). Folglich muss c gegenüber x_j von mindestens $\lambda - 2$ Wählern der Form (5.29) vorgezogen werden, die von \tilde{v} verschieden sind. Zusammengefasst enthält V_λ mindestens $\lambda - 1$ Wähler der Form (5.29).

Da andererseits c Condorcet-Gewinner in $\langle C, V_\lambda \rangle$ ist, muss c insbesondere a schlagen, das c bei den λ Wählern der Form (5.30) oder (5.31) unterliegt und von allen anderen Wählern den Vorzug gegenüber c erhält. Folglich enthält V_λ höchstens $\lambda - 1$ Wähler der Form (5.29), (5.32), (5.33) oder (5.34). Somit enthält V_λ genau $\lambda - 1$ Wähler der Form (5.29) und keinen Wähler der Form (5.32), (5.33) oder (5.34).

Für einen Widerspruch nimm an, dass es einen Kandidat x_j gibt, dem von mehr als einem Wähler der Form (5.29) in V_λ der Vorzug gegenüber c gegeben wird. Dann gilt:

- höchstens zwei Wähler der Form (5.30) und höchstens $(\lambda - 1) - 2 = \lambda - 3$ Wähler der Form (5.29) ziehen c gegenüber x_j vor;
- mindestens $\lambda - 2$ Wähler der Form (5.31) und mindestens zwei Wähler der Form (5.29) ziehen x_j gegenüber c vor.

Da im direkten Vergleich c somit höchstens $\lambda - 1$ Stimmen und x_j mindestens λ Stimmen in V_λ erhält, ist c kein Condorcet-Gewinner in $\langle C, V_\lambda \rangle$, ein Widerspruch. Folglich gibt es für jeden Kandidaten x_i , $1 \leq i \leq m$, höchstens einen Wähler der Form (5.29) in V_λ , der x_i gegenüber c vorzieht. Das jedoch bedeutet, dass die $\lambda - 1$ Wähler der Form (5.29) in V_λ paarweise disjunkte Mengen aus \mathcal{S}_1 repräsentieren, womit das Lemma bewiesen ist. \square

Lemma 5.50

Zur Fortsetzung des Beweises von Satz 5.49 sei $k = \text{YScore}(C, c, V)$. Sei weiter $\hat{V} \subseteq V$ eine Teilmultimenge der Größe k , so dass c der Condorcet-Gewinner in $\langle C, \hat{V} \rangle$ ist. Angenommen, es gibt genau $\lambda \leq \|\mathcal{S}_1\| + 1$ Wähler der Form (5.30) oder (5.31) in \hat{V} . Da c als Condorcet-Gewinner von $\langle C, \hat{V} \rangle$ insbesondere a schlagen muss, gilt

$\lambda \geq \lceil \frac{k+1}{2} \rceil$, wobei für jede reelle Zahl r die Schreibweise $\lceil r \rceil$ die kleinste ganze Zahl s mit $s \geq r$ bezeichnet. Nach unserer Annahme, dass $\kappa(\mathcal{S}_1) > 2$ ist, folgt aus $k \geq 2 \cdot \kappa(\mathcal{S}_1) + 1$, dass $\lambda > 3$ gilt.

Gemäß Lemma 5.50 gibt es somit genau $\lambda - 1$ Wähler der Form (5.29) in \hat{V} , welche paarweise disjunkte Mengen aus \mathcal{S}_1 repräsentieren, und \hat{V} enthält keine Wähler der Form (5.32), (5.33) oder (5.34). Also ist $k = 2 \cdot \lambda - 1$ ungerade, und es gilt:

$$\frac{k-1}{2} = \lambda - 1 \leq \kappa(\mathcal{S}_1),$$

womit (5.27) gezeigt ist. (5.28) kann analog bewiesen werden. Es folgt:

$$\kappa(\mathcal{S}_1) \geq \kappa(\mathcal{S}_2) \iff \text{YScore}(C, c, V) \geq \text{YScore}(C, d, V).$$

Der Beweis von Satz 5.49 ist somit abgeschlossen. □ Satz 5.49

Satz 5.51. *YoungWinner ist Θ_2^P -vollständig.*

Beweis. Dass YoungWinner in Θ_2^P liegt, wurde bereits im Beweis von Satz 5.49 erwähnt. Um die Θ_2^P -Härte zu beweisen, modifizieren wir die Reduktion aus dem Beweis von Satz 5.49 zu einer Reduktion vom Problem Max-SetPacking-Geq auf das Problem YoungWinner wie folgt. Sei $\langle C, V \rangle$ das Präferenzprofil, das im Beweis von Satz 5.49 konstruiert wurde, wobei c und d die ausgezeichneten Kandidaten sind. Dieses Präferenzprofil wird nun so geändert, dass alle anderen Kandidaten schlechter als c und d dastehen. Ausgehend von $\langle C, V \rangle$ wird also ein neues Präferenzprofil $\langle D, W \rangle$ konstruiert. Die neue Kandidatenmenge D erhält man, indem jeder Kandidat $g \in C$ außer c und d durch $\|V\|$ neue Kandidaten $g^1, g^2, \dots, g^{\|V\|}$ ersetzt wird.

Die neue Multimenge W von Wählern erhält man, indem jedes Vorkommen eines Kandidaten g im i -ten Wähler von V ersetzt wird durch

$$g^{i \bmod \|V\|} > g^{i+1 \bmod \|V\|} > g^{i+2 \bmod \|V\|} > \dots > g^{i+\|V\|-1 \bmod \|V\|}.$$

Sei \tilde{V} eine beliebige Teilmultimenge von V , und sei \tilde{W} die \tilde{V} entsprechende Teilmultimenge von W . Es ist leicht zu sehen, dass c genau dann der Condorcet-Gewinner in \tilde{V} ist, wenn c der Condorcet-Gewinner in \tilde{W} ist. Somit ändert der Übergang von $\langle C, V \rangle$ zu $\langle D, W \rangle$ nicht den Young-Score von c oder d . Andererseits ist der Young-Score eines jeden anderen Kandidaten nun höchstens 1. Also gibt es keinen Kandidaten b mit $\text{YScore}(D, b, W) > \text{YScore}(D, c, W)$ oder $\text{YScore}(D, b, W) > \text{YScore}(D, d, W)$. Folglich gilt $\kappa(\mathcal{S}_1) \geq \kappa(\mathcal{S}_2)$ genau dann, wenn c ein Young-Gewinner der Wahl $\langle D, W \rangle$ ist. □

5.4 Frage-Hierarchien über NP

NP und coNP bilden jeweils die erste Stufe sowohl der Polynomialzeit-Hierarchie als auch der booleschen Hierarchie über NP. In diesem Abschnitt wird gezeigt, dass

letztere vollständig in der zweiten Stufe der Polynomialzeit-Hierarchie enthalten ist, und zwar sogar in Θ_2^P . Nach Definition ist Θ_2^P die Klasse $P^{NP[\log]}$, zu der die von einer DPOTM mit logarithmisch vielen Turing-Fragen an ein NP-Orakel lösbarer Probleme gehören. Entsprechend enthält die Klasse $P^{NP[\mathcal{O}(1)]}$ genau die Probleme L , die eine DPOTM M^A mit Zugriff auf ein NP-Orakel A lösen kann, wobei höchstens konstant viele Orakelfragen nach Art einer Turing-Reduktion gestellt werden. Analog dazu kann man Mengenklassen definieren, die von einer DPOTM lösbar sind, die ihrem NP-Orakel eine beschränkte Anzahl von Fragen *parallel* stellt, d.h., der Orakelzugriff erfolgt nach Art einer Truth-table-Reduktion. Mit diesen Konzepten kann man die *Frage-Hierarchie* und die *parallele Frage-Hierarchie* über NP definieren.

Definition 5.52 (Frage-Hierarchie und Parallele Frage-Hierarchie über NP).

1. Die Frage-Hierarchie über NP ist definiert durch:

$$\begin{aligned} P^{NP[k]} &= \left\{ L \mid L = L(M^{\text{SAT}}) \text{ für eine DPOTM } M, \text{ die nicht mehr als } k \text{ Orakelfragen nach Art einer Turing-Reduktion stellt} \right\}, \\ P^{NP[\mathcal{O}(1)]} &= \bigcup_{k \in \mathbb{N}} P^{NP[k]} \quad \text{und} \quad P^{NP[\log]} = \bigcup_{k \in \mathcal{O}(\log)} P^{NP[k]}, \end{aligned}$$

wobei im letzteren Fall $k \in \mathcal{O}(\log)$ eine Funktion der Eingabegröße ist.

2. Die parallele Frage-Hierarchie über NP ist definiert durch:

$$\begin{aligned} P_{k\text{-tt}}^{NP} &= \left\{ L \mid L = L(M^{\text{SAT}}) \text{ für eine DPOTM } M, \text{ die nicht mehr als } k \text{ Orakelfragen nach Art einer Truth-table-Reduktion stellt} \right\}, \\ P_{\text{bit}}^{NP} &= \bigcup_{k \in \mathbb{N}} P_{k\text{-tt}}^{NP} \quad \text{und} \quad P_{\text{tt}}^{NP} = \bigcup_{p \in \text{IPol}} P_{p\text{-tt}}^{NP}, \end{aligned}$$

wobei im letzteren Fall $p \in \text{IPol}$ eine Funktion der Eingabegröße ist.

3. Für jede Orakelmenge A und für jede Klasse \mathcal{C} von Orakelmengen sind die Klassen $P^{A[k]}$, $P_{k\text{-tt}}^A$, $P_{\mathcal{C}[k]}$, $P_{k\text{-tt}}^{\mathcal{C}}$ usw. analog definiert.

Da eine DPOTM höchstens polynomiell viele Fragen stellen kann, ist P_{tt}^{NP} nichts anderes als die \leq_{tt}^P -Hülle von NP. Das folgende Resultat zeigt die Beziehungen zwischen der booleschen Hierarchie und den Frage-Hierarchien über NP. Insbesondere ergibt sich, dass diese drei Hierarchien ineinander verflochten sind, woraus folgt, dass sie alle gemeinsam stehen oder fallen (d.h. gemeinsam unendlich sind oder aber auf eine endliche Stufe kollabieren).

Satz 5.53. 1. $P_{k\text{-tt}}^{NP} = P^{A[1]}$ für eine Menge A , die \leq_m^P -vollständig in $\text{BH}_k(\text{NP})$ ist.
 2. $P^{NP[k]} = P_{(2^k-1)\text{-tt}}^{NP}$.
 3. $\text{BH}_k(\text{NP}) \cup \text{coBH}_k(\text{NP}) \subseteq P_{k\text{-tt}}^{NP} \subseteq \text{BH}_{k+1}(\text{NP}) \cap \text{coBH}_{k+1}(\text{NP})$.
 4. $\text{BH}_{2^k-1}(\text{NP}) \cup \text{coBH}_{2^k-1}(\text{NP}) \subseteq P^{NP[k]} \subseteq \text{BH}_{2^k}(\text{NP}) \cap \text{coBH}_{2^k}(\text{NP})$.

Beweis. 1. Definiere für festes $k \geq 1$ das Problem

$$\text{Odd-}k\text{-SAT} = \left\{ \langle \varphi_1, \varphi_2, \dots, \varphi_k \rangle \mid \begin{array}{l} \varphi_1, \varphi_2, \dots, \varphi_k \text{ sind boolesche Formeln,} \\ \text{so dass } |\{i \mid \varphi_i \in \text{SAT}\}| \text{ ungerade ist} \end{array} \right\}.$$

Offenbar ist Odd- k -SAT in $\text{BH}_k(\text{NP})$. Nach Lemma 5.11 ist Odd- k -SAT sogar eine $\text{BH}_k(\text{NP})$ -vollständige Menge; siehe Übung 5.29.

Um zu zeigen, dass $P_{k\text{-tt}}^{\text{NP}} \subseteq P^{A[1]}$ für $A = \text{Odd-}k\text{-SAT}$ gilt, sei L eine beliebige Menge in $P_{k\text{-tt}}^{\text{NP}}$. Nach Definition gibt es eine Menge $B \in \text{NP}$ und eine in Polynomialzeit berechenbare Funktion f , so dass für jede Eingabe x :

- $f(x) = \langle \tau, q_1, q_2, \dots, q_k \rangle$,
- q_1, q_2, \dots, q_k die erzeugten k Truth-table-Fragen sind und
- τ eine k -stellige boolesche Funktion ist, die als boolescher Schaltkreis codiert ist, dessen EingabevARIABLEN b_1, b_2, \dots, b_k die k Werte der charakteristischen Funktion von B bei q_i sind, $1 \leq i \leq k$, und so dass $\tau(b_1, b_2, \dots, b_k)$ genau dann wahr ist, wenn $x \in L$. Das heißt, setzt man $b_i = c_B(q_i) = 1$ für $q_i \in B$ und $b_i = c_B(q_i) = 0$ für $q_i \notin B$, $1 \leq i \leq k$, so ist

$$x \in L \iff \tau(b_1, b_2, \dots, b_k) = 1. \quad (5.35)$$

Der anschaulichkeit halber – und dies wird im weiteren Beweis zweckmäßig sein – könnte man sich (5.35) folgendermaßen vorstellen: Die rechte Seite der Äquivalenz stellt ein Gerichtsurteil in einer Verhandlung dar, in der x des Mordes angeklagt ist, L ist das Staatsgefängnis, B ein der Wahrheitsfindung dienliches NP-Orakel und τ die Wahrheitstafel der Geschworenen.⁷ Die Frage, die die Geschworenen zu entscheiden haben, ist, ob x schuldig ist und somit ins Gefängnis L gehört oder nicht. Ihre Antwort auf diese Frage hängt von den (hoffentlich korrekten) Antworten auf k andere Fragen ab, die während der Verhandlung untersucht werden und jeweils von der Form „ $q_i \in B$ “ sind. So könnte „ $q_1 \in B$ “ etwa die Frage sein, ob sich x zur Tatzeit am Tatort aufgehalten hat. Nicht überraschend ist, dass die Vertreter der Anklage und die der Verteidigung von x manche der Fragen „ $q_i \in B$ “ verschieden bzw. widersprüchlich beantworten.

Nun kann man sich eine Folge von Orakelantworten als einen k -dimensionalen Bitvektor $\mathbf{v} = (v_1, v_2, \dots, v_k)$ vorstellen, wobei $v_i = 1$ die Antwort „ja“ ($q_i \in B$) und $v_i = 0$ die Antwort „nein“ ($q_i \notin B$) bedeutet. Indem sie einen gegebenen Antwortvektor \mathbf{v} mit ihrer Wahrheitstafel τ auswerten, versuchen die Geschworenen die Entscheidung zu treffen, ob x ins Gefängnis L gehört oder nicht. Mittels dieser Interpretation kann man (5.35) folgendermaßen lesen: x gehört genau dann ins Gefängnis L , wenn die Geschworenen mit Hilfe ihrer Wahrheitstafel τ zum Urteil kommen, dass x schuldig ist, unter Verwendung der Information, ob die einzelnen q_i in B sind. Die Schwierigkeit besteht für die Geschworenen darin, den Wahrheitsgehalt der einzelnen Antworten der Anklage und der Verteidigung von x zu bestimmen.

Verschiedene Antwortvektoren stellen unterschiedliche Meinungen der Staatsanwaltschaft bzw. der Anwälte von x hinsichtlich der Zugehörigkeit der Fragewörter q_1, q_2, \dots, q_k zu B dar. Leider wissen die Geschworenen nicht, ob eine solche Meinung verlässlich ist oder nicht. Da B jedoch in NP ist, hat gemäß Satz 5.24 jede Behauptung der Form „ $q_i \in B$ “ einen effizient überprüfbaren Zeugen. Das heißt, die

⁷ Dieser Prozess findet in Boston, Massachusetts, unter dem Rechtssystem dieses US-Staates statt.

Geschworenen können jede positive Antwort verifizieren, indem sie einen Zeugen für „ $q_i \in B$ “ (unter Eid) aussagen lassen. Sollte die Aussage eines Zeugen die Geschworenen zu der Überzeugung bringen, dass ein q_i , von welchem sie zuvor annahmen, es gehöre nicht zu B , in Wirklichkeit doch in B ist, und falls außerdem diese aktualisierte Information ein neues Ergebnis der Wahrheitstafel τ über den Urteilspruch liefert (also ob „ $x \in L$ “ oder „ $x \notin L$ “ zu urteilen sei), dann sagen wir, τ hat seine Meinung geändert (auf englisch: τ has changed its mind). Deshalb bezeichnet man die im Folgenden beschriebene Beweistechnik als die „Mind-Change“-Technik.

Interpretiert man Bitvektoren $\mathbf{v} = (v_1, v_2, \dots, v_k)$ als Bitstrings $v_1 v_2 \cdots v_k \in \{0, 1\}^k$, so induziert die lexikographische Ordnung auf $\{0, 1\}^k$ eine Ordnung der Antwortvektoren: Sind $\mathbf{u} = (u_1, u_2, \dots, u_k)$ und $\mathbf{v} = (v_1, v_2, \dots, v_k)$ gegeben, so gilt $\mathbf{u} \leq \mathbf{v}$ genau dann, wenn $u_i \leq v_i$ für jedes i gilt, und $\mathbf{u} < \mathbf{v}$ schreiben wir, falls $\mathbf{u} \leq \mathbf{v}$ und $u_j < v_j$ für ein j gilt. Betrachten wir nur wachsende Ketten von Antwortvektoren, so können darin höchstens k Meinungsänderungen (*mind changes*) vorkommen. Es gibt jedoch kein effizientes Verfahren, um zu bestimmen, wann genau eine solche Meinungsänderung auftritt, nicht einmal nichtdeterministisch. Glücklicherweise kann man jedoch in NP überprüfen, ob mindestens m Meinungsänderungen vorkommen. Anfangs beginnt τ vorsichtig mit dem Nullvektor $\mathbf{v}^0 = (0, 0, \dots, 0)$, nimmt also an, alle Antworten wären negativ. Dann rät τ eine lexikographisch wachsende Folge von m verschiedenen Antwortvektoren, $\mathbf{v}^1, \mathbf{v}^2, \dots, \mathbf{v}^m$. Für jedes j mit $1 \leq j \leq m$ überprüft τ ,

- (a) ob jedes „ja“ in \mathbf{v}^{j-1} auch ein „ja“ in \mathbf{v}^j ist, um die bereits verifizierten korrekten „ja“-Antworten zu behalten, und
- (b) ob $\tau(\mathbf{v}^{j-1}) \neq \tau(\mathbf{v}^j)$ gilt, d.h., ob eine Meinungsänderung aufgetreten ist.

Anschließend rät τ Zeugen für „ $q_i \in B$ “ für jede „ja“-Antwort in \mathbf{v}^m und überprüft jeden solchen Zeugen deterministisch in Polynomialzeit. Unter Verwendung der oben eingeführten Bezeichnungen wird nun die Menge T definiert durch

$$T = \left\{ \langle x, m \rangle \mid \begin{array}{l} (\exists \mathbf{v}^1, \mathbf{v}^2, \dots, \mathbf{v}^m) [(\forall i \leq k) [v_i^m = 1 \implies q_i \in B]] \\ \wedge (\forall j \leq m) [\mathbf{v}^{j-1} < \mathbf{v}^j \wedge \tau(\mathbf{v}^{j-1}) \neq \tau(\mathbf{v}^j)] \end{array} \right\}.$$

Es ist offensichtlich, dass T in NP ist. Sei nun $\mathbf{b} = (b_1, b_2, \dots, b_k)$ der korrekte Antwortvektor gemäß B . Sei weiter $\hat{m}(x) = \max_{\langle x, m \rangle \in T} m$ das größte m mit $\langle x, m \rangle \in T$, und seien $\mathbf{v}^1, \mathbf{v}^2, \dots, \mathbf{v}^{\hat{m}(x)}$ die Vektoren, die dies bezeugen. Natürlich ist $\hat{m}(x) \leq k$ die größte Zahl von Meinungsänderungen. Nach Definition von T gilt

$$\tau(\mathbf{v}^{\hat{m}(x)}) = \tau(\mathbf{b}), \quad (5.36)$$

da andernfalls die Folge $\mathbf{v}^1, \mathbf{v}^2, \dots, \mathbf{v}^{\hat{m}(x)}, \mathbf{b}$ von $\hat{m}(x) + 1$ Antwortvektoren zeigen würde, dass $\langle x, \hat{m}(x) + 1 \rangle \in T$, im Widerspruch zur Wahl von $\hat{m}(x)$.

Da $\tau(\mathbf{v}^j) \equiv 1 + \tau(\mathbf{v}^{j-1}) \pmod{2}$ für jedes j gilt, folgt

$$\tau(\mathbf{v}^{\hat{m}(x)}) \equiv \tau(\mathbf{v}^0) + \hat{m}(x) \pmod{2}. \quad (5.37)$$

(5.35), (5.36) und (5.37) implizieren nun

$$x \in L \iff \tau(\mathbf{v}^0) + \hat{m}(x) \equiv 1 \pmod{2}. \quad (5.38)$$

Nach Konstruktion ist $\hat{m}(x) = \|\{m \mid 1 \leq m \leq k \text{ und } \langle x, m \rangle \in T\}\|$. Da T in NP ist, folgt $T \leq_m^p \text{SAT}$ mittels einer Reduktion h . Für jedes m mit $1 \leq m \leq k$ sei $\varphi_m = h(\langle x, m \rangle)$ die entsprechende boolesche Formel. Es folgt

$$\hat{m}(x) = \|\{m \mid 1 \leq m \leq k \text{ und } \varphi_m \in \text{SAT}\}\|. \quad (5.39)$$

Definiere die DPOTM M mit Orakel $\text{Odd-}k\text{-SAT}$ wie folgt. Bei Eingabe x berechnet M zunächst deterministisch $f(x) = \langle \tau, q_1, q_2, \dots, q_k \rangle$, den Wert von $\tau(\mathbf{v}^0)$ und die k Formeln $\varphi_1, \varphi_2, \dots, \varphi_k$ wie oben definiert. Dann stellt M eine Frage, nämlich danach, ob $\langle \varphi_1, \varphi_2, \dots, \varphi_k \rangle$ zum Orakel $\text{Odd-}k\text{-SAT}$ gehört oder nicht. Sei $a \in \{0, 1\}$ die Antwort des Orakels, wobei $a = 1$, falls $\langle \varphi_1, \varphi_2, \dots, \varphi_k \rangle$ in $\text{Odd-}k\text{-SAT}$ ist, und $a = 0$ anderenfalls. Nach (5.39) ist $\hat{m}(x)$ genau dann ungerade, wenn $a = 1$. Schließlich akzeptiert M die Eingabe x genau dann, wenn $\tau(\mathbf{v}^0) + a \equiv 1 \pmod{2}$. Nach (5.38) ist $L(M^A) = L$, wobei M dem Orakel $A = \text{Odd-}k\text{-SAT}$ lediglich eine Frage stellt. Somit bezeugt M , dass $P_{k-\text{tt}}^{\text{NP}} \subseteq P^{A[1]}$ für eine Menge A gilt, die \leq_m^p -vollständig in $\text{BH}_k(\text{NP})$ ist.

Die umgekehrte Inklusion, $P^{A[1]} \subseteq P_{k-\text{tt}}^{\text{NP}}$, ergibt sich unmittelbar aus der Tatsache, dass $\text{Odd-}k\text{-SAT}$ in $P_{k-\text{tt}}^{\text{NP}}$ ist.

2. Die Inklusion $P_{(2^k-1)-\text{tt}}^{\text{NP}[k]} \subseteq P_{(2^k-1)-\text{tt}}^{\text{NP}}$ wird durch eine direkte Simulation einer gegebenen $P^{\text{NP}[k]}$ -Berechnung ohne Orakelaufrufe gezeigt. Versieht man dabei jeden Berechnungspfad mit einer geeigneten Polynomialzeituhr, so stellt man sicher, dass diese Simulation in Polynomialzeit terminiert. Da nicht mehr als k Fragen in der $P^{\text{NP}[k]}$ -Berechnung gestellt werden, kommen höchstens $2^k - 1$ mögliche Fragen im potenziellen Orakelbaum von $P^{\text{NP}[k]}$ vor. Indem man all diese Fragen simultan (d.h. parallel) in der entsprechenden $P_{(2^k-1)-\text{tt}}^{\text{NP}}$ -Simulation stellt, kann man den korrekten Pfad im potenziellen Orakelbaum ermitteln.

Umgekehrt zeigen wir nun $P_{(2^k-1)-\text{tt}}^{\text{NP}} \subseteq P^{\text{NP}[k]}$. Betrachte dazu die Menge $A = \text{Odd-}(2^k - 1)\text{-SAT}$. Eine einfache Binärsuche mit k Fragen an die NP-Orakelmenge SAT bestimmt, wie viele der $2^k - 1$ Eingabeformeln erfüllbar sind. Somit ist die Menge $\text{Odd-}(2^k - 1)\text{-SAT}$ in $P^{\text{NP}[k]}$. Aus der ersten Aussage dieses Satzes folgt:

$$P_{(2^k-1)-\text{tt}}^{\text{NP}} = P^{A[1]} \subseteq P^{\text{NP}[k]}.$$

3. Die Inklusion $\text{BH}_k(\text{NP}) \cup \text{coBH}_k(\text{NP}) \subseteq P_{k-\text{tt}}^{\text{NP}}$ folgt aus den Definitionen. Um die Inklusion $P_{k-\text{tt}}^{\text{NP}} \subseteq \text{BH}_{k+1}(\text{NP}) \cap \text{coBH}_{k+1}(\text{NP})$ zu beweisen, genügt es, $P_{k-\text{tt}}^{\text{NP}} \subseteq \text{BH}_{k+1}(\text{NP})$ zu zeigen, da $P_{k-\text{tt}}^{\text{NP}}$ unter Komplementbildung abgeschlossen ist. Da andererseits $\text{Odd-}(k+1)\text{-SAT}$ in $\text{BH}_{k+1}(\text{NP})$ und $\text{BH}_{k+1}(\text{NP}) \leq_m^p$ -abgeschlossen ist, folgt gemäß Lemma 3.36 die Inklusion $P_{k-\text{tt}}^{\text{NP}} \subseteq \text{BH}_{k+1}(\text{NP})$, falls wir zeigen können, dass $\text{Odd-}(k+1)\text{-SAT} \leq_m^p$ -hart für $P_{k-\text{tt}}^{\text{NP}}$ ist.

Sei also L eine beliebige Menge in $P_{k-\text{tt}}^{\text{NP}}$. Nach der ersten Aussage dieses Satzes ist L in $P^{A[1]}$ für die Orakelmenge $A = \text{Odd-}k\text{-SAT}$. Folglich gibt es eine DPOTM M , die L mit einem Orakelaufruft an $\text{Odd-}k\text{-SAT}$ entscheidet. Für eine gegebene Eingabe x sei $q_x = \langle \varphi_1^x, \varphi_2^x, \dots, \varphi_k^x \rangle$ diese Orakelfrage von $M^A(x)$. Sei $\psi \in \text{SAT}$ eine feste erfüllbare

Formel, und sei $\bar{\psi} \in \overline{\text{SAT}}$ eine feste unerfüllbare Formel. Definiere die boolesche Formel φ_0^x wie folgt:

$$\varphi_0^x = \begin{cases} \psi & \text{falls } (M^A \text{ akzeptiert } x \iff \text{die Antwort auf } q_x \text{ ist „nein“}) \\ \bar{\psi} & \text{sonst.} \end{cases}$$

Sei f die Funktion, die ein gegebenes x auf das $(k+1)$ -Tupel $\langle \varphi_0^x, \varphi_1^x, \dots, \varphi_k^x \rangle$ boolescher Formeln abbildet. Offenbar kann f in Polynomialzeit berechnet werden, und für jedes $x \in \Sigma^*$ gilt:

$$\begin{aligned} x \in L &\iff M^A \text{ akzeptiert } x \\ &\iff (\varphi_0^x \in \text{SAT} \wedge q_x \text{ hat die Antwort „nein“}) \vee \\ &\quad (\varphi_0^x \notin \text{SAT} \wedge q_x \text{ hat die Antwort „ja“}) \\ &\iff (\varphi_0^x \in \text{SAT} \wedge q_x \notin \text{Odd-}k\text{-SAT}) \vee (\varphi_0^x \notin \text{SAT} \wedge q_x \in \text{Odd-}k\text{-SAT}) \\ &\iff f(x) = \langle \varphi_0^x, \varphi_1^x, \dots, \varphi_k^x \rangle \in \text{Odd-}(k+1)\text{-SAT}. \end{aligned}$$

Somit beschreibt f eine \leq_m^p -Reduktion von L auf Odd- $(k+1)$ -SAT. Da L eine beliebige Menge in $P_{k-\text{tt}}^{\text{NP}}$ ist, ist Odd- $(k+1)$ -SAT \leq_m^p -hart für $P_{k-\text{tt}}^{\text{NP}}$.

4. Diese Aussage folgt unmittelbar aus der zweiten und der dritten Aussage dieses Satzes. \square

Eine unmittelbare Konsequenz aus Satz 5.53 ist, dass sowohl $P^{\text{NP}[\mathcal{O}(1)]}$ als auch $P_{\text{bit}}^{\text{NP}}$ genau mit der booleschen Hierarchie über NP – und nach Satz 5.6 ebenso mit der booleschen Hülle über NP – übereinstimmt.

Korollar 5.54. $BH(\text{NP}) = P^{\text{NP}[\mathcal{O}(1)]} = P_{\text{bit}}^{\text{NP}} = BC(\text{NP})$.

Beweis. Die ersten beiden Gleichheiten folgen aus Satz 5.53. Die letzte Gleichheit folgt aus Korollar 5.7. \square

Der Beweis der zweiten Aussage von Satz 5.53, also der Gleichheit von $P^{\text{NP}[k]}$ und $P_{(2^k-1)-\text{tt}}^{\text{NP}}$, funktioniert auch dann, wenn man die konstante Zahl k von Fragen durch $c \cdot \log n$ Fragen ersetzt, für eine Konstante c und für Eingaben der Länge n . Somit ist die boolesche Hierarchie über NP in der Θ_2^p -Stufe der Polynomialzeit-Hierarchie enthalten.

Korollar 5.55. $BH(\text{NP}) \subseteq P^{\text{NP}[\log]} = P_{\text{tt}}^{\text{NP}} = \Theta_2^p$.

Die Gleichheit $P^{\text{NP}[\log]} = P_{\text{tt}}^{\text{NP}}$ im obigen Korollar wurde unabhängig von L. Hemaspaandra [Hem87, Hem89], von Buss und Hay [BH88, BH91] sowie von Köbler, Schöning und Wagner [KSW87, Wag90] bewiesen.

5.5 Die boolesche Hierarchie kollabiert die Polynomialzeit-Hierarchie

In diesem Abschnitt werden die Beziehungen zwischen der Polynomialzeit-Hierarchie und der booleschen Hierarchie über NP genauer untersucht, und dabei wird sich ein interessanter, enger Zusammenhang zwischen diesen beiden Hierarchien zeigen. Diesen ganz erstaunlichen Zusammenhang soll zunächst die folgende kleine Geschichte illustrieren, in der Paula und Ella die Hauptrolle spielen, dieselben beiden kleinen Mädchen, die in Abschnitt 3.3 die „Upward-Separation“-Technik von Book erklärten. Auch sollte man sich an den Boolean Hierarchy Tower (kurz BHT) erinnern, der in Abschnitt 5.1 geschildert wurde, um die „Upward-Collapse“-Eigenschaft der booleschen Hierarchie zu erläutern. Ist die Erinnerung daran wieder da? Falls ja,⁸ dann hört zu:

Story 5.56 *Paula und Ella wohnen in der zweiten Etage des Polynomial Hierarchy Tower (kurz PHT), einem der höchsten Wolkenkratzer ihrer Stadt, noch höher und noch beeindruckender als der BHT. Tatsächlich sagt Korollar 5.55 oben, dass der gesamte BHT in die zweite Etage des PHT passt. Es ist vielleicht nicht ganz einfach, sich vorzustellen, dass ein gewaltiges Gebäude wie der BHT mit potenziell unendlich vielen Stockwerken gänzlich in den ersten beiden Etagen eines anderen Gebäudes wie dem PHT verschwindet. Stellen wir uns den BHT nun also als einen Spielzeugturm vor, als das Modell eines Gebäudes, das in Paula und Ellas Kinderzimmer steht. Damit die womöglich unendlich vielen Stockwerke des BHT bequem in ihr Zimmer passen, stellen wir uns weiter vor, dass jede Etage des BHT-Spielzeugmodells lediglich halb so groß wie die darunter befindliche Etage ist.*

Die beiden Schätzchen sind heute in aggressiver, destruktiver Stimmung: Absichtlich beschädigen sie das BHT-Spielzeugmodell mit (echten) Hämmern, so schlimm, dass eine seiner unteren Etagen, nämlich die fünfte, schließlich völlig kaputt geht, kollabiert und auf die vierte Etage herunterkracht. Da sie Satz 5.10 kennen, erwarten Paula und Ella nun natürlich, dass der gesamte BHT auf seine vierte Etage kollabiert, und zu ihrem ausgesprochenen Vergnügen passiert genau das. Kreischend, klatschend und kichernd hüpfen sie auf und ab.

Unglücklicherweise jedoch kennen sie Satz 5.57 noch nicht. Deshalb rechnen sie nicht mit dem, was im nächsten Augenblick geschehen wird: Erst hören sie ein hohles Rumpeln von weit oben, dann sehen sie kleine Risse in den Wänden und an der Decke ihres Zimmers, das ganze Gebäude zittert und bebt und wankt wie bei einem Erdbeben, und ganz plötzlich bricht es mit fürchterlichem Getöse in sich zusammen. Hustend und weinend stürzen die erschrockenen Kinder die Treppen hinunter durch all den Staub und die fallenden Betonbrocken und stürmen hinaus ins Freie. Dort sehen sie, was sie angerichtet haben, als sie gedankenlos das BHT-Spielzeugmodell zerstörten: Der ganze PHT ist auf seine dritte Etage kollabiert, und Teile der zweiten Etage sind ebenfalls zerstört.

⁸ Falls nein, dann sollte man zurückgehen und die Stories 3.23 und 5.9 in den Abschnitten 3.3 und 5.1 noch einmal lesen.

Glücklicherweise wurde niemand verletzt, da sich im dritten und in den höheren Stockwerken des PHT ausschließlich Büroräume befinden, die zu dieser Tageszeit verlassen sind. Paula und Ella sehen nun ihre zu Tode erschrockenen und völlig verwirrten Eltern aus dem PHT – oder was davon noch übrig ist – eilen. Nachdem die Kinder der Polizei und der Versicherungsgesellschaft ihre Tat gestanden und dafür eine angemessene Strafe erhalten haben (u.a. Entzug der Hämmer), hören sie den nachdrücklichen und äußerst deutlichen Erklärungen ihrer Eltern über Satz 5.57 zu.

Satz 5.57 (Kadin). Wenn es ein $k \geq 1$ gibt, so dass $\text{BH}_k(\text{NP}) = \text{coBH}_k(\text{NP})$ gilt, dann kollabiert die Polynomialzeit-Hierarchie auf ihre dritte Stufe: $\text{PH} = \Sigma_3^P \cap \Pi_3^P$.

Nach Satz 5.10 folgt aus der Annahme $\text{BH}_k(\text{NP}) = \text{coBH}_k(\text{NP})$ von Satz 5.57, dass $\text{BH}_k(\text{NP}) = \text{BH}_{k+1}(\text{NP})$ gilt und somit die gesamte boolesche Hierarchie auf die k -te Stufe kollabiert. Satz 5.57 gibt das ursprüngliche Resultat Kadins an [Kad88]. Die Kollapskonsequenz dieses Satzes ist später verschärft worden. Insbesondere zeigte Wagner [Wag87b, Wag89], dass aus der Annahme $\text{BH}_k(\text{NP}) = \text{coBH}_k(\text{NP})$ der tiefere Kollaps $\text{PH} = \Delta_3^P$ und sogar $\text{PH} = \text{BH}(\Sigma_2^P)$ folgt. Unter der selben Annahme verschärften Chang und Kadin [Cha91, CK96] die Kollapskonsequenz für PH bis hinunter zu $\text{BH}_k(\Sigma_2^P)$, welches in $\text{BH}(\Sigma_2^P) \subseteq \Theta_3^P$ enthalten ist. Weitere Verbesserungen wurden von Beigel, Chang und Ogihara [BCO93] und von E. und L. Hemaspaandra und Hempel [HHH98b] erreicht, siehe auch [Hem98]. Abschnitt 5.9 liefert weitere Details und Literaturhinweise.

Der Beweis von Satz 5.57 wendet eine Methode an, die als die „Easy-Hard“-Technik bezeichnet wird. Wesentlich in dieser Technik sind der Begriff der dünnen Menge sowie ein Resultat von Yap [Yap83], das sagt, dass wenn es eine dünne \leq_T^{NP} -harte Menge für coNP gibt, dann tritt die Kollapskonsequenz aus Satz 5.57 auf. Dünne Mengen haben nur einen geringen Informationsgehalt: Bis zu jeder gegebenen Länge enthalten sie nicht mehr als polynomiell viele Wörter. Das Ergebnis von Yap wird hier als Lemma 5.59 ohne Beweis angegeben. Für Verbesserungen dieses Resultates und für verwandte Resultate sei auf Abschnitt 5.9 verwiesen.

Definition 5.58 (Dünne Menge). Definiere für jede Menge S und jedes $n \in \mathbb{N}$ die Menge der Wörter bis zur Länge n als $S^{\leq n} = \{x \mid x \in S \text{ und } |x| \leq n\}$. Eine Menge S heißt dünn, falls gilt:

$$(\exists p \in \text{IPol}) (\forall n \in \mathbb{N}) [\|S^{\leq n}\| \leq p(n)].$$

Lemma 5.59 (Yap). Wenn es eine dünne Menge S mit $\text{coNP} \subseteq \text{NP}^S$ gibt, dann ist

$$\text{PH} = \Sigma_3^P \cap \Pi_3^P.$$

Beweis von Satz 5.57. Für $k = 1$ impliziert Satz 5.33 sofort einen noch stärkeren Kollaps: $\text{PH} = \text{NP} \cap \text{coNP}$. Wir geben hier einen Beweis für den Fall $k = 2$; der allgemeine Fall für $k \geq 2$ kann analog bewiesen werden.

Sei $\text{BH}_2(\text{NP}) = \text{coBH}_2(\text{NP})$ angenommen, d.h., $\text{DP} = \text{coDP}$. Da die Menge

$$\overline{\text{SAT}} = \{\varphi \mid \varphi \text{ ist eine unerfüllbare boolesche Formel}\}$$

coNP-vollständig ist, genügt es zu zeigen, dass $\overline{\text{SAT}}$ in NP^S für eine dünne Menge S liegt, woraus die Hypothese $\text{coNP} \subseteq \text{NP}^S$ von Lemma 5.59 folgt. Nach Lemma 5.59 ergibt sich dann der gewünschte Kollaps $\text{PH} = \Sigma_3^P \cap \Pi_3^P$.

Definiere die Menge SAT-UNSAT durch

$$\text{SAT-UNSAT} = \left\{ \langle \varphi, \psi \rangle \mid \begin{array}{l} \varphi \text{ und } \psi \text{ sind boolesche Formeln in KNF,} \\ \text{so dass } \varphi \text{ erfüllbar und } \psi \text{ nicht erfüllbar ist} \end{array} \right\}.$$

SAT-UNSAT ist DP-vollständig; siehe Übung 5.2. Aus unserer Annahme $\text{DP} = \text{coDP}$ folgt, dass SAT-UNSAT ebenso \leq_m^P -vollständig für coDP ist. Insbesondere folgt $\text{SAT-UNSAT} \leq_m^P \overline{\text{SAT-UNSAT}}$. Sei $f \in \text{FP}$ diese \leq_m^P -Reduktion von SAT-UNSAT auf $\overline{\text{SAT-UNSAT}}$. Boolesche Formeln werden geeignet durch Wörter über dem Alphabet $\{0, 1\}$ codiert. Für jedes Paar $\langle \varphi, \psi \rangle$ solcher Wörter ergibt sich dann

$$\langle \varphi, \psi \rangle \in \text{SAT-UNSAT} \iff f(\langle \varphi, \psi \rangle) \in \overline{\text{SAT-UNSAT}}. \quad (5.40)$$

Die Reduktion f bildet Paare von Wörtern, die boolesche Formeln codieren, auf Paare von Wörtern ab, die ebenfalls boolesche Formeln codieren. Seien g und h die Funktionen in FP, die für jedes solche Paar $\langle \varphi, \psi \rangle$ erfüllen, dass

$$f(\langle \varphi, \psi \rangle) = \langle g(\langle \varphi, \psi \rangle), h(\langle \varphi, \psi \rangle) \rangle.$$

Die Äquivalenz (5.40) kann dann so geschrieben werden:

$$\varphi \in \text{SAT} \wedge \psi \in \overline{\text{SAT}} \iff g(\langle \varphi, \psi \rangle) \in \overline{\text{SAT}} \vee h(\langle \varphi, \psi \rangle) \in \text{SAT}. \quad (5.41)$$

Die folgenden Bezeichnungen erklären, weshalb diese Beweistechnik die „Easy-Hard“-Technik genannt wird:

- Ein Wort $\psi \in \{0, 1\}^*$ heißt *leicht*, falls

$$(\exists \varphi \in \{0, 1\}^*) [|\varphi| = |\psi| \wedge h(\langle \varphi, \psi \rangle) \in \text{SAT}]. \quad (5.42)$$

- Ein Wort $\psi \in \{0, 1\}^*$ heißt *hart*, falls

$$\psi \text{ nicht leicht ist und } \psi \in \overline{\text{SAT}}. \quad (5.43)$$

Behauptung 5.60. 1. Die Menge $E = \{\psi \in \{0, 1\}^* \mid \psi \text{ ist leicht}\}$ ist in NP.

2. Ist ψ in E , so ist ψ in $\overline{\text{SAT}}$.

3. Ist ψ hart, so gilt für jedes φ mit $|\varphi| = |\psi|$:

$$\varphi \in \text{SAT} \iff g(\langle \varphi, \psi \rangle) \in \overline{\text{SAT}}. \quad (5.44)$$

Beweis von Behauptung 5.60. 1. Der Beweis der ersten Aussage folgt leicht aus Satz 5.24; die Details werden dem Leser als Übung 5.28 überlassen.

2. Sei $\psi \in E$. Da ψ leicht ist, gibt es ein Wort φ , $|\varphi| = |\psi|$, so dass $h(\langle \varphi, \psi \rangle)$ in SAT ist. Folglich ist die rechte Seite der Äquivalenz (5.41) erfüllt, gleichgültig, ob $g(\langle \varphi, \psi \rangle)$ in $\overline{\text{SAT}}$ ist oder nicht. Somit ist die linke Seite von (5.41) ebenfalls erfüllt, woraus insbesondere $\psi \in \overline{\text{SAT}}$ folgt.

```

 $M^S(\varphi) \{$ 
    // Eingabewort  $\varphi \in \{0,1\}^*$ ,  $|\varphi| = n$ , codiert eine boolesche Formel
     $\psi := \varepsilon;$ 
    for ( $i = n, n-1, \dots, 1$ ) {
        if ( $\psi 0 2^{i-1} \in S$ )  $\psi := \psi 0$ ;
        else  $\psi := \psi 1$ ;
    }
    return  $\psi$ 
}

```

Abb. 5.12. „Easy-Hard“-Technik: Präfixsuche nach dem kleinsten harten Wort der Länge n

3. Wenn ψ hart ist, dann kann es nicht leicht sein. Nach (5.42) gibt es also kein φ der Länge $|\varphi|$ mit $h(\langle \varphi, \psi \rangle) \in \text{SAT}$. Und weil ψ hart ist, folgt $\psi \in \overline{\text{SAT}}$. Somit reduziert sich die Äquivalenz (5.41) auf die Äquivalenz (5.44). \square Behauptung 5.60

Um den Beweis von Satz 5.57 fortzusetzen, sei 2 ein neues Symbol. Definiere die Menge S über dem Alphabet $\{0, 1, 2\}$ wie folgt. Für jedes $n \in \mathbb{N}$ sei die Menge S_n bei Länge n definiert durch

$$S_n = \left\{ \psi 2^{n-|\psi|} \middle| \begin{array}{l} (\exists \alpha \in \{0, 1\}^*) [|\alpha| = n \text{ und } \alpha \text{ ist hart}] \text{ und} \\ \psi \in \{0, 1\}^* \text{ ist ein Präfix des lexikographisch} \\ \text{kleinsten harten Wortes der Länge } n \end{array} \right\}.$$

Sei $S = \bigcup_{n \in \mathbb{N}} S_n$. Für jedes $n \in \mathbb{N}$ enthält S_n entweder gar kein hartes Wort, und in diesem Fall ist $\|S_n\| = 0$, oder aber es gilt $\|S_n\| = n + 1$. Folglich ist $\|S^{\leq n}\| \leq (n + 1)^2$ für jedes $n \in \mathbb{N}$. Also ist S eine dünne Menge.

Um zu beweisen, dass $\overline{\text{SAT}}$ in NP^S ist, definiere eine NPOTM M^S , die bei einer Eingabe φ der Länge n folgendermaßen arbeitet:

Schritt 1: M^S bestimmt, ob S_n leer ist oder nicht mit einer Frage an das Orakel: „ $2^n \in S$?“

Schritt 2: Entsprechend einer positiven bzw. einer negativen Orakelantwort sind zwei Fälle zu unterscheiden.

Fall 1: Die Orakelantwort ist „ja“, d.h., $2^n \in S$ und $S_n \neq \emptyset$. In diesem Fall arbeitet M^S bei Eingabe φ wie folgt. Zunächst bestimmt M^S durch eine Präfixsuche mit n Fragen an das Orakel S das lexikographisch kleinste harte Wort $\psi \in S_n$ der Länge n . Der Algorithmus für diese Präfixsuche ist in Abbildung 5.12 angegeben.

Da ψ hart ist, impliziert die Äquivalenz (5.44) aus Behauptung 5.60.3, dass für jedes φ mit $|\varphi| = |\psi|$ gilt:

$$\varphi \in \overline{\text{SAT}} \iff g(\langle \varphi, \psi \rangle) \in \text{SAT}.$$

Um $\overline{\text{SAT}}$ zu akzeptieren, berechnet $M^S(\varphi)$ nun die Formel $g(\langle \varphi, \psi \rangle)$ deterministisch in Polynomialzeit, simuliert die Standard-NP-Maschine für SAT

bei Eingabe $g(\langle \varphi, \psi \rangle)$ und akzeptiert φ genau dann, wenn $g(\langle \varphi, \psi \rangle) \in \text{SAT}$. Es folgt $L(M^S) = \overline{\text{SAT}}$.

Fall 2: Die Orakelantwort ist „nein“, d.h., $2^n \notin S$ und $S_n = \emptyset$. In diesem Fall gibt es kein hartes Wort der Länge n in S . Insbesondere ist φ nicht hart. Gemäß der Definition von „hart“ (und durch Negieren von (5.43)) ergibt sich, dass φ leicht ist oder $\varphi \notin \overline{\text{SAT}}$. Äquivalent dazu ist, dass $\varphi \in \overline{\text{SAT}}$ impliziert, dass φ leicht ist. Aus Behauptung 5.60.2 folgt, dass φ genau dann in $\overline{\text{SAT}}$ ist, wenn φ leicht ist. Nach Behauptung 5.60.1, gibt es eine NP-Maschine N , die genau die leichten Eingabewörter akzeptiert. Also simuliert $M^S(\varphi)$ im vorliegenden Fall einfach die Berechnung von N bei Eingabe φ . Es folgt $L(M^S) = L(N) = \overline{\text{SAT}}$.

Da $L(M^S) = \overline{\text{SAT}}$ in beiden Fällen gilt, ist $\overline{\text{SAT}}$ in NP^S für die dünne Menge S , und damit ist die Voraussetzung $\text{coNP} \subseteq \text{NP}^S$ von Lemma 5.59 erfüllt. Dieses Lemma wiederum impliziert den Kollaps $\text{PH} = \Sigma_3^P \cap \Pi_3^P$. \square Satz 5.57

5.6 Alternierende Turingmaschinen

Die Klassen der Polynomialzeit-Hierarchie, die nach Satz 5.31 durch alternierende existentielle und universelle Quantoren charakterisiert werden können, verallgemeinern die Klassen P und NP. Diese Charakterisierung legt demnach nahe, dass *Alternierung* eine natürliche Verallgemeinerung von sowohl Determinismus als auch Nichtdeterminismus ist, welche als die grundlegenden Berechnungsparadigma in Kapitel 3 eingeführt wurden. In diesem Abschnitt wird diese Idee dadurch formalisiert, dass noch ein weiterer Typ einer Turingmaschine eingeführt wird, die *alternierende Turingmaschine* (kurz ATM).

Syntaktisch ist eine alternierende Turingmaschine nichts anderes als eine nicht-deterministische Turingmaschine (kurz NTM; formal wurde dieses Modell in den Definitionen 2.15 und 2.16 in Abschnitt 2.2 beschrieben), deren Zustände, sofern sie keine Endzustände sind, entweder als „existenziell“ oder als „universell“ markiert sind. Semantisch unterscheidet sich der Akzeptierungsmodus einer ATM dadurch von dem einer NTM, dass erstere ihre Eingaben mittels so genannter „akzeptierender alternierender Teilebäume“ akzeptiert, während bei letzterer dafür bereits ein akzeptierender Pfad genügt.

Nun definieren wir die Syntax und Semantik von ATMs formal.

Definition 5.61 (Alternierende Turingmaschine).

1. **Syntax:** Eine NTM M heißt alternierende Turingmaschine (kurz ATM), falls die Zustandsmenge S von M in vier disjunkte Teilmengen zerlegt ist, nämlich $S = E \cup U \cup A \cup R$, wobei
 - die Elemente von E die existenziellen Zustände von M ,
 - die Elemente von U die universellen Zustände von M ,
 - die Elemente von A die akzeptierenden Zustände von M und
 - die Elemente von R die ablehnenden Zustände von M sind.

2. Semantik: Sei M eine ATM, sei x ein Eingabewort und sei $M(x)$ der Berechnungsbaum von M bei Eingabe x . Für jede Konfiguration C in $M(x)$, deren Zustand $s \in S$ ist, wird C bezeichnet als

- eine existenzielle Konfiguration von $M(x)$ (markiert mit \vee), falls $s \in E$,
- eine universelle Konfiguration von $M(x)$ (markiert mit \wedge), falls $s \in U$,
- eine akzeptierende Konfiguration von $M(x)$ (markiert mit 1), falls $s \in A$ und
- eine ablehnende Konfiguration von $M(x)$ (markiert mit 0), falls $s \in R$.

Akzeptierende und ablehnende Konfigurationen sind die einzigen Endkonfigurationen – und als solche die Blätter – von $M(x)$. Existenzielle und universelle Konfigurationen sind die inneren Knoten von $M(x)$. Da wir ohne Beschränkung der Allgemeinheit annehmen dürfen, dass der nichtdeterministische Verzweigungsgrad von M bei jeder Eingabe genau zwei ist, hat jede existenzielle oder universelle Konfiguration genau zwei unmittelbare Folgekonfigurationen.

Sei \mathfrak{K}_M die Menge aller Konfigurationen von M . Definiere die Bewertungsfunktion $\text{eval} : \mathfrak{K}_M \rightarrow \{0, 1\}$ wie folgt:

$$\text{eval}(C) = \begin{cases} 0 & \text{falls } C \text{ ablehnend} \\ 1 & \text{falls } C \text{ akzeptierend} \\ \text{eval}(C_1) \vee \text{eval}(C_2) & \text{falls } C \text{ existenziell} \\ \text{eval}(C_1) \wedge \text{eval}(C_2) & \text{falls } C \text{ universell} \end{cases}$$

ist, wobei C_1 und C_2 die beiden unmittelbaren Folgekonfigurationen von C sind. Sei $\text{START}_M(x)$ die eindeutige Startkonfiguration von M bei Eingabe x . Die ATM M akzeptiert die Eingabe x genau dann, wenn $\text{eval}(\text{START}_M(x)) = 1$. Die von M akzeptierte Sprache ist definiert als

$$L(M) = \{x \in \Sigma^* \mid \text{eval}(\text{START}_M(x)) = 1\}.$$

Anders gesagt, bewertet die Funktion eval den Baum $M(x)$ von den Blättern zur Wurzel hin, indem sie ihn sozusagen als einen booleschen Schaltkreis auffasst, dessen Eingabegatter die Blätter des Baumes sind und dessen Ausgabegatter die Wurzel des Baumes ist.

Um „alternierende“ Komplexitätsmaße und -klassen für ATMs zu definieren (also etwa die Begriffe „alternierende Zeit“ und „alternierender Raum“), wird nun das Konzept des „akzeptierenden alternierenden Teilbaums“ eingeführt.

Definition 5.62 (Akzeptierender Alternierender Teilbaum einer ATM). Sei M eine ATM, sei x ein Eingabewort und sei $M(x)$ der Berechnungsbaum von M bei Eingabe x . Für jeden Teilbaum T von $M(x)$ sei eval_T die auf T eingeschränkte Bewertungsfunktion eval aus Definition 5.61.

T ist ein akzeptierender alternierender Teilbaum von $M(x)$, falls

1. T die Wurzel $\text{START}_M(x)$ von $M(x)$ enthält und
2. $\text{eval}_T(\text{START}_M(x)) = 1$ gilt.

Abbildung 5.13 gibt ein Beispiel eines akzeptierenden alternierenden Teilbaums im Berechnungsbaum einer ATM M bei einer Eingabe x . Die inneren Knoten des

Baums $M(x)$ sind mit \vee oder \wedge markiert, um existenzielle oder universelle Konfigurationen darzustellen. Die Blätter von $M(x)$ sind mit 1 oder 0 markiert, um akzeptierende oder ablehnende Konfigurationen darzustellen. Die schattierten Knoten von $M(x)$ repräsentieren einen akzeptierenden alternierenden Teilbaum, da die Wurzel dieses Teilbaums gemäß der Funktion `eval` mit 1 bewertet wird.

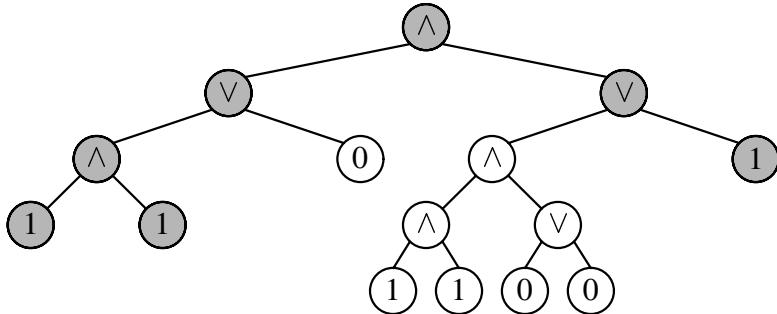


Abb. 5.13. Ein akzeptierender alternierender Teilbaum für eine ATM

Definition 5.63 (Alternierende Zeit und Alternierender Raum). Seien M eine ATM und x ein Eingabewort, und seien t und s Funktionen in \mathbb{R} , die von \mathbb{N} in \mathbb{N} abbilden.

1. M akzeptiert x in der Zeit k , falls $M(x)$ einen akzeptierenden alternierenden Teilbaum der Tiefe höchstens k hat.
2. M akzeptiert eine Sprache L in der Zeit t , falls $L(M) = L$ ist und M jedes $x \in L$ in der Zeit $t(|x|)$ akzeptiert.
3. M akzeptiert x im Raum k , falls $M(x)$ einen akzeptierenden alternierenden Teilbaum hat, dessen sämtliche Knoten Konfigurationen der Größe höchstens k repräsentieren.
4. M akzeptiert eine Sprache L im Raum s , falls $L(M) = L$ ist und M jedes $x \in L$ im Raum $s(|x|)$ akzeptiert.

Definiere die folgenden alternierenden Komplexitätsklassen mit Ressourcenfunktion t bzw. s :

$$\begin{aligned} \text{ATIME}(t) &= \{A \mid \text{es gibt eine ATM } M, \text{ die } A \text{ in der Zeit } t \text{ akzeptiert}\}; \\ \text{ASPACE}(s) &= \{A \mid \text{es gibt eine ATM } M, \text{ die } A \text{ im Raum } s \text{ akzeptiert}\}. \end{aligned}$$

Alternierende Turingmaschinen sind ein „paralleles“ Berechnungsmodell. Parallelität bedeutet hier, dass eine gegebene Aufgabe in kleinere Teilaufgaben aufgeteilt wird, die *parallel – also gleichzeitig – bearbeitet* werden. Diesen Prozess setzt man

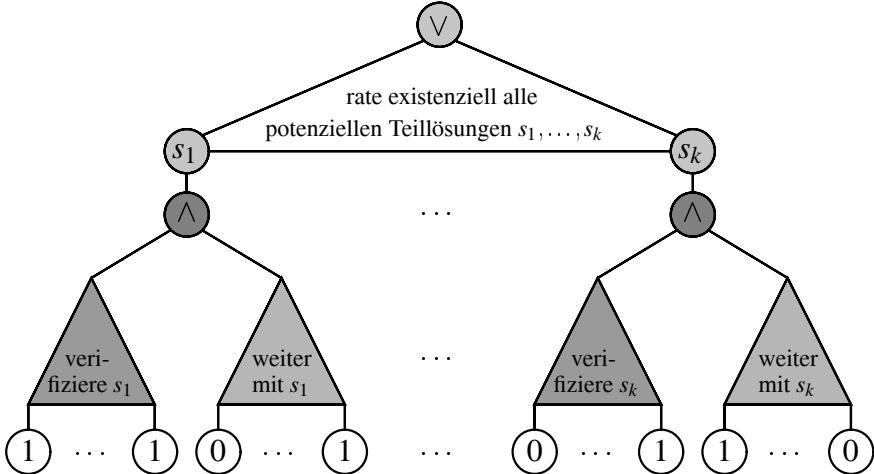


Abb. 5.14. ATMs als ein paralleles Berechnungsmodell

rekursiv fort, indem man die Teilaufgaben weiter in Teilteilaufgaben zerlegt, welche wieder parallel bearbeitet werden, und so weiter, bis man schließlich Aufgaben erhält, die so klein sind, dass sie unmittelbar gelöst werden können. Das Resultat ist eine sehr große Zahl von Lösungen sehr kleiner Aufgaben auf der Blattebene des Berechnungsbaumes, wobei die Blätter entweder akzeptierende oder ablehnende Konfigurationen darstellen. Diese kleinen Teillösungen müssen nun miteinander kombiniert werden, um eine Lösung des Ausgangsproblems zu erhalten, um also die Entscheidung zu fällen, ob die Eingabe an der Wurzel des Baumes, die die Startkonfiguration darstellt, zu akzeptieren oder abzulehnen ist. Parallelität erfordert daher, die an den Blättern erhaltene Information über die Teillösungen rückwärts durch den Baum bis zu seiner Wurzel zu übertragen. Im Fall einer ATM wird dies durch die Bewertungsfunktion eval aus Definition 5.61 und den Akzeptierungsmodus mittels akzeptierender alternierender Teilbäume aus Definition 5.62 erreicht. Die Rückübertragung der Information von den Blättern zur Wurzel des Berechnungsbaums erlaubt die Bewertung der Startkonfiguration, wodurch die kleinen Teillösungen zur Lösung der ursprünglichen Aufgabe wie gewünscht kombiniert werden können.

Abbildung 5.14 zeigt die Berechnung einer ATM als ein paralleles Berechnungsmodell. Zuerst werden in einer existenziellen Phase alle potenziellen Teillösungen s_1, s_2, \dots, s_k geraten. Dann verzweigt die ATM universell: Auf ihrem linken Zweig überprüft sie die Korrektheit von s_i , der hier geratenen Teillösung; auf ihrem rechten Zweig rechnet sie weiter mit s_i , wobei sie annimmt, dass sie richtig geraten hatte. Nur dann, wenn beide dieser Teilbäume akzeptieren, wird die entsprechende universelle Konfiguration den Wert 1 weiter nach oben propagieren. Im rechten Teilbaum unter s_i ist die ATM wieder in einer existenziellen Phase, rät Teillösungen $s_{i,j}$, die s_i erweitern, wobei $1 \leq j \leq \ell$. Dann wird wieder universell verzweigt, wobei parallel

die $s_{i,j}$ verifiziert werden und mit den $s_{i,j}$ weitergerechnet wird. Setzt man diesen Prozess rekursiv fort, so erreicht man schließlich die Blattebene, auf der nur noch entweder zu akzeptieren oder abzulehnen ist.

Intuitiv ist Alternierung ein sehr mächtiges Berechnungsparadigma, viel mächtiger als bloßer Nichtdeterminismus. Der Zusammenhang zwischen Alternierung und Determinismus hinsichtlich der Komplexitätsmaße Raum und Zeit wird später in diesem Abschnitt untersucht werden.

Im folgenden Beispiel wird eine interessante Beobachtung gemacht: Die einer ATM inhärente Parallelität erlaubt uns, von *alternierender logarithmischer Zeit* zu sprechen. Dies ist anders als bei einer DTM oder einer NTM, wo es nicht sinnvoll wäre, logarithmische Zeit zu betrachten, einfach weil eine solche Maschine mit nur logarithmisch vielen Rechenschritten nicht einmal die ganze Eingabe lesen könnte. Im Gegensatz dazu können sämtliche Bits einer Eingabe x der Länge n parallel in $\log n$ Schritten gelesen werden. Zu diesem Zweck wird das ATM-Modell aus Definition 5.61 mit einem zusätzlichen Leseband ausgestattet, dem so genannten *Indexband* oder *Adressregister*, mit dem die ATM direkt auf das Eingabeband zugreifen kann. Steht eine Zahl i in Binärdarstellung auf dem Indexband, so liest die ATM das i -te Bit der Eingabe innerhalb eines Schrittes.

Beispiel 5.64 (Alternierende Logarithmische Zeit). Definiere die Komplexitätsklasse $\text{ALOGTIME} = \text{ATIME}(\log)$. Die Menge $L = \{0^n 1^n \mid n \geq 1\}$ ist in ALOGTIME .

Eine ATM M , die L in der Zeit $\mathcal{O}(\log n)$ akzeptiert, arbeitet folgendermaßen bei Eingabe $x = x_1 x_2 \cdots x_n$, wobei jedes x_i in $\{0, 1\}$ ist:

Schritt 1: Bestimme zunächst die Länge n der Eingabe wie folgt:

1. Schreibe sukzessive die Wörter 1, 10, 100, ..., die in Binärdarstellung die Zahlen $2^0, 2^1, 2^2, \dots$ repräsentieren, auf das Indexband und lies das entsprechende Eingabebit, bis das erste Leerzeichen gelesen wird.
2. Sind k Eingabebits gelesen und das 2^{k+1} -te Feld des Eingabebands enthält ein Leerzeichen, so wissen wir, dass $2^k \leq n < 2^{k+1}$ gilt. Eine Binärsuche auf dem Intervall $[2^k, 2^{k+1})$ bestimmt nun den Wert von n genau.

Schritt 2: Unterscheide zwei Fälle:

1. Ist n ungerade, so lehnt M die Eingabe x ab.
2. Ist n gerade, so verzweigt M universell und überprüft, ob für jedes $i \leq \frac{n}{2}$ gilt, dass $x_i = 0$ und $x_{n-i+1} = 1$, indem sie die Binärdarstellungen von i bzw. von $n - i + 1$ auf das Indexband schreibt.

Da nach Definition $\text{NTIME}(t)$ in $\text{ATIME}(t)$ enthalten ist, verbessert das folgende Ergebnis Korollar 3.28, welches sagt, dass $\text{NTIME}(t)$ in $\text{DSPACE}(t)$ enthalten ist.

Satz 5.65. Für jede raumkonstruierbare Funktion $t \geq \log$ gilt:

$$\text{ATIME}(t) \subseteq \text{DSPACE}(t).$$

Beweis. Sei A eine beliebige Menge in $\text{ATIME}(t)$, und sei M eine ATM mit k Bändern, die A in der Zeit t akzeptiert. Konstruiere wie folgt eine DTM N mit $k + 1$ Bändern, die A im Raum t entscheidet. Mit den ersten k Bändern von N wird die

Berechnung von M simuliert, und das $(k+1)$ -te Band von N wird verwendet, um mit Tiefensuche zu entscheiden, ob bei einer Eingabe x der Berechnungsbaum $M(x)$ einen akzeptierenden alternierenden Teilbaum hat oder nicht.

Bei Eingabe x der Länge n arbeitet N wie folgt:

Phase 1: Initialisierung. N markiert genau $t(n) + 1$ Bandzellen auf ihrem $(k+1)$ -ten Band, welches in zwei Spuren unterteilt ist. Dann schreibt N den lexikographisch ersten Pfadnamen, $0^{t(n)}$, in die Bandzellen $1, 2, \dots, t(n)$ der ersten Spur und lässt die nullte Bandzelle dieser Spur leer; siehe Abbildung 5.15.

Spur 1:	□	0	0	0	...	0	0	0	...
Spur 2:	#				...			\$...
Bandzelle mit Nr.:	0	1	2	3	...			$t(n)$...

Abb. 5.15. Initialisierungsphase im Beweis von $\text{ATIME}(t) \subseteq \text{DSPACE}(t)$

Phase 2: Simulation. Unter Verwendung der ersten k ihrer Bänder simuliert N die Berechnung von $M(x)$ entlang des Pfades, der auf der ersten Spur ihres $(k+1)$ -ten Bands steht. Wurde für ein $i \geq 0$ nach dem i -ten Schritt in dieser Simulation die i -te Konfiguration, C_i , auf dem aktuellen Pfad erreicht, so schreibt N in die i -te Bandzelle der zweiten Spur ihres $(k+1)$ -ten Bands das Symbol:

- 0, falls C_i ablehnende Konfiguration,
- 1, falls C_i akzeptierende Konfiguration,
- \vee , falls C_i existenzielle Konfiguration,
- \wedge , falls C_i universelle Konfiguration ist.

Insbesondere enthält die nullte Bandzelle auf der zweiten Spur des $(k+1)$ -ten Bands von N die Information, ob die Startkonfiguration $C_0 = \text{START}_M(x)$ existenziell oder universell ist.

Abbildung 5.16 gibt ein Beispiel mit einer universellen Startkonfiguration und einer akzeptierenden Endkonfiguration nach $t(n) - 1$ Takten. Wenn bis zum $t(n)$ -ten Takt der Simulation keine Endkonfiguration erreicht wird, bricht N die Simulation von $M(x)$ entlang des aktuellen Pfades ab und schreibt eine „0“ in die $t(n)$ -te Bandzelle der zweiten Spur von Band $k+1$.

Spur 1:	□	0	0	0	...	0	0	0	...
Spur 2:	\wedge	\vee	\vee	\wedge	...	\wedge	1	\$...
Bandzelle mit Nr.:	0	1	2	3	...			$t(n)$...

Abb. 5.16. Simulationsphase im Beweis von $\text{ATIME}(t) \subseteq \text{DSPACE}(t)$

Spur 1:	\square	0	0	0	\dots	0	0	0	\dots
Spur 2:	\wedge	\vee	\vee	\wedge	\dots	\wedge_1	\square	$\$$	\dots
Bandzelle mit Nr.:	0	1	2	3	\dots			$t(n)$	\dots

Abb. 5.17. Auswertungsphase im Beweis von $\text{ATIME}(t) \subseteq \text{DSPACE}(t)$

Phase 3: Auswertung. Wenn N eine „0“ oder eine „1“ in die zweite Spur des $(k+1)$ -ten Bands schreibt, etwa in die i -te Bandzelle, dann ändert N den Inhalt der $(i-1)$ -ten und der i -ten Bandzelle auf dieser Spur gemäß der Funktion e_i , die in (5.45) bis (5.50) unten definiert ist. Sei $\Gamma = \{\square, 0, 1, \wedge, \vee, \wedge_1, \vee_0\}$ ein Alphabet. Um die Bewertungsfunktion eval aus Definition 5.61 zu beschreiben, bildet die Funktion

$$e_i : \Gamma \times \Gamma \rightarrow \Gamma \times \Gamma$$

Paare von Symbolen aus Γ auf Paare von Symbolen aus Γ wie folgt ab:

$$e_i(\wedge, 0) = (0, \square) \quad (5.45)$$

$$e_i(\vee, 1) = (1, \square) \quad (5.46)$$

$$e_i(\wedge, 1) = (\wedge_1, \square) \quad (5.47)$$

$$e_i(\vee, 0) = (\vee_0, \square). \quad (5.48)$$

Die erste Komponente eines solchen Paares entspricht dem Inhalt der $(i-1)$ -ten Bandzelle auf der zweiten Spur des $(k+1)$ -ten Bands von N , und die zweite Komponente entspricht dem Inhalt der i -ten Bandzelle. (5.45) und (5.46) implizieren, dass die Auswertung rekursiv mit e_{i-1} fortgesetzt werden kann, um die Simulation für die $(i-1)$ -te Bandzelle zu evaluieren, und so weiter.

Wenn jedoch (5.47) oder (5.48) anzuwenden sind, dann ist zur Auswertung von C_{i-1} , der $(i-1)$ -ten Konfiguration auf dem aktuell in der ersten Spur des $(k+1)$ -ten Bandes von N notierten Pfad, zusätzliche Information erforderlich. In Fortsetzung des Beispiels aus Abbildung 5.16 zeigt Abbildung 5.17 eine Anwendung von (5.47): $e_{t(n)-1}(\wedge, 1) = (\wedge_1, \square)$.

C_{i-1} hat zwei unmittelbare Folgekonfigurationen, C_i und C_j , im Berechnungsbaum $M(x)$. Allein den Wert $\text{eval}(C_i)$ zu kennen, ist natürlich nicht genug, um $\text{eval}(C_{i-1})$ zu bestimmen; N muss ebenso $\text{eval}(C_j)$ kennen. Daher setzt N die Tiefensuche fort, indem sie den Pfadnamen, der derzeit auf der ersten Spur ihres $(k+1)$ -ten Bands steht, aktualisiert:

- die Bandzellen $0, 1, \dots, i-1$ bleiben unverändert,
- die i -te Bandzelle wird von „0“ zu „1“ aktualisiert und
- die Bandzellen $i+1, i+2, \dots, t(n)$ enthalten jeweils eine „0“.

Da C_{i-1} nicht gespeichert wurde, wird $M(x)$ von Anfang an (siehe Phase 2) entlang des aktualisierten Pfades neu simuliert, um den Wert von $\text{eval}(C_j)$ zu bestimmen. Die Simulation von $M(x)$ und ebenso die Auswertung ist ein rekursiver Prozess. Da Raum jedoch (anders als Zeit) eine wiederverwendbare Ressource ist, kann N diese Rekursion im ausgelegten Raum vollbringen. Sobald sie den

Wert von $\text{eval}(C_j) \in \{0, 1\}$ rekursiv bestimmt hat, aktualisiert N die $(i - 1)$ -te und die i -te Bandzelle auf der zweiten Spur ihres $(k + 1)$ -ten Bands gemäß der folgenden Vorschrift:

$$e_i(\wedge_1, \text{eval}(C_j)) = (\text{eval}(C_j), \square) \quad (5.49)$$

$$e_i(\vee_0, \text{eval}(C_j)) = (\text{eval}(C_j), \square), \quad (5.50)$$

und setzt rekursiv mit e_{i-1} fort, um die Simulation der $(i - 1)$ -ten Bandzelle auszuwerten, und so weiter.

Phase 4: Akzeptierung. Sobald die nullte Bandzelle auf der zweiten Spur des $(k + 1)$ -ten Bands von N in dieser Weise ausgewertet wurde, akzeptiert N die Eingabe x genau dann, wenn diese Bandzelle den Wert „1“ enthält, was gemäß der hier beschriebenen Konstruktion genau dann der Fall ist, wenn $M(x)$ einen akzeptierenden alternierenden Teilbaum besitzt.

Damit ist die Beschreibung von N abgeschlossen. Offenbar arbeitet N im Raum $\mathcal{O}(t(n))$, und es gilt $L(N) = L(M) = A$. Somit ist A in $\text{DSPACE}(t)$. \square

Die zu der Inklusion aus Satz 5.65 umgekehrte Inklusion kann beinahe erreicht werden. Jedoch verlangt die Simulation deterministischen Raumes in alternierender Zeit einen kleinen Preis, nämlich einen Zuwachs der verbrauchten Ressource, der gleichwohl lediglich quadratisch ist.

Satz 5.66. Für jede zeitkonstruierbare Funktion $s \geq \log$ gilt:

$$\text{DSPACE}(s) \subseteq \text{ATIME}(s^2).$$

Beweisskizze. Die Beweisidee beruht auf der Technik zum Beweis des Satzes von Savitch, siehe Satz 3.29. Das Ausformulieren der Details dieses Beweises wird dem Leser als Übung 5.30 überlassen. Hier wird der Beweis nur grob skizziert.

Sei A eine beliebige Menge in $\text{DSPACE}(s)$, und sei M eine DTM, die A im Raum s entscheidet. Nach Satz 3.6 gilt $\text{DSPACE}(s) \subseteq \text{DTIME}(2^{\mathbb{L}_{\text{in}}(s)})$ für $s \geq \log$. Also gibt es eine Konstante c , so dass M in der Zeit $2^{c \cdot s(n)}$ arbeitet. Sei k die kleinste ganze Zahl mit $2^{c \cdot s(n)} \leq 2^k$. Sei ACCEPT_M die eindeutige akzeptierende Endkonfiguration von M bei jeder Eingabe, und sei $\text{START}_M(x)$ die eindeutige Startkonfiguration von M bei Eingabe x . Dann gilt:

$$\begin{aligned} x \in A &\iff \text{START}_M(x) \vdash_M^{2^k} \text{ACCEPT}_M \\ &\iff (\exists i) [\text{START}_M(x) \vdash_M^{2^{k-1}} C_i \text{ und } C_i \vdash_M^{2^{k-1}} \text{ACCEPT}_M]. \end{aligned} \quad (5.51)$$

Konstruiere eine ATM N , die A in der Zeit $\mathcal{O}((s(n))^2)$ akzeptiert, wie folgt. Bei Eingabe x der Länge n , arbeitet N so:

1. Existentielle Ratephase: Gemäß (5.51) wird eine Konfiguration C_i der Größe höchstens $s(n)$ in $\mathcal{O}(s(n))$ Takten existenziell geraten.

2. Universelle Prüfphase: Für eine jede geratene Konfiguration C_i wird universell die Aussage von (5.51) verifiziert:

$$\text{START}_M(x) \vdash_M^{2^{k-1}} C_i \text{ und } C_i \vdash_M^{2^{k-1}} \text{ACCEPT}_M.$$

Zu diesem Zweck werden rekursiv existenzielle Rate- und universelle Prüfphasen wie oben parallel ausgeführt, bis nach höchstens k rekursiven Aufrufen dieser Prozedur nur noch die unmittelbare Überführung $C_j \vdash_M C_\ell$ für Paare von Konfigurationen C_j und C_ℓ verifiziert werden muss.

Da es höchstens $k \in \mathcal{O}(s(n))$ Rate- und Prüfphasen von jeweils $\mathcal{O}(s(n))$ Taktten gibt, benötigt N nicht mehr als $\mathcal{O}((s(n))^2)$ Takte. \square

Aus den Sätzen 5.65 und 5.66 zusammen ergibt sich das folgende Korollar, das insbesondere sagt, dass alternierende Polynomialzeit, also die durch $\text{AP} = \text{ATIME}(\text{IPol})$ definierte Klasse, gleich deterministischem polynomialem Raum ist. Dieses Resultat steht in scharfem Gegensatz zum Fall der nichtdeterministischen Zeit, für den die Inklusion $\text{DSPACE}(\text{IPol}(t)) \subseteq \text{NTIME}(\text{IPol}(t))$ nicht bekannt ist. Die berühmteste Inkarnation dieses wichtigen offenen Problems ist die Frage, ob PSPACE in NP enthalten ist oder nicht.

Korollar 5.67. Für jede raumkonstruierbare Funktion $t \geq \log$ gilt $\text{ATIME}(\text{IPol}(t)) = \text{DSPACE}(\text{IPol}(t))$. Insbesondere gilt $\text{AP} = \text{PSPACE}$.

Indem man „alternierende“ Varianten NP-vollständiger Probleme definiert, können mittels Korollar 5.67 PSPACE-vollständige Probleme gefunden werden. Ein solches Beispiel ist bereits aus Satz 5.36 bekannt: QBF ist PSPACE-vollständig. QBF kann als eine „alternierende“ Verallgemeinerung des NP-vollständigen Erfüllbarkeitsproblems aufgefasst werden.

Korollar 5.67 sagt, dass die alternierende Turingmaschine ein vernünftiges Modell paralleler Berechnung ist, denn sie erfüllt das Kriterium von Cook:

Parallele Zeit entspricht sequenziellem (d.h. deterministischem) Raum.

Das Wort „entspricht“ bedeutet hier, dass es sich um einen polynomialen „Trade-off“ zwischen zwei Komplexitätsmaßen handelt.

Nun wenden wir uns dem *Tradeoff* zwischen den Komplexitätsmaßen des alternierenden Raumes und der deterministischen Zeit zu. Da nach Definition $\text{NSPACE}(s)$ in $\text{ASPACE}(s)$ enthalten ist, verschärft Satz 5.68 die aus Korollar 3.28 bekannte Inklusion $\text{NSPACE}(s) \subseteq \text{DTIME}(2^{\text{Lin}(s)})$.

Satz 5.68. Ist $s \geq \log$ in der Zeit $2^{\text{Lin}(s)}$ konstruierbar, so gilt:

$$\text{ASPACE}(s) \subseteq \text{DTIME}(2^{\text{Lin}(s)}).$$

Beweisskizze. Der Beweis folgt den Zeilen des Beweises von „ $\text{NL} \subseteq \text{P}$ “ aus Satz 3.27. Der Unterschied ist lediglich, dass wir nun nicht von einer NTM, sondern von einer ATM M ausgehen, deren Berechnung bei Eingabe x deterministisch

in der Zeit $2^{\text{Lin}(s)}$ simuliert werden soll. Deshalb muss nicht nur ein akzeptierender Pfad im Berechnungsbaum von $M(x)$ gefunden werden, falls einer existiert, sondern die deterministische Simulation muss entscheiden, ob $M(x)$ einen akzeptierenden alternierenden Teilbaum hat oder nicht.

Es ist nicht schwer, eine DTM N zu entwerfen, die eine gegebene Eingabe x genau dann akzeptiert, wenn $\text{eval}(\text{START}_M(x)) = 1$ gilt, wobei eval die Auswertungsfunktion aus Definition 5.61 und $\text{START}_M(x)$ die eindeutige Startkonfiguration von M bei Eingabe x ist. Die Details der Konstruktion von N werden dem Leser als Übung 5.31 überlassen. \square

Für das alternierende Raummaß kann die umgekehrte Inklusion zu der aus Satz 5.68 gezeigt werden. Der Beweis von Satz 5.69 macht in überaus massiver Weise Gebrauch von Parallelität.

Satz 5.69. Für jede Funktion $s \geq \log$ gilt $\text{DTIME}(2^{\text{Lin}(s)}) \subseteq \text{ASPACE}(s)$.

Beweis. Sei A eine beliebige Menge in $\text{DTIME}(2^{\text{Lin}(s)})$, und sei M eine Ein-Band-DTM, die A in der Zeit $2^{\text{Lin}(s)}$ entscheidet. Die Annahme, dass M nur ein Band hat, kann ohne Beschränkung der Allgemeinheit gemacht werden, da der quadratische Zuwachs der Ressource Zeit, der von der Simulation einer mehrbändrigen DTM durch eine einbändrige DTM verursacht wird (siehe Übung 3.8(c)) für Ressourcenfunktionen in $2^{\text{Lin}(s)}$ vernachlässigbar ist.

Das Ziel besteht darin, eine ATM N zu konstruieren, die A im Raum $s(n)$ durch Simulation der Berechnung von M bei einer Eingabe x der Länge n akzeptiert. Abbildung 5.18 zeigt die Bewegung des Kopfes von M als eine Funktion der Zeit.

Jeder Schritt von $M(x)$ wird durch ein Tripel (z, t, p) bestimmt, wobei z der aktuelle Zustand von M ist, t der aktuelle Takt von M und p die Nummer der Bandzelle, die aktuell vom Kopf der Maschine M gelesen wird. Bei der Simulation der Berechnung von M kann die ATM N nicht die gesamte Bandinschrift von M speichern, die dafür viel zu lang ist: M kann ihren Kopf in $2^{c \cdot s(n)}$ Takten um bis zu $2^{c \cdot s(n)}$ Bandzellen weiterbewegen. Jedoch kann N massiven Gebrauch von ihrer Parallelität machen: N *rät existenziell* den Inhalt einer Bandzelle von M und rechnet unter der Annahme weiter, dass dieses Symbol korrekt geraten wurde, während sie parallel dazu *universell überprüft*, ob diese Vermutung korrekt war. Nimm beispielsweise an, $N(x)$ hätte die Berechnung von $M(x)$ bis zum Takt t_1 bereits simuliert und N hätte geraten, dass der Kopf von M aktuell das Symbol α an Position p liest; siehe Abbildung 5.18. Parallel zur weiteren Simulation, die mit dieser Vermutung weiterarbeitet, überprüft N die Korrektheit dieser Vermutung. Zu diesem Zweck startet N die Berechnung von $M(x)$ neu von Anfang an, um festzustellen, (a) ob es einen Takt $t_0 < t_1$ gibt, in dem das Symbol α tatsächlich in die Bandzelle mit Nummer p geschrieben wurde, und (b) ob t_0 der letzte Zeitpunkt vor Takt t_1 war, zu dem der Kopf von M die Position p auf dem Band besuchte.

Um dies etwas detaillierter zu beschreiben, nehmen wir an, dass Z die Zustandsmenge von M und Γ ihr Arbeitsalphabet ist. Dann habe N für jedes $z \in Z$ und jedes $\alpha \in \Gamma$ Zustände der Form z, z_α und α , deren Zweck unten genauer erläutert wird. Außerdem sei N mit einem zusätzlichen Arbeitsband ausgestattet, um sich bei der

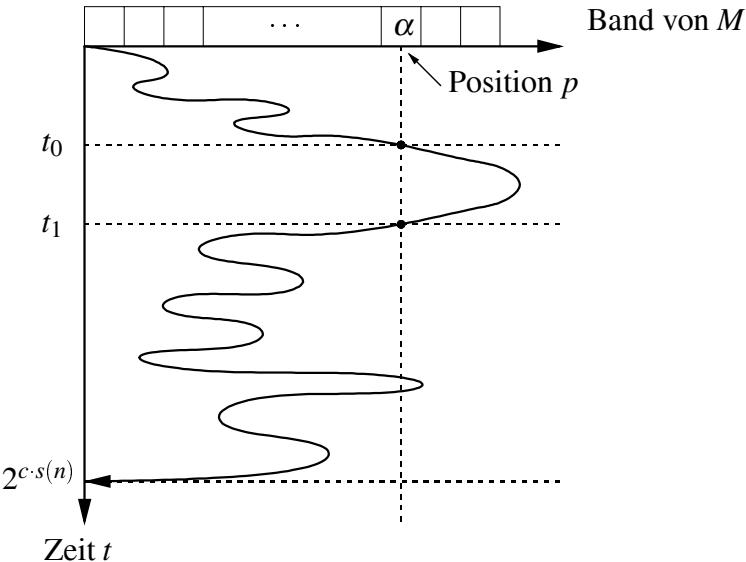


Abb. 5.18. Kopfbewegung der DTM M im Beweis von $\text{DTIME}(2^{\text{Lin}(s)}) \subseteq \text{ASPACE}(s)$

Simulation von $M(x)$ den aktuellen Takt t und die aktuelle Kopfposition p merken und den bisherigen Verlauf der Berechnung zurückverfolgen zu können. Das Speichern dieser Zahlen in Binärdarstellung erfordert nicht mehr als $\mathcal{O}(s(n))$ Raum, da sowohl die Anzahl der Takte von $M(x)$ als auch die Anzahl der möglichen Kopfpositionen von M während der Berechnung nach oben durch $2^{c \cdot s(n)}$ für eine geeignete Konstante c beschränkt sind.

Sei etwa ein Tripel (z, t, p) gegeben, und sei $z\alpha \rightarrow \hat{z}\hat{\alpha}\mu$ eine Überführung von M . Dabei sind z und \hat{z} Zustände in Z , α und $\hat{\alpha}$ Symbole in Γ , und $\mu \in \{-1, 0, 1\}$ gibt die entsprechende Kopfbewegung von M an. Ein Blick auf Abbildung 5.19, welche auf dem allgemeinen Muster von Abbildung 5.14 beruht, zeigt das „parallele“ Verhalten von N . Zunächst rät N in einer existentiellen Phase das von M 's Kopf aktuell gelesene Symbol gemäß dem gegebenen Tripel (z, t, p) , d.h., in der Simulation von M liegt aktuell der Zustand z vor, sie ist in Takt t und an Position p .

Angenommen, N hat das Symbol α geraten. Dann geht N in den universellen Zustand z_α über, um universell wie in Abbildung 5.19 zu verzweigen. Der rechte Teilbaum unter (z_α, t, p) , dessen Wurzel mit $(\hat{z}, t+1, p+\mu)$ markiert ist, setzt die Simulation von $M(x)$ unter der Annahme fort, dass das Symbol α korrekt geraten wurde, und wendet entsprechend die Überführung $z\alpha \rightarrow \hat{z}\hat{\alpha}\mu$ an. Der linke Teilbaum unter (z_α, t, p) , dessen Wurzel mit (α, t, p) markiert ist, verifiziert parallel dazu, dass das Symbol α tatsächlich korrekt geraten wurde. Zu diesem Zweck geht N in den Zustand α über, speichert die aktuellen Werte für Takt t und Position p auf einem zusätzlichen Arbeitsband und löst die Neuberechnung von $M(x)$ von Anfang an bis

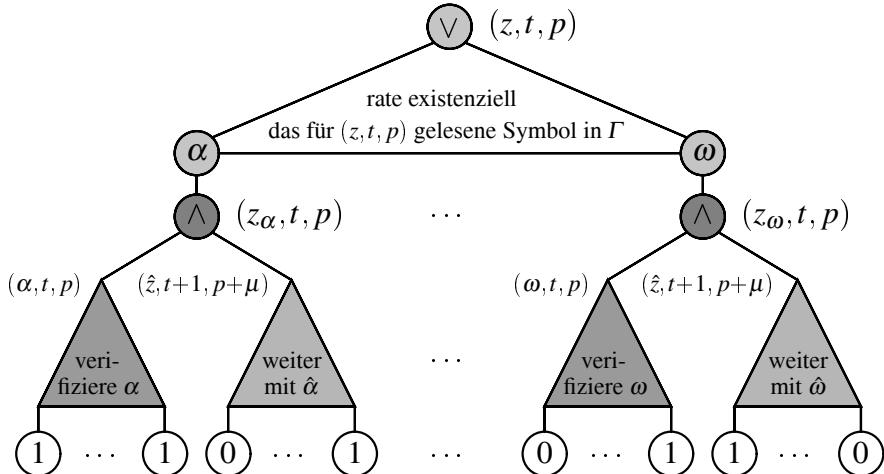


Abb. 5.19. Berechnungsbaum der ATM N im Beweis von $\text{DTIME}(2^{\text{Lin}(s)}) \subseteq \text{ASPACE}(s)$

zum Takt t aus, in welchem ein Symbol in Position p geschrieben wird. Die bei α ausgelöste Neuberechnung akzeptiert genau dann, wenn dieses Symbol schließlich mit α übereinstimmt.

Man beachte hier, dass jede solche Neuberechnung permanent neue Verifikationsprozesse und damit neue Neuberechnungen auslöst, welche rekursiv wiederum weitere Verifikationsprozesse und damit weitere Neuberechnungen auslösen, und so weiter. Da jedoch jede solche Neuberechnung nur ihr „eigenes“ Tripel (α, t, p) und nicht die der übergeordneten Neuberechnungen speichert, terminiert diese Prozedur (wegen der Wiederverwendbarkeit der Ressource Raum) insgesamt im Raum $\mathcal{O}(s(n))$. Somit ist A in $\text{ASPACE}(s)$, womit der Satz bewiesen ist. \square

Aus den Sätzen 5.68 und 5.69 zusammen ergibt sich das folgende Korollar, das insbesondere sagt, dass alternierender logarithmisches Raum, also die durch $\text{AL} = \text{ASPACE}(\log)$ definierte Klasse, gleich deterministischer Polynomialzeit ist. Wieder steht dieses Resultat in scharfem Gegensatz zum Fall des nichtdeterministischen Raumes, für den die Inklusion $\text{DTIME}(2^{\text{Lin}(s)}) \subseteq \text{NSPACE}(s)$ nicht bekannt ist. Die berühmteste Inkarnation dieses wichtigen offenen Problems ist die Frage, ob P in NL enthalten ist oder nicht.

Korollar 5.70. $\text{ASPACE}(s) = \text{DTIME}(2^{\text{Lin}(s)})$ für jede Funktion $s \geq \log$, die in der Zeit $2^{\text{Lin}(s)}$ konstruierbar ist. Insbesondere gilt $\text{AL} = P$.

Indem man „alternierende“ Varianten NL -vollständiger Probleme definiert, können mittels Korollar 5.70 Probleme gefunden werden, die \leq_m^{\log} -vollständig in P sind. Ein solches Beispiel wird unten angegeben: das „alternierende“ Grapherreichbar-

keitsproblem. Wie in Satz 3.43 gezeigt wurde, ist das Grapherreichbarkeitsproblem, GAP, \leq_m^{\log} -vollständig in NL.

Definition 5.71 (Alternierendes Grapherreichbarkeitsproblem). Ein alternierender Graph ist ein gerichteter azyklischer Graph, dessen innere Knoten (d.h. dessen Knoten mit auslaufenden Kanten) entweder existenziell (d.h. mit \vee markiert) oder universell (d.h. mit \wedge markiert) sind. Definiere den Begriff der Erreichbarkeit in einem alternierenden Graphen G induktiv wie folgt. Für jeden Knoten $x \in V(G)$ gilt:

- x ist von x erreichbar;
- ist x existenziell, also mit \vee markiert, so ist ein Knoten $z \in V(G)$ von x genau dann erreichbar, wenn es einen Knoten $y \in V(G)$ gibt, so dass $(x, y) \in E(G)$ eine Kante in G ist und z von y erreichbar ist;
- ist x universell, also mit \wedge markiert, so ist ein Knoten $z \in V(G)$ von x genau dann erreichbar, wenn für alle Knoten $y \in V(G)$ mit $(x, y) \in E(G)$ gilt, dass z von y erreichbar ist.

Definiere das alternierende Grapherreichbarkeitsproblem durch

$$\text{AGAP} = \left\{ \langle G, s, t \rangle \middle| \begin{array}{l} G \text{ ist ein alternierender Graph mit } s \text{ und } t \\ \text{in } V(G), \text{ so dass } t \text{ von } s \text{ in } G \text{ erreichbar ist} \end{array} \right\}.$$

Satz 5.72. AGAP ist \leq_m^{\log} -vollständig in P.

Beweis. Satz 3.43 sagte, dass GAP \leq_m^{\log} -vollständig in NL ist. Ein zu diesem Beweis analoges Argument zeigt, dass AGAP \leq_m^{\log} -vollständig in AL ist. Korollar 5.70 schließt dann den Beweis ab. \square

5.7 Die Low- und die High-Hierarchie in NP

Sei H eine NP-vollständige Menge, und seien M und N eine DPOTM bzw. eine NPOTM, die H als Orakelmenge verwenden. Wie viel zusätzliche Berechnungskraft gibt H den Maschinen M und N ? Da H NP-vollständig ist (und da $\leq_m^P \subseteq \leq_T^P$, siehe Behauptung 5.28.1), gilt $NP \subseteq P^H$ und $NP^{NP} \subseteq NP^H$. Also „springen“ sowohl M als auch N mittels H eine ganze Stufe in der Polynomialzeit-Hierarchie nach oben: M^H kann jedes NP-Problem und N^H kann jedes Σ_2^P -Problem lösen. In diesem Sinn sind die NP-vollständigen Mengen wie etwa H die mächtigsten (oder „highest“) Mengen in NP.

Nun betrachte eine Menge L in P als ein Orakel für M und N . Nach Behauptung 5.28 gilt $P^P = P$ und $NP^P = NP$. Also stellt L den Maschinen M bzw. N nicht mehr zusätzliche Berechnungskraft als die leere Menge zur Verfügung: L ist vollständig nutzlos als Orakelmenge sowohl für M als auch für N . In diesem Sinn sind die P-Mengen wie etwa L die schwächsten (oder „lowest“) Mengen in NP. Insbesondere kann, unter der Annahme $P \neq NP$, keine NP-vollständige Menge in P sein: Die Klassen der NP-Mengen, die im o.g. Sinn low bzw. high sind, sind disjunkt.

Was aber ist zu erwarten, wenn unsere Orakelmaschine keine DPOTM oder NPOTM, sondern eine Σ_k^p -Orakelmaschine für ein $k > 1$ ist? Das heißt, welche Mengen sind *low* und welche Mengen sind *high* für Σ_k^p , die k -te Stufe der Polynomialzeit-Hierarchie?

Schöning [Sch83] führte die Low- und die High-Hierarchie in NP ein, die beide eine Feinstruktur in dieser zentralen Komplexitätsklasse bilden. Mit der *Lowness* und *Highness* kann die Nützlichkeit einer NP-Menge als ein Orakel für die Stufen der Polynomialzeit-Hierarchie gemessen werden.

Definition 5.73 (Low-Hierarchie und High-Hierarchie in NP).

1. Die Low-Hierarchie in NP ist definiert durch

$$\text{Low}_k = \{L \mid L \in \text{NP} \text{ und } \Sigma_k^{p,L} \subseteq \Sigma_k^p\} \quad \text{für jedes } k \geq 0, \text{ und}$$

$$\text{LH} = \bigcup_{k \geq 0} \text{Low}_k.$$

2. Die High-Hierarchie in NP ist definiert durch

$$\text{High}_k = \{H \mid H \in \text{NP} \text{ und } \Sigma_{k+1}^p \subseteq \Sigma_k^{p,H}\} \quad \text{für jedes } k \geq 0, \text{ und}$$

$$\text{HH} = \bigcup_{k \geq 0} \text{High}_k.$$

Die niedrigen Stufen der Low- und der High-Hierarchie können durch wohlbekannte Komplexitätsklassen bzw. durch Klassen vollständiger Mengen bezüglich bestimmter Reduzierbarkeiten charakterisiert werden. Eine solche Reduzierbarkeit wird unten eingeführt.

Definition 5.74 (Starke Nichtdeterministische Turing-Reduzierbarkeit). Für ein Alphabet $\Sigma = \{0, 1\}$ seien A und B Mengen von Wörtern über Σ , und \mathcal{C} sei eine Komplexitätsklasse.

1. Definiere die starke nichtdeterministische polynomialzeit-beschränkte Turing-Reduzierbarkeit, bezeichnet mit \leq_{sT}^{NP} , wie folgt: $A \leq_{sT}^{\text{NP}} B$ genau dann, wenn es eine NPOTM M mit drei Typen von Endzuständen gibt, nämlich akzeptierende Zustände, ablehnende Zustände und einen Zustand $s_?$ (welcher für „weiß nicht“ steht), so dass gilt:

- a) Ist $x \in A$, so hat $M^B(x)$ mindestens einen akzeptierenden Pfad und keinen ablehnenden Pfad.
- b) Ist $x \notin A$, so hat $M^B(x)$ mindestens einen ablehnenden Pfad und keinen akzeptierenden Pfad.

In beiden Fällen darf $M^B(x)$ Pfade haben, die im Zustand $s_?$ halten.

- 2. Eine Menge B ist \leq_{sT}^{NP} -hart für \mathcal{C} , falls $A \leq_{sT}^{\text{NP}} B$ für jede Menge $A \in \mathcal{C}$ gilt.
- 3. Eine Menge B ist \leq_{sT}^{NP} -vollständig für \mathcal{C} , falls $B \leq_{sT}^{\text{NP}}$ -hart für \mathcal{C} ist und $B \in \mathcal{C}$.

Das folgende Lemma geht auf Selman [Sel78] zurück. Es liefert äquivalente Bedingungen für die \leq_{sT}^{NP} -Reduzierbarkeit, analog zu den entsprechenden Charakterisierungen der \leq_T^p -Reduzierbarkeit: $A \leq_T^p B$ ist äquivalent zu $A \in P^B$, was wiederum äquivalent zu $P^A \subseteq P^B$ ist. Der Beweis von Lemma 5.75 wird dem Leser als Übung 5.32(a) überlassen.

Lemma 5.75. *Die folgenden drei Aussagen sind paarweise äquivalent:*

1. $A \leq_{sT}^{\text{NP}} B$.
2. $A \in \text{NP}^B \cap \text{coNP}^B$.
3. $\text{NP}^A \subseteq \text{NP}^B$.

Satz 5.76. 1. $\text{Low}_0 = \text{P}$.

$$2. \text{Low}_1 = \text{NP} \cap \text{coNP}.$$

$$3. \text{High}_0 = \{H \mid H \text{ ist } \leq_T^p\text{-vollständig für NP}\}.$$

$$4. \text{High}_1 = \{H \mid H \text{ ist } \leq_{sT}^{\text{NP}}\text{-vollständig für NP}\}.$$

Beweis. 1. Nach Definition ist eine NP-Menge L genau dann in Low_0 , wenn $\text{P}^L \subseteq \text{P}$, was wiederum dazu äquivalent ist, dass L in P liegt. Somit ist $\text{Low}_0 = \text{P}$.

2. Nach Definition ist eine NP-Menge L genau dann in Low_1 , wenn $\text{NP}^L \subseteq \text{NP} = \text{NP}^\emptyset$, was nach Lemma 5.75 genau dann der Fall ist, wenn $L \in \text{NP}^\emptyset \cap \text{coNP}^\emptyset = \text{NP} \cap \text{coNP}$. Somit ist $\text{Low}_1 = \text{NP} \cap \text{coNP}$.

3. Nach Definition ist eine NP-Menge H genau dann in High_0 , wenn $\text{NP} \subseteq \text{P}^H$, was wiederum dazu äquivalent ist, dass $H \leq_T^p\text{-vollständig für NP}$ ist. Somit ist $\text{High}_0 = \{H \mid H \text{ ist } \leq_T^p\text{-vollständig für NP}\}$.

4. Nach Definition ist eine NP-Menge H genau dann in High_1 , wenn $\text{NP}^{\text{NP}} \subseteq \text{NP}^H$, was wiederum genau dann wahr ist, wenn $\text{NP}^{\text{SAT}} \subseteq \text{NP}^H$. Nach Lemma 5.75 ist dies genau dann der Fall, wenn $\text{SAT} \leq_{sT}^{\text{NP}} H$.

Da $\text{SAT} \leq_m^p\text{-vollständig für NP}$ ist und da für beliebige Mengen A und B die Implikation „ $A \leq_m^p B \implies A \leq_{sT}^{\text{NP}} B$ “ gilt, folgt die \leq_{sT}^{NP} -Vollständigkeit von H für NP. Somit ist $\text{High}_1 = \{H \mid H \text{ ist } \leq_{sT}^{\text{NP}}\text{-vollständig für NP}\}$. \square

Einige grundlegenden Eigenschaften der Low- und der High-Hierarchie in NP werden unten aufgelistet. Abbildung 5.20 zeigt die Inklusionsstruktur der Stufen dieser beiden Hierarchien, die NP „verfeinern“. Unterschiedliche Helligkeitsgrade der schattierten Regionen in Abbildung 5.20 zeigen Inklusionen an: Dunklere Klassen sind in helleren Klassen enthalten.

Satz 5.77. 1. $\text{Low}_0 \subseteq \text{Low}_1 \subseteq \dots \subseteq \text{Low}_k \subseteq \dots \subseteq \text{LH} \subseteq \text{NP}$.

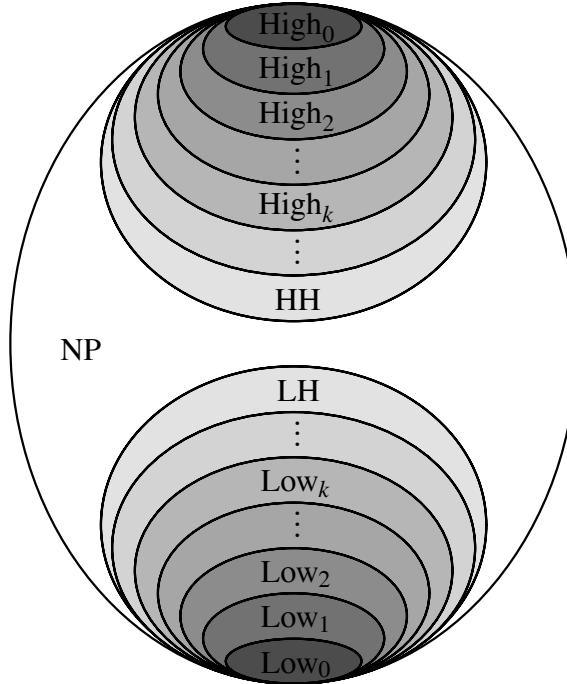
$$2. \text{High}_0 \subseteq \text{High}_1 \subseteq \dots \subseteq \text{High}_k \subseteq \dots \subseteq \text{HH} \subseteq \text{NP}.$$

3. Für jedes $k \geq 0$ ist $\text{Low}_k \cap \text{High}_k \neq \emptyset$ genau dann, wenn die Polynomialzeit-Hierarchie auf die k -te Stufe kollabiert: $\Sigma_k^p = \Sigma_{k+1}^p = \dots = \text{PH}$.

Beweis. 1. Fixiere ein $k \geq 0$. Sei L eine beliebige Menge in Low_k . Nach Definition ist $\Sigma_k^{p,L} \subseteq \Sigma_k^p$. Folglich gilt $\Sigma_k^{p,L} = \text{NP}^{\Sigma_k^{p,L}} \subseteq \text{NP}^{\Sigma_k^p} = \Sigma_{k+1}^p$, woraus folgt, dass L in Low_{k+1} ist.

2. Fixiere ein $k \geq 0$. Sei H eine beliebige Menge in High_k . Nach Definition ist $\Sigma_{k+1}^p \subseteq \Sigma_k^{p,H}$. Folglich gilt $\Sigma_{k+1}^p = \text{NP}^{\Sigma_{k+1}^p} \subseteq \text{NP}^{\Sigma_k^{p,H}} = \Sigma_{k+1}^{p,H}$, woraus folgt, dass H in High_{k+1} ist.

3. Für die Implikation von links nach rechts sei angenommen, dass $\text{Low}_k \cap \text{High}_k$ nicht leer ist. Sei A eine Menge in $\text{Low}_k \cap \text{High}_k$. Dann gilt $\Sigma_{k+1}^p \subseteq \Sigma_k^{p,A} \subseteq \Sigma_k^p$, und der Kollaps folgt aus Satz 5.33.

**Abb. 5.20.** Die Low- und die High-Hierarchie in NP

Für die Implikation von rechts nach links sei der Kollaps $\Sigma_{k+1}^p = \Sigma_k^p$ angenommen. Dann gilt für jede Menge A in NP: $\Sigma_{k+1}^p = \Sigma_k^p \subseteq \Sigma_k^{p,A} \subseteq \Sigma_{k+1}^p$. Folglich ist jede NP-Menge gleichzeitig in Low_k und in High_k . Somit ist $\text{NP} = \text{Low}_k = \text{High}_k$. Insbesondere ist $\text{Low}_k \cap \text{High}_k$ nicht leer. \square

Korollar 5.78. 1. Für jedes $k \geq 0$ ist entweder $\text{Low}_k \cap \text{High}_k = \emptyset$ oder $\text{NP} = \text{Low}_k = \text{High}_k$.

2. Die Polynomialzeit-Hierarchie ist genau dann unendlich, wenn $\text{LH} \cap \text{HH} = \emptyset$.

Falls die Polynomialzeit-Hierarchie bis zu ihrer $(k+1)$ -ten Stufe echt ist, d.h., falls $\Sigma_k^p \neq \Sigma_{k+1}^p$, dann sagen Satz 5.77 und die erste Aussage von Korollar 5.78 oben, dass Low_k und High_k disjunkte Teilklassen von NP sind. Gibt es Mengen in NP, die weder in Low_k noch in High_k sind? Ebenso sagt die zweite Aussage von Korollar 5.78, dass falls die Polynomialzeit-Hierarchie echt unendlich ist, dann sind die Hierarchien LH und HH disjunkte Teilklassen von NP. Gibt es Mengen in NP, die weder in LH noch in HH sind?

Diese Fragen werden später insofern beantwortet, als dass sie durch Kollapsen und Separationen der Stufen der Polynomialzeit-Hierarchie beschrieben werden

können. Um diese Resultate zu beweisen, werden die Stufen der Low- und der High-Hierarchie zunächst mit Hilfe von \leq_m^p -Reduktionen und dem unten definierten iterierten K-Operator charakterisiert.

Schönings [Sch83] ursprüngliche Definition der Low- und der High-Hierarchie unterscheidet sich etwas von der in Definition 5.73. Er definiert eine komplexitätstheoretische Version des durch das Halteproblem definierten „Sprung“-Operators (englisch: *jump operator*) der Berechenbarkeitstheorie.⁹ Die entscheidende Eigenschaft des Halteproblems ist dabei, dass es ein RE-vollständiges Problem ist, wobei RE die Klasse der rekursiv aufzählbaren Mengen ist und sich Vollständigkeit auf die Many-one-Reduzierbarkeit im rekursionstheoretischen Sinn bezieht (d.h., die Reduktion muss berechenbar, aber nicht unbedingt in Polynomialzeit berechenbar sein).

Das komplexitätstheoretische Analogon von RE ist NP; siehe z.B. die Sätze 5.24 und 2.20. Um also ein komplexitätstheoretisches Analogon des Sprung-Operators zu definieren, genügt es, irgendeine \leq_m^p -vollständige Menge für NP zu wählen und sie zu relativieren. Dies tun wir in Definition 5.79 unten, die den K-Operator einführt, welcher auf der generischen \leq_m^p -vollständigen Menge K für NP beruht und definiert ist durch

$$K = \left\{ \langle M, x, 1^t \rangle \mid \begin{array}{l} M \text{ ist eine geeignet als Wort codierte NTM, die} \\ \text{die Eingabe } x \text{ in } t \text{ oder weniger Takten akzeptiert} \end{array} \right\}.$$

Definition 5.79 (K-Operator und Iterierter K-Operator). Sei A eine Menge.

1. Definiere die Menge $K(A)$, die Anwendung des K-Operators auf A, wie folgt:

$$K(A) = \left\{ \langle M, x, 1^t \rangle \mid \begin{array}{l} M \text{ ist eine geeignet als Wort codierte} \\ \text{NOTM, die mit Orakel } A \text{ die Eingabe} \\ x \text{ in } t \text{ oder weniger Takten akzeptiert} \end{array} \right\}.$$

2. Definiere die Anwendung des iterierten K-Operators auf A induktiv durch:

$$\begin{aligned} K^0(A) &= A; \\ K^n(A) &= K(K^{n-1}(A)) \quad \text{für } n \geq 1. \end{aligned}$$

Ist $A = \emptyset$, so schreiben wir $K^n = K^n(\emptyset)$.

Eine Alternative zur obigen Definition wäre, eine *natürliche* \leq_m^p -vollständige Menge für NP, wie etwa SAT, zu wählen und eine geeignet relativierte Version davon zu definieren. Schöning [Sch81] hat diesen Zugang (und dabei das Erfüllbarkeitsproblem) gewählt, der jedoch etwas komplizierter ist und einige technische Raffinessen aufweist.

Lemma 5.80 stellt einige nützliche Eigenschaften des K-Operators zusammen. So fügt etwa die dritte Aussage von Lemma 5.80 eine weitere äquivalente Bedingung zu den Charakterisierungen der \leq_{ST}^{NP} -Reduzierbarkeit aus Lemma 5.75 hinzu.

⁹ Genauer gesagt, ist der *Sprung einer Menge A*, bezeichnet mit A' , definiert als das Halteproblem relativ zu A; siehe H. Rogers' Buch [Rog67] für mehr Hintergrundinformation.

- Lemma 5.80.** 1. Für jede Menge A ist die Menge $K(A) \leq_m^p$ -vollständig für NP^A .
 Falls insbesondere $A = \emptyset$ gilt, ist K eine \leq_m^p -vollständige Menge für NP .
 2. Für jede Menge A gilt $A \leq_m^p K(A)$.
 3. Für je zwei Mengen A und B gilt $A \leq_{ST}^{NP} B$ genau dann, wenn $K(A) \leq_m^p K(B)$.

Beweis. 1. Der Beweis der ersten Aussage des Lemmas wird dem Leser als Übung 5.33(a) überlassen.

2. Definiere die folgende NOTM M : Bei Eingabe x fragt M ihr Orakel nach x . Ist die Antwort „ja“, so akzeptiert M ; andernfalls lehnt M ab. Es folgt $A = L(M^A)$ für jede Menge A . Offenbar arbeitet M in der Zeit p für ein Polynom p . Somit ist die durch $f(x) = \langle M, x, 1^{p(|x|)} \rangle$ definierte Funktion f eine \leq_m^p -Reduktion von A auf $K(A)$.

3. Für die Richtung von rechts nach links nehmen wir an, dass $K(A) \leq_m^p K(B)$ durch eine Reduktion f in FP bezeugt wird. Sei $\langle \cdot, \cdot, \cdot \rangle : \Sigma^* \times \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ eine Paarungsfunktion, die sich in Polynomialzeit berechnen lässt und polynomialzeit-berechenbare Projektionen π_1, π_2 und π_3 hat. Seien M und \bar{M} die trivialen NPOTMs mit $A = L(M^A)$ bzw. $\bar{A} = L(\bar{M}^A)$, und sei $p \in \text{IPol}$ die Zeitschranke sowohl für M als auch für \bar{M} . Um $A \leq_{ST}^{NP} B$ zu zeigen, definiere eine NPOTM N , die bei Eingabe x folgendermaßen arbeitet:

Schritt 1: N berechnet das Wort $y = f(\langle M, x, 1^{p(|x|)} \rangle)$ in Σ^* . Dann überprüft N , ob y die Form $y = \langle \pi_1(y), \pi_2(y), \pi_3(y) \rangle$ hat, wobei $M_1 = \pi_1(y)$ die syntaktisch korrekte Darstellung einer NOTM codiert, $x_1 = \pi_2(y)$ ein Wort ist und $t_1 = |\pi_3(y)|$ eine Zahl in \mathbb{N} . Wenn y nicht die gewünschte Form hat, lehnt N ab. Andernfalls simuliert N nichtdeterministisch M_1 mit Orakel B bei Eingabe x_1 für t_1 Takte. Auf den Pfaden in der Simulation von $M_1^B(x_1)$, die nach t_1 oder weniger Takten akzeptieren, akzeptiert auch N . Auf den Pfaden, die nach t_1 oder weniger Takten nicht akzeptieren, geht N zu Schritt 2 weiter.

Schritt 2: Wie in Schritt 1 berechnet N $z = f(\langle \bar{M}, x, 1^{p(|x|)} \rangle)$ und überprüft, ob das Wort $z = \langle \pi_1(z), \pi_2(z), \pi_3(z) \rangle$ die syntaktisch korrekte Form hat. Seien $M_2 = \pi_1(z)$, $x_2 = \pi_2(z)$ und $t_2 = |\pi_3(z)|$. Wie in Schritt 1 simuliert N nicht-deterministisch die Berechnung von M_2 bei Eingabe x_2 für t_2 Takte, wobei sie B als Orakel benutzt. Auf jedem akzeptierenden Pfad in dieser Simulation lehnt N ab. Auf allen anderen Pfaden hält N im „weiß nicht“-Zustand s ?

Falls x in $A = L(M^A)$ ist, ist $\langle M, x, 1^{p(|x|)} \rangle$ in $K(A)$, woraus folgt, dass das Tripel $f(\langle M, x, 1^{p(|x|)} \rangle) = \langle M_1, x_1, 1^{t_1} \rangle$ in $K(B)$ ist. Also akzeptiert $M_1^B(x_1)$ in der Zeit t_1 auf einem Pfad. Nach Konstruktion akzeptiert auch $N^B(x)$ auf einem Pfad in Schritt 1. Weil andererseits $x \notin \bar{A} = L(\bar{M}^A)$ gilt, ergibt sich $\langle M, x, 1^{p(|x|)} \rangle \notin K(A)$. Es folgt, dass $f(\langle M, x, 1^{p(|x|)} \rangle) = \langle M_2, x_2, 1^{t_2} \rangle$ nicht in $K(B)$ ist. Nach Konstruktion hat $N^B(x)$ keine ablehnenden Pfade in Schritt 2.

Ein analoges Argument zeigt, dass $x \notin A$ impliziert, dass $N^B(x)$ mindestens einen ablehnenden Pfad in Schritt 2 hat, jedoch keinen akzeptierenden Pfad in Schritt 1. Somit bezeugt N , dass $A \leq_{ST}^{NP} B$ gilt.

Für die Richtung von links nach rechts nehmen wir an, dass $A \leq_{ST}^{NP} B$ gilt. Nach Lemma 5.75 ist A in $NP^B \cap \text{co}NP^B$. Seien M und \bar{M} NPOTMs, so dass $A = L(M^B)$ und $\bar{A} = L(\bar{M}^B)$ gilt. Das Ziel ist, eine Reduktion $f \in \text{FP}$ zu konstruieren, die $K(A)$

auf $K(B) \leq_m^p$ -reduziert. Das heißt, f bildet Tripel der Form $\langle N_1, x_1, 1^{t_1} \rangle$ auf Tripel der Form $\langle N_2, x_2, 1^{t_2} \rangle$ ab, so dass gilt:

$$\begin{aligned} N_1^A \text{ akzeptiert } x_1 \text{ in höchstens } t_1 \text{ Schritten} \\ \iff N_2^B \text{ akzeptiert } x_2 \text{ in höchstens } t_2 \text{ Schritten.} \end{aligned} \quad (5.52)$$

Gegeben N_1 , M und \bar{M} , konstruieren wir die NPOTM N_2 wie folgt. N_2 verhält sich genau wie N_1 , abgesehen davon, dass sie ihre Orakelfragen anders behandelt. Immer wenn N_1 ihr Orakel nach einem Wort fragt, etwa nach q , dann rät N_2 stattdessen nichtdeterministisch, ob die Antwort „ja“ oder „nein“ ist und verifiziert die geratene Antwort, indem sie $M^B(q)$ simuliert, falls „ja“ geraten wurde, und indem sie $\bar{M}^B(q)$ simuliert, falls „nein“ geraten wurde. In beiden Fällen wird die korrekt geratene Antwort durch einen akzeptierenden Berechnungspfad bezeugt, und eine inkorrekt geratene Antwort erzeugt ausschließlich ablehnende Pfade. Auf den akzeptierenden Pfaden, die die korrekte Orakelantwort bezeugen, setzt N_2 die Simulation von N_1 mit dieser korrekten Antwort fort, und sie lehnt auf allen anderen Pfaden ab.

Für geeignete Darstellungen von NPOTMs kann das Wort, das N_2 codiert, in Polynomialzeit aus dem Wort, das N_1 codiert, berechnet werden. Sei p ein monoton wachsendes Polynom, das die Rechenzeit von sowohl M als auch \bar{M} beschränkt. In höchstens t_1 Takten kann N_1 nur Fragen der Länge höchstens t_1 stellen. Folglich akzeptiert N_1^A eine Eingabe x_1 in höchstens t_1 Takten genau dann, wenn N_2^B diese Eingabe x_1 in höchstens $t_1 \cdot p(t_1)$ Takten akzeptiert. Setzen wir $x_2 = x_1$ und $t_2 = t_1 \cdot p(t_1)$, so ist die Konstruktion von f vollständig beschrieben und beweist (5.52). Somit ist $K(A) \leq_m^p K(B)$ mittels f . \square

Eine unmittelbare Konsequenz der ersten Aussage von Lemma 5.80 wird im folgenden Korollar angegeben, das leicht mit Induktion bewiesen werden kann; siehe Übung 5.33(b).

Korollar 5.81. 1. Für jede Menge A und alle $n \geq 1$ ist die Menge $K^n(A) \leq_m^p$ vollständig für $\Sigma_n^{p,A}$. Falls insbesondere $A = \emptyset$ gilt, ist K^n eine \leq_m^p -vollständige Menge für Σ_n^p .
2. Für alle $k, n \in \mathbb{N}$ gilt: Ist $A \leq_m^p$ -vollständig für Σ_k^p , so ist $K^n(A)$ eine \leq_m^p -vollständige Menge für Σ_{k+n}^p .

Für je zwei Mengen A und B folgt aus $A \leq_T^p B$ die flexiblere Reduktion $A \leq_{sT}^{NP} B$. Deshalb hat die dritte Aussage von Lemma 5.80 das folgende Korollar. Schöning [Sch83] zeigte, dass die Umkehrungen der Implikationen aus Korollar 5.82 nicht gelten.

Korollar 5.82. 1. Wenn $A \leq_T^p B$, dann $K(A) \leq_m^p K(B)$.
2. Wenn $A \leq_m^p B$, dann $K(A) \leq_m^p K(B)$.

Als nächstes wird gezeigt, dass die Definition der Lowness und Highness gemäß Definition 5.73 äquivalent zu Schönings [Sch83] ursprünglicher Definition gemäß Korollar 5.84 unten ist. Es ist interessant, zu bemerken, dass Satz 5.83 und Korollar 5.84 im Fall $n = 0$ nicht anwendbar sind. Für Low_0 ist der Grund dafür, dass $K^0(L)$

bzw. K^0 als L bzw. als die leere Menge definiert sind, woraus ein kleines technisches Problem bei der \leq_m^p -Reduktion von $K^0(L) = L$ auf $K^0 = \emptyset$ resultiert.

Definiert man K^0 jedoch (abweichend von unserer Definition) als eine feste nicht-triviale Menge B in P (d.h., $B \neq \emptyset$ und $B \neq \Sigma^*$), so gelten (5.53) in Satz 5.83 und die Charakterisierung von Low_n in Korollar 5.84 auch für den Fall $n = 0$: Eine NP-Menge L ist genau dann in Low_0 , wenn $K^0(L) = L \leq_m^p$ -reduzierbar auf $B = K^0$ ist, was wiederum äquivalent zur \leq_m^p -Vollständigkeit von $K^0(L)$ für P ist.

Im Gegensatz dazu ist für High_0 nicht bekannt, ob (5.54) auch für $n = 0$ gilt. Nach Satz 5.76, ist High_0 gleich der Klasse der Mengen, die \leq_T^p -vollständig für NP sind, wohingegen die Charakterisierung von Satz 5.83 für $n = 0$ die Gleichheit von High_0 mit der Klasse der für NP \leq_m^p -vollständigen Mengen erfordern würde.

Satz 5.83. Für jedes $n > 0$ und für alle Mengen L und H in NP gilt:

$$L \in \text{Low}_n \iff K^n(L) \leq_m^p K^n; \quad (5.53)$$

$$H \in \text{High}_n \iff K^{n+1} \leq_m^p K^n(H). \quad (5.54)$$

Beweis. Um (5.53) zu beweisen, sei L eine beliebige Menge in Low_n , $n > 0$. Somit gilt $\Sigma_n^{p,L} \subseteq \Sigma_n^p$. Da $K^n(L)$ in $\Sigma_n^{p,L} \subseteq \Sigma_n^p$ ist und da K^n nach Korollar 5.81 eine \leq_m^p -vollständige Menge für Σ_n^p ist, folgt $K^n(L) \leq_m^p K^n$.

Umgekehrt sei $K^n(L) \leq_m^p K^n$ angenommen. Nach Korollar 5.81 ist $K^n(L)$ eine \leq_m^p -vollständige Menge für $\Sigma_n^{p,L}$. Also kann jede Menge aus $\Sigma_n^{p,L}$ auf $K^n \leq_m^p$ -reduziert werden. Da $K^n \in \Sigma_n^p$ und da Σ_n^p unter \leq_m^p -Reduktionen abgeschlossen ist, folgt $\Sigma_n^{p,L} \subseteq \Sigma_n^p$. Somit ist L in Low_n .

Die Äquivalenz (5.54) kann analog bewiesen werden; siehe Übung 5.34. \square

Korollar 5.84. Für jedes $n > 0$ gilt:

$$\text{Low}_n = \{L \in \text{NP} \mid K^n(L) \text{ ist } \leq_m^p\text{-vollständig für } \Sigma_n^p\};$$

$$\text{High}_n = \{H \in \text{NP} \mid K^n(H) \text{ ist } \leq_m^p\text{-vollständig für } \Sigma_{n+1}^p\}.$$

Nun wenden wir uns den Fragen zu, die nach Korollar 5.78 aufgeworfen wurden: Gibt es Mengen in NP, die weder in Low_k noch in High_k für ein k sind, und gibt es Mengen in NP, die weder in LH noch in HH sind? Die Antwort auf diese Fragen erfordert zwei etwas technische Voraussetzungen, die die Klassen Low_n , High_n , LH und HH erfüllen müssen: rekursive Präsentierbarkeit und Abschluss unter endlicher Variation.

Definition 5.85 (Rekursive Präsentierbarkeit und endlicher Variationsabschluss). Sei \mathcal{C} eine Klasse entscheidbarer Mengen.

1. \mathcal{C} heißt rekursiv präsentierbar, falls es eine rekursiv aufzählbare Folge M_0, M_1, M_2, \dots von Turingmaschinen gibt, so dass gilt:
 - a) $\mathcal{C} = \{L(M_i) \mid i \geq 0\}$ und
 - b) jede der Turingmaschinen M_i hält bei jeder Eingabe.

2. \mathcal{C} ist abgeschlossen unter endlicher Variation, falls für je zwei Mengen A und B gilt: Ist $A \in \mathcal{C}$ und ist $A \Delta B = (A - B) \cup (B - A)$, die symmetrische Differenz von A und B , eine endliche Menge, so ist $B \in \mathcal{C}$.

Lemma 5.86. Für jedes $n \geq 0$ sind die Klassen Low_n , High_n , LH und HH rekursiv präsentierbar und abgeschlossen unter endlicher Variation.

Beweis. Der Beweis, dass diese Klassen unter endlicher Variation abgeschlossen sind, wird dem Leser als Übung 5.35(a) überlassen. Um zu zeigen, dass Low_n rekursiv präsentierbar ist, sei N_0, N_1, N_2, \dots eine fest gewählte Gödelisierung (d.h., eine effektive Nummerierung) aller NPTMs. Sei weiter T_0, T_1, T_2, \dots eine fest gewählte Gödelisierung aller deterministischen, polynomialzeit-beschränkten Turing-Transducer.¹⁰ Das heißt:

$$\text{NP} = \{L(N_i) \mid i \geq 0\};$$

$$\text{FP} = \{f_i \mid i \geq 0 \text{ und } f_i \text{ ist die von } T_i \text{ berechnete Funktion}\}.$$

Da jede Maschine N_i polynomialzeit-beschränkt ist und daher bei jeder Eingabe hält, ist die Menge $\{\langle x, i, n \rangle \mid x \in K^n(L(N_i))\}$ entscheidbar. Konstruiere eine rekursiv aufzählbare Folge M_0, M_1, M_2, \dots von Turingmaschinen, so dass $\text{Low}_n = \{L(M_i) \mid i \geq 0\}$ gilt, wie folgt. Nach (5.53) in Satz 5.83 ist eine Menge L genau dann in Low_n , wenn $K^n(L) \leq_m^p K^n$ gilt. Die Σ_n^p -vollständige Menge K^n in dieser Reduktion kann durch $K^n(A)$ für eine beliebige, fest gewählte nichttriviale Menge A in P ersetzt werden, d.h., $A \neq \emptyset$ und $A \neq \Sigma^*$. Dies folgt aus Lemma 3.37, das sagt, dass jede nichttriviale Menge in $P \leq_m^p$ -vollständig für P ist, sowie aus der zweiten Aussage von Korollar 5.81. Somit gilt:

$$L \in \text{Low}_n \iff K^n(L) \leq_m^p K^n(A). \quad (5.55)$$

Definiere für $i = \langle j, k \rangle$ die i -te Turingmaschine M_i wie folgt:

Schritt 1: Bei Eingabe x überprüft M_i für jedes Wort y mit $|y| < |x|$, ob

$$y \in K^n(L(N_j)) \iff T_k(y) \in K^n(A). \quad (5.56)$$

Schritt 2: Falls (5.56) für alle solchen Wörter y erfüllt ist, dann akzeptiert M_i die Eingabe x genau dann, wenn $x \in L(N_j)$. Andernfalls akzeptiert M_i die Eingabe x genau dann, wenn i gerade ist.

Offenbar hält M_i bei jeder Eingabe. Zu zeigen bleibt: $\text{Low}_n = \{L(M_i) \mid i \geq 0\}$.

Sei $L = L(M_i)$ für ein $i \geq 0$ mit $i = \langle j, k \rangle$. Nach Konstruktion von M_i gilt entweder (a) L ist eine endliche oder co-endliche Menge (wobei co-endlich heißt, dass ihr Komplement endlich ist), oder (b) $L = L(M_i) = L(N_j)$. Im Fall (a) ist L trivialerweise in Low_n , denn bereits $\text{Low}_0 = P$ enthält alle endlichen und co-endlichen Mengen. Im Fall (b) bezeugt T_k die Reduktion $K^n(L) \leq_m^p K^n(A)$. Nach (5.55) ist L in Low_n .

¹⁰ Ein Transducer ist eine Turingmaschine, die eine Funktion berechnet.

Umgekehrt sei angenommen, dass L in Low_n ist. Dann ist L insbesondere in NP. Sei N_j eine NPTM, die L akzeptiert, d.h., $L(N_j) = L$. Außerdem gibt es nach (5.55) ein k mit $K^n(L) \leq_m^p K^n(A)$ mittels T_k . Es folgt $L = L(M_{\langle j,k \rangle})$. Somit ist Low_n rekursiv präsentierbar.

Der Beweis, dass LH rekursiv präsentierbar ist, ist analog, nur dass nun Tripel $i = \langle j, k, n \rangle$ anstatt Paare $i = \langle j, k \rangle$ in der Konstruktion von M_i vorkommen.

Der Beweis, dass High_n und HH rekursiv präsentierbar sind, ist ähnlich zum obigen Beweis und wird dem Leser als Übung 5.35(b) überlassen. \square

Lemma 5.86 zeigt, dass die Klassen der Low- und der High-Hierarchie die Voraussetzungen von Lemma 5.87 erfüllen, das auf Schöning [Sch82] zurückgeht und als „Uniform Diagonalization Theorem“ bekannt ist. Dieses Lemma, auf dessen Beweis wir hier verzichten, wird im Beweis von Satz 5.88 angewandt.

Lemma 5.87. *Seien A und B entscheidbare Mengen, und seien \mathcal{C} und \mathcal{D} Klassen entscheidbarer Mengen mit den folgenden Eigenschaften:*

- (a) $A \notin \mathcal{C}$ und $B \notin \mathcal{D}$,
- (b) \mathcal{C} und \mathcal{D} sind rekursiv präsentierbar und
- (c) \mathcal{C} und \mathcal{D} sind abgeschlossen unter endlicher Variation.

Dann gibt es eine entscheidbare Menge X , so dass gilt:

1. $X \notin \mathcal{C}$ und $X \notin \mathcal{D}$.
2. Wenn $A \in P$ und B nichttrivial ist, dann gilt $X \leq_m^p B$.

Mit den oben angegebenen Lemmata können die Fragen, ob es Mengen in NP gibt, die weder low noch high sind, nun durch Kollapse und Separationen der Stufen der Polynomialzeit-Hierarchie charakterisiert werden.

Satz 5.88 (Schöning).

1. Für jedes $n \geq 0$ gibt es Mengen in NP, die weder in Low_n noch in High_n sind, genau dann, wenn $\Sigma_n^p \neq \Sigma_{n+1}^p$.
2. Es gibt Mengen in NP, die weder in LH noch in HH sind, genau dann, wenn die Polynomialzeit-Hierarchie echt unendlich ist.

Beweis. 1. Fixiere ein beliebiges $n \geq 0$. Für die Richtung von rechts nach links sei $\Sigma_n^p \neq \Sigma_{n+1}^p$ angenommen. Um Lemma 5.87 anzuwenden, wähle $A = \emptyset$, $B = \text{SAT}$, $\mathcal{C} = \text{High}_n$ und $\mathcal{D} = \text{Low}_n$.

Offenbar ist $A = \emptyset$ in $P = \text{Low}_0 \subseteq \text{Low}_n$ und die NP-vollständige Menge $B = \text{SAT}$ ist gemäß den Sätzen 5.76 und 5.77 in $\text{High}_0 \subseteq \text{High}_n$. Nach der dritten Aussage von Satz 5.77 impliziert dann unsere Annahme $\Sigma_n^p \neq \Sigma_{n+1}^p$, dass $A = \emptyset$ nicht in $\mathcal{C} = \text{High}_n$ und $B = \text{SAT}$ nicht in $\mathcal{D} = \text{Low}_n$ ist, womit die Voraussetzung (a) von Lemma 5.87 erfüllt ist. Die Voraussetzungen (b) und (c) von Lemma 5.87 sind nach Lemma 5.86 für $\mathcal{C} = \text{High}_n$ und $\mathcal{D} = \text{Low}_n$ erfüllt.

Wenden wir nun Lemma 5.87 an, so gibt es eine entscheidbare Menge X , so dass X weder in $\mathcal{C} = \text{High}_n$ noch in $\mathcal{D} = \text{Low}_n$ liegt. Weiterhin gilt $X \leq_m^p \text{SAT}$, da $A = \emptyset$ in P und $B = \text{SAT}$ natürlich nichttrivial ist. Somit ist X in NP.

Die Richtung von links nach rechts wird durch Kontraposition bewiesen. Die dritte Aussage von Satz 5.77 sagt, dass wenn $\Sigma_n^P = \Sigma_{n+1}^P$ gilt, so ist $\text{Low}_n \cap \text{High}_n \neq \emptyset$. Nach Korollar 5.78 ist $\text{NP} = \text{Low}_n = \text{High}_n$.

2. Diese Aussage folgt analog zur ersten Aussage des Satzes durch Anwendung von Korollar 5.78 und Lemma 5.87. \square

Man beachte, dass der Spezialfall $n = 0$ in der ersten Aussage von Satz 5.88 nichts anderes ist, als Ladners Resultat [Lad75], welches sagt, dass $\text{P} \neq \text{NP}$ genau dann gilt, wenn es eine Menge in NP gibt, die weder in P noch NP -vollständig ist; siehe Satz 3.68.

In Kapitel 6 werden konkrete Beispiele natürlicher Probleme und Klassen von Problemen in der Low-Hierarchie präsentiert, einschließlich des Graphisomorphie-Problems und bestimmter Klassen, die durch randomisierte Algorithmen definiert sind.

5.8 Übungen und Probleme

Aufgabe 5.1 Vervollständige den Beweis von Fakt 5.2: $\text{Exact-}i\text{-DNP}$ ist NP -hart für $i > 3$.

Aufgabe 5.2 Betrachte das Problem SAT-UNSAT , das im Beweis von Satz 5.57 definiert ist als

$$\text{SAT-UNSAT} = \left\{ \langle \varphi, \psi \rangle \mid \begin{array}{l} \varphi \text{ und } \psi \text{ sind boolesche Formeln in KNF,} \\ \text{so dass } \varphi \text{ erfüllbar und } \psi \text{ nicht erfüllbar ist} \end{array} \right\}.$$

Beweise, dass SAT-UNSAT DP -vollständig ist.

Hinweis: Jede Menge L in DP kann man als $L = A \cap \overline{B}$ für Mengen A und B in NP schreiben. Verwende die Reduktionen von A bzw. B auf die NP -vollständige Menge SAT .

Aufgabe 5.3 Beweise:

(a) $\text{NP} \cap \text{coNP} \subseteq \text{NP} \subseteq \text{NP} \cup \text{coNP} \subseteq (\text{NP} \wedge \text{coNP}) \cup (\text{NP} \vee \text{coNP})$.

(Kann irgendeine dieser Inklusionen zu einer Gleichheit verstärkt werden?)

Hinweis: Wenn ja, Glückwunsch! Soeben ist sowohl die boolesche als auch die Polynomialzeit-Hierarchie kollabiert. Dieses Ergebnis sollte in einer hochkarätigen Informatik-Zeitschrift publiziert werden ... nach einer sorgfältigen Überprüfung des Beweises.)

(b) $\text{NP} \wedge \text{coNP} = \text{co}(\text{NP} \vee \text{coNP})$.

(c) Ist $\text{P} = \text{NP}$, so ist $\text{P} = \text{coNP} = \text{NP} \cap \text{coNP} = \text{NP} \wedge \text{coNP} = \text{NP} \vee \text{coNP}$.

Aufgabe 5.4 Beweise:

(a) NP ist ein Mengenring, d.h., NP ist abgeschlossen unter Vereinigung und Durchschnitt.

- (b) P und PSPACE sind boolesche Algebren, also abgeschlossen unter Vereinigung, Durchschnitt und Komplement. Was ist $\text{BC}(P)$, und was ist $\text{BC}(\text{PSPACE})$?

- Aufgabe 5.5** (a) Beweise, dass $\text{NP} = \text{coNP}$ genau dann gilt, wenn $\text{NP} = \text{BC}(\text{NP})$.
 (b) Beweise, dass $P = \text{NP}$ genau dann gilt, wenn $P = \text{BC}(\text{NP})$.
 (c) Warum folgt $P \neq \text{NP}$ aus $\text{NP} \neq \text{BC}(\text{NP})$?

- Aufgabe 5.6** (a) Beweise, dass UP unter Durchschnitt abgeschlossen ist.
 (b) Ist UP unter Vereinigung abgeschlossen? Ist es abgeschlossen unter Komplement?
 (c) Von welchen Komplexitätsklassen weiß man, dass sie unter Durchschnitt, aber nicht, ob sie unter Vereinigung abgeschlossen sind?

- Aufgabe 5.7** Beweise Satz 5.6: Für jeden Mengenring \mathcal{C} gelten die folgenden beiden Aussagen:

1. Die verschachtelte Differenzenhierarchie über \mathcal{C} stimmt Stufe für Stufe mit der Vereinigung-von-Differenzen-Hierarchie über \mathcal{C} überein:

$$\text{BH}_k(\mathcal{C}) = \left\{ L \left| \begin{array}{l} L = A_1 - A_2 - \cdots - A_k \text{ für Mengen } A_i \text{ in } \mathcal{C}, \\ 1 \leq i \leq k, \text{ wobei } A_k \subseteq A_{k-1} \subseteq \cdots \subseteq A_1 \end{array} \right. \right\}.$$

2. $\text{BH}(\mathcal{C}) = \text{BC}(\mathcal{C})$.

- Aufgabe 5.8** Korollar 5.13 sagt, dass Exact-5-DNP DP-vollständig ist. Finde Indizien dafür, dass im Gegensatz dazu Exact-2-DNP vermutlich nicht DP-vollständig ist. Insbesondere:

- (a) Beweise, dass Exact-2-DNP in coNP liegt.

Hinweis: Zeige, dass jeder Graph ohne isolierte Knoten eine dominante Zahl von mindestens 2 hat. Bestimme die Komplexität der Menge $\{G \mid \delta(G) \geq 2\}$.

- (b) Beweise, dass Exact-2-DNP sogar coNP-vollständig ist.
 (c) Angenommen, Exact-2-DNP wäre DP-vollständig. Welche Konsequenzen würden aus dieser Annahme für die boolesche Hierarchie und welche würden für die Polynomialzeit-Hierarchie folgen?

- Aufgabe 5.9** Gibt es \leq_m^p -vollständige Mengen für die Klasse $\text{BH}(\text{NP})$?

- Aufgabe 5.10** Argumentiere, weshalb die Annahmen im Beweis von Satz 5.12 ohne Beschränkung der Allgemeinheit gemacht werden können:

- (a) Für die chromatische Zahl $\chi(G_j)$ eines jeden Graphen G_j gilt $3 \leq \chi(G_j) \leq 4$.
 (b) Keiner der Graphen G_j enthält isolierte Knoten.

- Aufgabe 5.11** Wende (ohne Benutzung von Lemma 5.18 oder von Satz 5.20) die Lemmata 5.11 und 5.17 direkt an, um die folgenden Aussagen zu beweisen:

- (a) Exact-7-Colorability ist DP-vollständig.
 (b) Für die k -elementige Menge $M_k = \{6k+1, 6k+3, \dots, 8k-1\}$ ist das Problem Exact- M_k -Colorability $\text{BH}_{2k}(\text{NP})$ -vollständig.

Aufgabe 5.12 Beweise Behauptung 5.16: Sei $k \geq 1$ fest gewählt, und sei M_k eine beliebige Menge von k positiven ganzen Zahlen inklusive 3. Dann ist das Problem Exact- M_k -Colorability in $\text{BH}_{2k-1}(\text{NP})$ und somit nicht $\text{BH}_{2k}(\text{NP})$ -vollständig, außer wenn die boolesche Hierarchie über NP kollabiert.

Aufgabe 5.13 Beweise, dass die Operation \bowtie auf Graphen, die im Beweis von Satz 5.20 definiert wird, assoziativ ist. Beweise außerdem, dass für je zwei Graphen A und B die Gleichheit $\chi(A \bowtie B) = \chi(A) + \chi(B)$ gilt, wobei $\chi(G)$ die chromatische Zahl des Graphen G bezeichnet.

Aufgabe 5.14 (a) In Abschnitt 5.1 wurden das exakte Domatische-Zahl-Problem und das exakte Färbarkeitsproblem betrachtet. Definiere analog dazu weitere solche exakten Varianten NP-vollständiger Optimierungsprobleme. Gegeben seien also eine Zahl $i \geq 1$ und eine k -elementige Menge M_k von positiven ganzen Zahlen. Definiere die exakten Versionen des Clique-Problems, des Unabhängige-Menge-Problems und eines dritten NP-vollständigen Problems deiner Wahl:

- Exact- i -Clique und Exact- M_k -Clique;
 - Exact- i -IS und Exact- M_k -IS;
 - Exact- i -Favorite und Exact- M_k -Favorite, wobei Favorite dein Lieblingsproblem unter allen NP-vollständigen Problemen ist, in deren Definition ein Parameter (wie das k in IS gemäß Definition 3.52) eingeht.
- (b) Beweise, dass die in (a) definierten Probleme vollständig für die Stufen der booleschen Hierarchie über NP sind. Was kann über die Optimalität der Parameter i und der Zahlen in M_k gesagt werden?
- (c) Beweise, dass das Problem Clique-Facet, das in Abschnitt 5.9 definiert wird, DP-vollständig ist.

Hinweis: Siehe nicht in der Arbeit von Papadimitriou und Yannakakis [PY84] nach, bevor du nicht selbst versucht hast, eine Reduktion von Exact- i -Clique auf Clique-Facet zu finden.

Aufgabe 5.15 (a) Zeige, dass Unique-SAT coNP-hart und in DP enthalten ist.

(b) Beweise, dass $\text{NP} = \text{coNP}$ gilt, falls Unique-SAT in NP ist.

Aufgabe 5.16 (a) Beweise Satz 5.24: Eine Menge A ist genau dann in NP, wenn es eine Menge B in P und ein Polynom p gibt, so dass für alle $x \in \Sigma^*$ gilt:

$$x \in A \iff (\exists w)[|w| \leq p(|x|) \text{ und } \langle x, w \rangle \in B].$$

(b) Beweise die analoge Quantorencharakterisierung für coNP.

Hinweis: Verwende dabei weder Satz 5.31 noch Korollar 5.32.

Aufgabe 5.17 (a) Beweise, dass die in Beispiel 5.26 definierte Menge Pre-Iso in NP liegt und dass die Maschine N in Abbildung 5.9 tatsächlich eine DPOTM ist; d.h., zeige, dass N in Polynomialzeit arbeitet.

- (b) Sei M eine fest gewählte NP-Maschine für das Problem S0S. Definiere die Funktion

$$g(x) = \begin{cases} \min\{w \mid w \in \text{Wit}_M(x)\} & \text{falls } x \in \text{S0S} \\ \varepsilon & \text{sonst,} \end{cases}$$

wobei das Minimum bezüglich der lexikographischen Ordnung von Wörtern genommen wird. Ohne Beschränkung der Allgemeinheit codiere dabei ε , das leere Wort, keinen akzeptierenden Pfad von $M(x)$. Beweise, dass g in FP^{NP} ist.

Hinweis: Implementiere eine Präfixsuche ähnlich der aus Abbildung 5.9 in Beispiel 5.26 (Pseudocode genügt).

Aufgabe 5.18 Beweise Behauptung 5.28:

1. $\leq_m^{\log} \subseteq \leq_m^p \subseteq \leq_T^p \subseteq \leq_T^{\text{NP}}$.
2. Die Relation \leq_T^p ist reflexiv und transitiv, aber nicht antisymmetrisch.
3. Die Relation \leq_T^{NP} ist reflexiv, aber weder antisymmetrisch noch transitiv.
4. P und PSPACE sind \leq_T^p -abgeschlossen, d.h., $P^P = P$ und $P^{\text{PSPACE}} = \text{PSPACE}$.
5. $\text{NP}^P = \text{NP}$ und $\text{NP}^{\text{PSPACE}} = \text{PSPACE}$.
6. Gilt $A \leq_T^p B$ und ist $A \leq_T^p$ -hart für eine Komplexitätsklasse \mathcal{C} , so ist auch $B \leq_T^p$ -hart für \mathcal{C} .
7. Ist $L \neq \text{NP}$, so gibt es Mengen A und B in NP , so dass $A \leq_T^{\text{NP}} B$, aber $A \not\leq_m^{\log} B$.

Aufgabe 5.19 Eine DPOTM M heißt *positiv*, falls $L(M^A) \subseteq L(M^B)$ aus $A \subseteq B$ folgt. Definiere die *polynomialzeit-beschränkte positive Turing-Reduzierbarkeit*, bezeichnet mit $\leq_{\text{pos-T}}^p$, wie folgt: $A \leq_{\text{pos-T}}^p B$ genau dann, wenn es eine positive DPOTM M mit $A = L(M^B)$ gibt.

- (a) Zeige, dass NP unter $\leq_{\text{pos-T}}^p$ -Reduktionen abgeschlossen ist.
- (b) Welche anderen Komplexitätsklassen sind $\leq_{\text{pos-T}}^p$ -abgeschlossen?

Aufgabe 5.20 Beweise die dritte Aussage von Satz 5.30:

- (a) Jede der Klassen Δ_i^p , Σ_i^p , Π_i^p und PH ist \leq_m^p -abgeschlossen.
- (b) Die Δ_i^p -Stufen der Polynomialzeit-Hierarchie sind sogar unter \leq_T^p -Reduktionen abgeschlossen.

Aufgabe 5.21 Betrachte die im Beweis von Satz 5.31 definierten Mengen C_{ja} und C_{nein} . Es wird in diesem Beweis behauptet, dass C_{ja} in Σ_i^p und C_{nein} in Π_i^p ist. Beweise diese Behauptungen.

Aufgabe 5.22 Zeige, dass für jede Klasse \mathcal{C} aus der Inklusion $\text{co}\mathcal{C} \subseteq \mathcal{C}$ die Gleichheit $\mathcal{C} = \text{co}\mathcal{C}$ folgt.

Aufgabe 5.23 Im Beweis von Satz 5.33 wird die Äquivalenz $\Sigma_i^p = \Pi_i^p \iff \Sigma_i^p = \Sigma_{i+1}^p$ für jedes $i \geq 1$ gezeigt, jedoch nur die Implikation von rechts nach links für $i = 0$. An welcher Stelle versagt das Argument dieses Beweises bei der umgekehrten Implikation: $\Sigma_0^p = \Pi_0^p \implies \Sigma_0^p = \Sigma_1^p$?

Aufgabe 5.24 (a) Beweise $\Sigma_1 \text{SAT} = \text{SAT}$. Vergleiche $\Pi_1 \text{SAT}$ mit dem Tautologieproblem, das in Übung 5.27 unten definiert wird. Was kann man über die Komplexität dieser beiden Probleme sagen?

(b) Beweise die erste Behauptung von Satz 5.36: QBF ist PSPACE-vollständig.

Hinweis: Verwende die im Beweis des Satzes von Savitch eingeführte Methode; siehe Satz 3.29. Alternativ dazu kann man auch die im Beweis der zweiten Aussage von Satz 5.36 beschriebene Methode anwenden.

(c) Eine geschlossene QBF F heißt *einfach*, falls es für jedes Vorkommen einer Variablen x_i in F höchstens einen \forall -Quantor zwischen x_i und der Stelle der Quantifizierung von x_i gibt. Einfache QBFs sind wichtig im Beweis von A. Shamirs Resultat $\text{PSPACE} = \text{IP}$ (siehe [Sha92]), das in den Abschnitten 6.7 und 8.8 erwähnt wird.

Beweise, dass die folgende Einschränkung von QBF PSPACE-vollständig ist:

$$\text{QBF}_{\text{simple}} = \{F \mid F \text{ ist eine wahre einfache QBF}\}.$$

(d) Beweise Korollar 5.37: $\text{PSPACE} = \Sigma_i^P$ genau dann, wenn $\text{QBF} \in \Sigma_i^P$.

Aufgabe 5.25 (a) Beweise Behauptung 5.40: IN-Odd, IN-Equ und IN-Geq sind in Θ_2^P .

(b) Vervollständige den Beweis von Satz 5.44: Zeige, dass es eine Reduktion g für $3\text{-SAT} \leq_m^P \text{IS}$ gibt, die (5.25) und (5.26) erfüllt. Zeige außerdem, dass IN-Odd und IN-Geq Θ_2^P -hart sind.

Hinweis: Dass die Probleme IN-Odd und IN-Geq Θ_2^P -vollständig sind, wird in [Wag87a, Cor. 6.4] bzw. in [SV00, Thm. 12] explizit bewiesen.

(c) Beweise Lemma 5.48: Max-SetPacking-Geq ist Θ_2^P -vollständig.

Hinweis: Konstruiere eine Reduktion von IN-Geq: Gegeben zwei Graphen G_1 und G_2 , definiere $U_i = E(G_i)$ für $i \in \{1, 2\}$, und füge für jeden Knoten v von G_i zu \mathcal{S}_i die Menge der mit v inzidenten Kanten hinzu. Argumentiere, dass $\alpha(G_i) = \kappa(\mathcal{S}_i)$ gilt. Siehe [RSV03] für weitere Details.

Aufgabe 5.26 Beweise, dass YoungRanking und YoungWinner in Θ_2^P liegen.

Aufgabe 5.27 (a) Beweise, dass das Problem MEE-DNF, das in Abschnitt 5.9 definiert wird, in Σ_2^P liegt.

(b) Beweise, dass MEE-DNF coNP-hart ist.

Hinweis: Finde eine Reduktion vom coNP-vollständigen Tautologieproblem: Gegeben eine boolesche Formel φ , ist φ eine Tautologie? Das heißt, ist φ unter jeder möglichen Belegung mit Wahrheitswerten wahr?

Aufgabe 5.28 Beweise Behauptung 5.60.1: $E = \{\psi \in \{0, 1\}^* \mid \psi \text{ ist leicht}\}$ ist eine Menge in NP. **Hinweis:** Wende Satz 5.24 an.

Aufgabe 5.29 (a) Beweise, dass das im Beweis von Satz 5.53 definierte Problem Odd- k -SAT $\text{BH}_k(\text{NP})$ -vollständig ist. **Hinweis:** Wende Lemma 5.11 an.

(b) Definiere das Problem

$$\text{Odd-SAT} = \left\{ \langle \varphi_1, \varphi_2, \dots, \varphi_k \rangle \mid k \in \mathbb{N} \text{ und } \varphi_1, \varphi_2, \dots, \varphi_k \text{ sind boolesche Formeln und } \|\{i \mid \varphi_i \in \text{SAT}\}\| \text{ ist ungerade} \right\}.$$

Für welche Komplexitätsklasse ist Odd-SAT vollständig? Beweise deine Vermutung.

Aufgabe 5.30 Gib einen detaillierten Beweis von Satz 5.66: Für jede zeitkonstruierbare Funktion $s \geq \log$ ist $\text{DSPACE}(s) \subseteq \text{ATIME}(s^2)$.

Aufgabe 5.31 Gib einen detaillierten Beweis von Satz 5.68: Ist $s \geq \log$ konstruierbar in der Zeit $2^{\text{Lin}(s)}$, so ist $\text{ASPACE}(s) \subseteq \text{DTIME}(2^{\text{Lin}(s)})$.

Aufgabe 5.32 (a) Beweise Lemma 5.75, das sagt, dass die folgenden drei Aussagen paarweise äquivalent sind:

- $A \leq_{\text{ST}}^{\text{NP}} B$.
- $A \in \text{NP}^B \cap \text{coNP}^B$.
- $\text{NP}^A \subseteq \text{NP}^B$.

(b) Beweise die analoge Behauptung für die $\leq_{\text{T}}^{\text{P}}$ -Reduzierbarkeit, d.h., zeige, dass die folgenden drei Aussagen paarweise äquivalent sind:

- $A \leq_{\text{T}}^{\text{P}} B$.
- $A \in \text{P}^B$.
- $\text{P}^A \subseteq \text{P}^B$.

Aufgabe 5.33 (a) Beweise Lemma 5.80.1: Für jede Menge A ist die Menge $K(A) \leq_{\text{m}}^{\text{P}}$ -vollständig für NP^A . Falls insbesondere $A = \emptyset$ gilt, ist K eine $\leq_{\text{m}}^{\text{P}}$ -vollständige Menge für NP .

(b) Beweise Korollar 5.81:

- a) Für jede Menge A und alle $n \geq 1$ ist die Menge $K^n(A) \leq_{\text{m}}^{\text{P}}$ -vollständig für $\Sigma_n^{P,A}$. Falls insbesondere $A = \emptyset$ gilt, ist K^n eine $\leq_{\text{m}}^{\text{P}}$ -vollständige Menge für Σ_n^P .
- b) Für alle $k, n \in \mathbb{N}$ gilt: Ist $A \leq_{\text{m}}^{\text{P}}$ -vollständig für Σ_k^P , so ist $K^n(A)$ eine $\leq_{\text{m}}^{\text{P}}$ -vollständige Menge für Σ_{k+n}^P .

Aufgabe 5.34 Beweise (5.54) aus Satz 5.83: Für jedes $n > 0$ ist H genau dann in High_n , wenn $K^{n+1} \leq_{\text{m}}^{\text{P}} K^n(H)$ gilt.

Aufgabe 5.35 Vervollständige den Beweis von Lemma 5.86:

- (a) Für jedes $n \geq 0$ sind die Klassen Low_n , High_n , LH und HH abgeschlossen unter endlicher Variation.
- (b) Beweise, dass High_n und HH rekursiv präsentierbar sind.

Aufgabe 5.36 Haben die Stufen Low_k der Low-Hierarchie vollständige Probleme?



Problem 5.1 (Kritische Graphenprobleme) Ein *Hamiltonkreis* in einem ungerichteten Graphen G mit n Knoten ist eine Folge (v_1, v_2, \dots, v_n) von Knoten aus $V(G)$, so dass $\{v_n, v_1\} \in E(G)$ und $\{v_i, v_{i+1}\} \in E(G)$ für alle i , $1 \leq i < n$. Hamiltonkreise in gerichteten Graphen sind analog definiert. Das *Hamiltonkreis-Problem* fragt, ob ein gegebener Graph einen Hamiltonkreis hat. Dieses Problem ist sowohl für gerichtete als auch für ungerichtete Graphen NP-vollständig.

- (a) Beweise die obigen Behauptungen.

Hinweis: Reduziere zunächst das Knotenüberdeckungsproblem auf das Hamiltonkreis-Problem für ungerichtete Graphen. Reduziere das letztere Problem dann auf das Hamiltonkreis-Problem für gerichtete Graphen. Beide Reduktionen findet man in [GJ79].

- (b) Wir definieren nun eine „kritische“ Variante des Hamiltonkreis-Problems (bezeichnet mit MNHC für „maximal non-hamiltonian circuit“) wie folgt: Gegeben ein ungerichteter Graph G , ist es wahr, dass G keinen Hamiltonkreis hat, aber das Hinzufügen einer neuen Kante zu G erzeugt einen Graphen mit einem Hamiltonkreis? Das entsprechende Problem für gerichtete Graphen bezeichnen wir mit MDNHC. Diese Probleme sind kritisch in dem Sinn, dass zu testen ist, ob ein gegebener Graph eine bestimmte Eigenschaft nicht hat (nämlich die, einen Hamiltonkreis zu besitzen), aber bereits eine minimale Änderung des Graphen bringt diese Eigenschaft hervor. Weitere kritische Probleme werden in Abschnitt 5.9 vorgestellt.

Beweise, dass MNHC und MDNHC in DP liegen.

- (c) Zeige $MNHC \leq_m^p MDNHC$.

Hinweis: Verwende die in (a) gesuchte Reduktion vom Hamiltonkreis-Problem für ungerichtete Graphen auf das für gerichtete Graphen. Diese Reduktion erhält zufälligerweise die Kritikalität. (Das ist jedoch ein ziemlicher Glücksfall. Im Allgemeinen kann man nicht selbstverständlich voraussetzen, dass Reduktionen die Kritikalität erhalten. Tatsächlich ist dies bei der überwiegenden Zahl von Reduktionen nicht der Fall. Das heißt, $MNHC \leq_m^p MDNHC$ ist einer der seltenen Fälle, in denen die übliche NP-Vollständigkeitsreduktion zufällig auch für die kritische Problemvariante funktioniert.)

- (d) Kann man zeigen, dass MNHC DP-vollständig ist? Was ist mit dem kritischen Problem TSP-Facet, dass in Abschnitt 5.9 erwähnt und in [PY84] definiert wird?

Hinweis: Diese Reduktionen sind ziemlich anspruchsvoll und raffiniert. Eine Reduktion von Minimal-3-UNSAT auf MNHC ist in [PW88] und eine Reduktion von MNHC auf TSP-Facet ist in [PY84] zu finden.

Problem 5.2 (Korrekttheit der Guruswami–Khanna–Reduktion)

Im Beweis von Lemma 5.18 wird die Guruswami–Khanna–Reduktion von 3-SAT auf 3-Colorability beschrieben. Intuitiv wird argumentiert, dass jede 4-Färbung des Graphen $H = \rho(G)$ die „Wurzel“ r_i einer jeden baumartigen Struktur S_i wählt und dass diese Auswahl nach unten zu den „Blättern“ von S_i vererbt wird.

Gib einen formalen Beweis dafür, dass diese Reduktion korrekt ist. Beweise insbesondere die folgenden Behauptungen:

- (a) Für jedes i mit $1 \leq i \leq m$ gibt es eine legale 3-Färbung der Knoten in S_i , so dass genau eines der drei „Blätter“ $t_{i,1}$, $t_{i,2}$ und $t_{i,3}$ gewählt wird.
- (b) Beweise Lemma 5.19: Jede legale 4-Färbung von S_i wählt wenigstens eines der „Blätter“ von S_i aus, also $t_{i,1}$, $t_{i,2}$ oder $t_{i,3}$.

Hinweis: Konstruiere ein Grundgitter, welches das aus Abbildung 5.5 leicht modifiziert, so dass das modifizierte Grundgitter die Auswahl der „Wurzel“ erzwingt. Unter Verwendung des in Abbildung 5.6 dargestellten Verbindungsmusters zwischen den Grundgittern einer baumartigen Struktur S_i ist dann zu zeigen, dass falls ein innerer „Knoten“ von S_i gewählt wird, dann wird auch eines seiner „Kinder“ gewählt.

- (c) Beweise, dass die in den Abbildungen 5.7 und 5.8 dargestellten Teilgraphen zur Verbindung der „Blätter“ in unterschiedlichen S_i und S_j , $i \neq j$, verhindern, dass beide „Blätter“ von einer legalen 4-Färbung gleichzeitig gewählt werden. Weshalb muss man sicherstellen, dass solche „Knoten“, etwa $t_{i,k}$ in S_i und $t_{j,\ell}$ in S_j , die benachbarten Knoten in G entsprechen, von einer legalen 4-Färbung nicht gleichzeitig gewählt werden können?

Problem 5.3 (Berechnen vollständiger Graph-Isomorphismen aus partiellen)

- (a) Sei g ein Funktionsorakel, das bei Eingabe zweier isomorpher Graphen mit n Knoten einen partiellen Isomorphismus zwischen den Graphen ausgibt, der aus mindestens $(3 + \varepsilon) \log n$ Knoten für eine geeignete Konstante $\varepsilon > 0$ besteht. Konstruiere eine DPOTM M , die mit dem Orakel g einen vollständigen Isomorphismus zwischen je zwei isomorphen Graphen ausgibt.

Hinweis: Dieses Problem und seine Lösung gehen auf Gál, Halevi, Lipton und Petrank [GHL99] zurück.

- (b) Das obige Resultat kann verbessert werden. Zeige dieselbe Konsequenz unter der folgenden Annahme: Sei f ein Funktionsorakel, das *lediglich ein Knotenpaar* ausgibt, welches zu einem Isomorphismus zwischen zwei gegebenen isomorphen Graphen gehört; das heißt, $f(G, H)$ berechnet einen partiellen Isomorphismus, der lediglich aus einem Paar (x, y) besteht. Dabei ist $x \in V(G)$ und $y \in V(H)$ mit $\sigma(x) = y$ für ein σ in $\text{Iso}(G, H)$. Konstruiere eine DPOTM M , die mit dem Orakel f einen vollständigen Isomorphismus φ in $\text{Iso}(G, H)$ berechnet, wobei φ und σ verschieden sein dürfen.

Hinweis: Diese Lösung geht auf Große, Rothe und Wechsung [GRW02] zurück, deren Resultat auf den Techniken beruht, mit denen man die Selbstreduzierbarkeit von GI zeigt; siehe auch die Kommentare in Abschnitt 3.8.

5.9 Zusammenfassung und bibliographische Notizen

Die Klasse DP: Papadimitriou und Yannakakis [PY84] führten in ihrer einflussreichen Arbeit DP ein, welches die zweite Stufe der booleschen Hierarchie über NP

bildet. Zusätzlich zu den exakten Varianten NP-vollständiger Optimierungsprobleme untersuchten sie auch mehrere andere wichtige Klassen von Problemen in DP, darunter *kritische Probleme*, *Facettenprobleme* und *Eindeutigkeitsprobleme*, und sie zeigten die DP-Vollständigkeit vieler dieser Probleme.

Kritische Probleme: Ein Graph heißt *kritisch*, falls er eine bestimmte Eigenschaft nicht hat, aber bereits eine minimale Änderung des Graphen, etwa das Löschen oder Hinzufügen eines einzigen Knotens oder einer einzigen Kante, bringt einen Graphen mit dieser Eigenschaft hervor. Kritische Graphenprobleme fragen, ob ein gegebener Graph kritisch hinsichtlich einer gegebenen Eigenschaft ist oder nicht. Zum Beispiel ist *Minimal-3-Uncolorability* ein kritisches Graphenproblem: Ist ein gegebener Graph nicht 3-färbbar, aber das Entfernen auch nur eines Knotens macht ihn 3-färbbar? Papadimitriou und Yannakakis [PY84] stellten fest, dass kritische Graphenprobleme dazu neigen, äußerst schwer fassbar zu sein, und dass gerade ihre Komplexität und die dafür nötige Reduktionsmethode ausgesprochen schwer in den Griff zu bekommen ist:

„This difficulty seems to reflect the extremely delicate and deep structure of critical problems—too delicate to sustain any of the known reduction methods. One way to understand this is that critical graphs is usually the object of hard theorems.“

Das erste kritische Problem, für das der Nachweis der DP-Vollständigkeit gelang, ist *Minimal-3-UNSAT*: Gegeben eine boolesche Formel φ in 3-KNF (und mit höchstens drei Vorkommen einer jeden Variablen), ist es wahr, dass φ unerfüllbar ist, aber das Entfernen auch nur einer Klausel von φ liefert eine erfüllbare Formel? Dass dieses Problem DP-vollständig ist, wurde von Papadimitriou und Wolfe [PW88] mittels einer Reduktion von SAT-UNSAT gezeigt, welches in Übung 5.2 definiert ist.

Mit einer sehr raffinierten Konstruktion bewiesen Cai und G. Meyer [CM87], dass das Problem *Minimal-3-Uncolorability* DP-vollständig ist, selbst wenn es auf planare Graphen eingeschränkt wird. Ihre Reduktion geht von dem Problem *Minimal-3-UNSAT* aus.

V. Vazirani, wie in [PW88] erwähnt, bewies die DP-Vollständigkeit des Problems *Critical-Clique*: Gegeben ein Graph G und eine ganze Zahl k , ist es wahr, dass G keine Clique der Größe k hat, aber das Hinzufügen auch nur einer Kante liefert einen Graphen mit einer k -Clique? Weitere kritische Graphenprobleme wurden in Problem 5.1 erwähnt.

Facettenprobleme: Viele kombinatorische Optimierungsprobleme, das Traveling-Salesperson-Problem¹¹ und das Clique-Problem (deren Entscheidungsversionen jeweils NP-vollständig sind) eingeschlossen, betreffen die Optimierung eines linearen

¹¹ Sei $C = \{c_1, c_2, \dots, c_n\}$ eine Menge von n Städten. Eine *Distanzfunktion auf C* ist eine $(n \times n)$ -Matrix D , mit den Einträgen $d(c_i, c_i) = 0$ auf der Diagonalen, $1 \leq i \leq n$, und $d(c_i, c_j) \in \mathbb{Z}^+$ für jedes Paar verschiedener Städte c_i und c_j in C , wobei \mathbb{Z}^+ die Menge der positiven ganzen Zahlen bezeichnet. Eine *Traveling-Salesperson-Tour auf C* ist eine Tour, die jede Stadt genau einmal besucht und dann zum Ausgangspunkt zurückkehrt. For-

Funktionalen über einem diskreten Vektorraum. Zum Beispiel betrifft das Clique-Problem die Maximierung von $\mathbf{c}'\mathbf{x}$ unter der Bedingung $\mathbf{x} \in C$, wobei \mathbf{x} ein n -dimensionaler Variablenvektor, \mathbf{c} der Vektor mit n Einsen und C die Menge der charakteristischen Vektoren der Cliques des gegebenen Graphen ist. Äquivalent dazu kann man das Clique-Problem als eine Maximierung von $\mathbf{c}'\mathbf{x}$ unter der Bedingung $\mathbf{x} \in \text{Polytope}(C)$ auffassen, wobei $\text{Polytope}(C)$ das *Cliquepolytop von C* ist, d.h. die konvexe Hülle von C . Die Optimierung eines linearen Funktionalen über einem konvexen Polytop mittels linearer Programmierung benötigt die Charakterisierung der *Facetten* dieses Polytops, d.h., sie benötigt das nicht-redundante System linearer Ungleichungen, die dieses Polytop definieren.

Definiere *Clique-Facet* als das folgende Problem: Gegeben ein Graph G und eine lineare Ungleichung $\sum_{i=1}^n x_i \leq k$, ist diese Ungleichung eine Facette des Cliquenpolytops von G ? Papadimitriou und Yannakakis [PY84] bewiesen, dass das Problem *Clique-Facet* DP-vollständig ist; siehe Übung 5.14(c). Ein alternativer Beweis, der eine Reduktion von dem Problem *Critical-Clique* auf *Clique-Facet* liefert, wird in [PW88] Lovász zugeschrieben. Die DP-Vollständigkeit von *TSP-Facet*, dem Traveling-Salesperson-Facet-Problem, wurde von Papadimitriou und Wolfe [PW88] bewiesen. Ihre Reduktion auf *TSP-Facet* geht vom Problem *MNHC* aus, einem kritischen Graphenproblem, das in Problem 5.1 definiert wurde. Die Übungen 5.2, 5.14(c) und 5.15 sowie Problem 5.1 stammen aus [PY84].

Eindeutigkeitsprobleme: Definiere *Unique-SAT* als die Menge der booleschen Formeln, die genau eine erfüllende Belegung haben. Es ist nicht schwer zu sehen, dass *Unique-SAT* coNP-hart und in DP enthalten ist; siehe Übung 5.15. Die genaue Komplexität von *Unique-SAT* ist jedoch unbekannt.

Die offensichtliche Frage, ob *Unique-SAT* DP-vollständig ist oder nicht, ist intensiv untersucht worden. Hier sind einige der partiellen Antworten, die dabei erhalten wurden: Blass und Gurevich [BG82] zeigten, dass die Frage, ob *Unique-SAT* DP-vollständig ist oder nicht, eng mit der Frage zusammenhängt, ob *Unique-SAT* NP-hart ist oder nicht:

$$\begin{aligned} \text{Unique-SAT ist DP-vollständig} &\iff \text{SAT-UNSAT} \leq_m^p \text{Unique-SAT} \\ &\iff \text{SAT} \leq_m^p \text{Unique-SAT}. \end{aligned}$$

Sie konstruierten ein Orakel, relativ zu dem keine \leq_m^p -Reduktion von *SAT* auf *Unique-SAT* existiert. Im Gegensatz dazu gaben Valiant und V. Vazirani in ihrer wegweisenden Arbeit [VV86] eine polynomialzeit-beschränkte *randomisierte* Reduktion von *SAT* auf *Unique-SAT* an; siehe Kapitel 6. Somit ist *Unique-SAT* DP-vollständig

mal ist sie durch eine Permutation π auf $\{1, 2, \dots, n\}$ gegeben, wobei die Städte $c_{\pi(1)}, c_{\pi(2)}, \dots, c_{\pi(n)}, c_{\pi(1)}$ in dieser Reihenfolge besucht werden, und die Länge der Tour ist $d(c_{\pi(n)}, c_{\pi(1)}) + \sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)})$. Das *Traveling-Salesperson-Problem* fragt nach einer Traveling-Salesperson-Tour minimaler Länge auf C . Die Entscheidungsversion *TSP* dieses Problems ist folgendermaßen definiert: Gegeben eine Menge C von n Städten, eine Distanzfunktion D auf C und eine positive Zahl $k \in \mathbb{Z}^+$, gibt es eine Traveling-Salesperson-Tour auf C , deren Länge höchstens k ist?

unter polynomialzeit-beschränkten randomisierten Reduktionen. Chang, Kadin und Rohatgi [CKR95] gingen noch einen Schritt weiter und analysierten das Schwellenverhalten solcher randomisierter Reduktionen. Sie bewiesen, dass wenn $\text{Unique-SAT} \leq_m^p$ -äquivalent zu seinem Komplement ist, dann gilt $\text{DP} = \text{coDP}$, und die boolesche Hierarchie über NP und die Polynomialzeit-Hierarchie kollabieren. Ebenso würde ein Kollaps der PH folgen, wenn Unique-SAT in coDP wäre.

Boolesche Hierarchie über NP: Cai et al. [CGH⁺88, CGH⁺89] verallgemeinerten DP zur booleschen Hierarchie über NP. Indem sie die unabhängig erzielten ursprünglichen Ergebnisse von Cai und L. Hemaspaandra [CH86] und von Gundermann, Wagner und Wechsung [Wec85, GW87] vereinigten, vereinheitlichten und erweiterten, stellt ihre Arbeit [CGH⁺88, CGH⁺89] eine fundamentale Untersuchung der grundlegenden strukturellen Eigenschaften der booleschen Hierarchie und ihrer komplexitätstheoretischen Anwendungen dar. Ausgehend von der frühen Arbeit von Hausdorff [Hau14], die boolesche Hierarchien über beliebigen Mengenringen analysiert, wird in [CGH⁺88, Wag87a, KSW87] die stufenweise Äquivalenz der verschiedenen Normalformen der booleschen Hierarchie über dem Mengenring NP gezeigt. Die Sätze 5.6, 5.8 und 5.10 sowie Korollar 5.7 findet man dort.

Diese Arbeit inspirierte eine Vielzahl weiterer Resultate über boolesche Hierarchien über NP und anderen Klassen. So untersuchten Wagner [Wag87a, Wag90], Köbler und Schöning [KSW87] die verschachtelte Differenzenhierarchie, die symmetrische Differenzenhierarchie und die erweiterte boolesche Hierarchie, und sie identifizierten viele natürliche Probleme als vollständig für die Stufen der booleschen Hierarchie über NP. Insbesondere ist die hinreichende Bedingung für den Nachweis der $\text{BH}_k(\text{NP})$ -Härte in Lemma 5.11 aus Wagners Arbeit [Wag87a] und ebenso die Übungen 5.11, 5.13, 5.14(a) und 5.14(b). Übung 5.12 ist aus [Rot03].

Gundermann, Nasser und Wechsung [GNW90] sowie Beigel, Chang und Ogi-hara [BCO93] untersuchten boolesche Hierarchien über Zählklassen. Bertoni et al. [BBJ⁺89] studierten boolesche Hierarchien über der Klasse RP, „Random Polynomial Time“ (siehe Kapitel 6), und über anderen Klassen. L. Hemaspaandra und Rothe [HR95, HR97b] untersuchten die Normalformen der booleschen Hierarchie über der Klasse UP, „Unambiguous Polynomial Time“, welche in Abschnitt 3.6.2 definiert wurde. Obwohl UP unter Durchschnitt abgeschlossen ist, ist es vermutlich nicht unter Vereinigung abgeschlossen und daher kein Mengenring; siehe Übung 5.6. In [HR95, HR97b] wird gezeigt, dass für bestimmte Normalformen der booleschen Hierarchie der Abschluss unter Durchschnitt der zugrunde liegenden Klasse \mathcal{C} genügt, um die gesamte boolesche Hülle von \mathcal{C} zu umfassen: Sowohl die Hierarchie der alternierenden Summen als auch die symmetrische Differenzenhierarchie über einer beliebigen Klasse, die unter Durchschnitt (aber womöglich nicht unter Vereinigung) abgeschlossen ist, umfasst die boolesche Hülle der Klasse. Insbesondere ist dieses Resultat auf UP anwendbar. Im Gegensatz zu NP (oder zu einem anderen Mengenring) umfassen weder die Hausdorff-Hierarchie noch die verschachtelte Differenzenhierarchie über UP die boolesche Hülle von UP in gewissen relativierten Welten [HR97b].

Exakte Probleme: Satz 5.12 geht auf Riege und Rothe [RR06b] zurück. In dieser Arbeit wird auch gezeigt, dass die exakten Versionen bestimmter verallgemeinerter Domatische-Zahl-Probleme vollständig für die Stufen der booleschen Hierarchie über NP sind. Diese verallgemeinerten Domatische-Zahl-Probleme wurden von Heggernes und Telle [HT98] eingeführt, deren Methode eine einheitliche Definition einer großen Vielzahl von Graphenproblemen erlaubt. Ihr Zugang verwendet Zerlegungen der Knotenmenge eines Graphen, so dass die Anzahl der Nachbarn für jeden Knoten in der Zerlegung gewissen Einschränkungen unterliegt. Weiterhin wird in [RR06b] bewiesen, dass das exakte Conveyer-Flow-Shop-Problem vollständig für die Stufen der booleschen Hierarchie über NP ist. Dieses Problem tritt in Anwendungen zur Logistik im Großhandel auf, wenn Warenhäuser mit Gütern aus einem zentralen Lager beliefert werden, und es wurde von Espelage und Wanke [EW00, Esp01, EW01, EW03] eingeführt und ausführlich untersucht.

Lemma 5.17, das in Kapitel 3 bewiesen wurde und das die Standardreduktion von 3-SAT auf 3-Colorability liefert, geht auf Stockmeyer [Sto73], Garey und Johnson [GJS76] zurück. Lemma 5.18 gibt die noch stärkere Reduktion $3\text{-SAT} \leq_m^p 3\text{-Colorability}$ von Guruswami und Khanna [GK00] an. Ursprünglich war ihre Arbeit nicht vom Problem der Härte der exakten Färbbarkeit, sondern von der Approximationshärte der chromatischen Zahl von Graphen motiviert.

PCP-Theorem: Ein sehr mächtiges Werkzeug zum Nachweis der Nichtapproximierbarkeit harter kombinatorischer Probleme ist das PCP-Theorem, das auch mit dem Begriff der interaktiven Beweissysteme zu tun hat (der verwandte Begriff der Arthur-Merlin-Spiele wird in den Abschnitten 6.3 und 8.4 erläutert). Insbesondere wenden Arora, Lund, Motwani, Sudan und Szegedy [ALM⁺98] Methoden der Beweisverifikation an, um die Approximationshärte von Problemen zu zeigen. Mehr über Approximation und Nichtapproximierbarkeit findet man z.B. in Ausiello et al. [ACG⁺03], Vazirani [Vaz03] und in dem umfassenden, aktuellen Kompendium der NP-Optimierungsprobleme, das von Crescenzi, Kann, Halldórsson, Karpinski und Woeginger herausgegeben wird:

<http://www.nada.kth.se/~viggo/problemlist/compendium.html>

PCP ist ein Akronym für „probabilistically checkable proof system“. Das PCP-Theorem, das auf Arora et al. [AS98, ALM⁺98] zurückgeht, sagt, dass NP durch solche Beweissysteme mit nur logarithmisch vielen Zufallsbits und mit einer konstanten Zahl von Fragen des Verifizierers charakterisiert werden kann. Für genaue Definitionen und mehr Details und Literaturhinweise sei auf den Übersichtsartikel von Goldreich [Gol97b] sowie die Dissertationen von Arora [Aro94] und Sudan [Sud95] verwiesen; siehe auch Zimand [Zim04].

Exakte Färbbarkeit: Viele starke Resultate über die Nichtapproximierbarkeit von harten Problemen wurden mit Hilfe des PCP-Theorems bewiesen. Um ein solches Resultat zu nennen: Khanna, Linial und Safra [KLS00] zeigten mit dem PCP-Theorem, dass es NP-hart ist, „einen 3-färbaren Graphen mit nur vier Farben zu färben“. Guruswami und Khanna [GK00] gaben einen neuen Beweis für dasselbe

Resultat, der auf der oben erwähnten cleveren Konstruktion beruht, die im Beweis von Lemma 5.18 präsentiert wurde. Diese direkte Transformation, die nicht vom PCP-Theorem abhängt, wurde hier angewandt, um die Frage der genauen Komplexität des exakten Färbarkeitsproblems zu beantworten. Diese Frage wurde explizit bereits 1987 von Wagner [Wag87a, p. 70] aufgeworfen. Unter Verwendung der Lemmata 5.17 und 5.18 wird in Satz 5.20 gezeigt, dass **Exact-4-Colorability** DP-vollständig ist. Dieses Resultat geht auf Rothe [Rot03] zurück; siehe auch die gemeinsame Arbeit mit Spakowski und Vogel [RSV02]. Im Gegensatz zu diesem Vollständigkeitsresultat ist **Exact-3-Colorability** in NP nach Behauptung 5.16.

Die Polynomialzeit-Hierarchie: Die Polynomialzeit-Hierarchie wurde von A. Meyer und Stockmeyer [MS72, Sto77] eingeführt. Die Sätze 5.24, 5.30, 5.31 und 5.33 sowie Korollar 5.32 gehen auf sie und auf Wrathall [Wra77] zurück. Historisch gesehen lag eine der Motivationen für die Einführung der Polynomialzeit-Hierarchie in dem Wunsch, die Komplexität von Minimierungsproblemen für boolesche Formeln möglichst präzise zu bestimmen. Betrachte zum Beispiel das Problem **Minimal** (siehe [MS72]), die Menge der booleschen Formeln, für die es keine kürzere (bzgl. der Anzahl der Variablen vorkommen) äquivalente Formel gibt. In [MS72] wurde Π_2^P als die beste obere Schranke für dieses Problem angegeben, aber keine untere Schranke. Eine der Varianten dieses Problems ist **MEE-DNF** (siehe [Sto77]): Gegeben ein Paar $\langle \varphi, k \rangle$, wobei φ eine boolesche Formel in DNF und k eine positive ganze Zahl ist, gibt es eine zu φ äquivalente boolesche Formel in DNF, in der höchstens k Variablen vorkommen? Es ist leicht zu sehen, dass **MEE-DNF** coNP-hart ist; siehe Übung 5.27(b). Jedoch schlug jeder Versuch, coNP-Vollständigkeit für dieses Problem zu zeigen, fehl. Mittels der Wagner-Technik gelang es E. Hemaspaandra und Wechsung [HW97, HW02] mehr als ein Vierteljahrhundert später, die untere Schranke für **MEE-DNF** von coNP-Härte auf Θ_2^P -Härte anzuheben. Andererseits sieht man unmittelbar, dass **MEE-DNF** in Σ_2^P liegt; siehe Übung 5.27(a). Umans [Uma01] klassifizierte **MEE-DNF** schließlich genau, indem er zeigte, dass es ein Σ_2^P -vollständiges Problem ist. Lässt man die Einschränkung, dass die gegebene und die zu ihr äquivalente minimale Formel in DNF vorliegen, in der Definition von **MEE-DNF** fallen, so erhält man das **Minimum-Equivalent-Expression-Problem MEE** (siehe [GJ79]), welches trivialerweise in Σ_2^P liegt und für das nach dem ebenfalls auf einer Anwendung der Wagner-Technik beruhenden Resultat von E. Hemaspaandra und Wechsung [HW97] Θ_2^P -Härte bekannt ist. Außerdem zeigten sie, dass **Minimal** coNP-hart ist, die erste und bisher beste untere Schranke für dieses Problem. Die präzise Komplexität von **MEE** ist immer noch unbekannt.

Eiter und Gottlob [EG93] untersuchten Probleme aus der Logik und der künstlichen Intelligenz. Sie bewiesen u.a. die Π_2^P -Vollständigkeit des Deduktionsproblems für beliebige propositionale Theorien unter der so genannten „*erweiterten abgeschlossene-Welt*“-Annahme bzw. unter „*circumscription*“. Dies sind Techniken, die in der nicht-monotonen Logik auftreten, welche sich mit dem logischen Schließen bei unvollständigem Wissen befasst.

Schaefer und Umans [SU02a, SU02b] haben einen sehr umfassenden, zweiteiligen Übersichtsartikel verfasst, der eine Vielzahl von natürlichen Problemen enthält, die für die zweite Stufe oder höhere Stufen der Polynomialzeit-Hierarchie

vollständig sind. Darin erwähnen sie auch eine Reihe interessanter offener Probleme, einschließlich der oben erwähnten Frage nach der genauen Komplexität von MEE.

Agrawal und Thierauf [AT01] studierten FI, das Formelisomorphie-Problem: Gegeben zwei boolesche Formeln φ und ψ , entscheide, ob es eine Permutation der Variablen von φ gibt, unter der φ und ψ äquivalent werden. Offenbar ist FI in Σ_2^P enthalten. Mit Techniken, die mit interaktiven Beweissystemen zu tun haben, und unter Verwendung eines Resultates von Bshouty et al. [BCKT94] aus der Lerntheorie bewiesen Agrawal und Thierauf [AT01], dass FI nicht Σ_2^P -vollständig sein kann, außer wenn die Polynomialzeit-Hierarchie kollabiert. Daher scheint die Komplexität des Formelisomorphie-Problems zwischen der ersten und zweiten Stufe der PH zu liegen, gerade so, wie die Komplexität des Graphisomorphie-Problems zwischen der nullten und ersten Stufe der PH liegt. Mehr Information über boolesche Isomorphie- und Äquivalenzprobleme findet man in [Thi00]; siehe auch die diesbezüglichen Arbeiten von Borchert, Ranjan und Stephan [BRS98, BR93], von Böhler et al. [BHRV02, BHRV04], von Bauland und E. Hemaspaandra [BH04] und von Borchert, L. Hemaspaandra und Rothe [BHR00].

Der FP^{NP} -Algorithmus für die Präfixsuche in Abbildung 5.9 ist die Standardmethode zum Finden einer lexikographisch kleinsten Lösung für eine positive Instanz eines NP-Problems. FP^{NP} ist das funktionale Analogon von $\Delta_2^P = P^{NP}$. Krentel [Kre88] und Wagner [Wag87a] bewiesen viele Δ_2^P -Vollständigkeitsresultate für NP-Optimierungsprobleme, etwa für das Problem Odd-Max-SAT, das fragt, ob die größte erfüllende Belegung einer gegebenen booleschen Formel ungerade ist oder nicht. Papadimitriou [Pap84] zeigte die Δ_2^P -Vollständigkeit des Problems, zu entscheiden, ob es eine *eindeutige* optimale Traveling-Salesperson-Tour für eine gegebene Distanzfunktion auf einer Menge von Städten gibt oder nicht. Große, Rothe und Wechsung [GRW06] bewiesen, dass es Δ_2^P -hart ist, die lexikographisch kleinste Vierfärbung für planare Graphen zu berechnen. Dieses Resultat verbessert ein Resultat von Khuller und V. Vazirani [KV91] optimal, die als untere Schranke lediglich NP-Härte für dieses Problem zeigten, woraus sie schlossen, dass es im Sinne von Schnorr [Sch76, Sch79] nicht selbstreduzierbar sein kann, außer wenn $P = NP$ gilt. Chen und Toda [CT95] schlügen einen allgemeinen Zugang zur Untersuchung der Komplexität von Optimierungsproblemen vor und erzielten Vollständigkeitsresultate für coNP, DP und Π_2^P .

Komplexitätsbeschränkte Reduzierbarkeiten: Cook [Coo71] führte die polynomialzeit-beschränkte Turing-Reduzierbarkeit \leq_T^P ein. Die polynomialzeit-beschränkte Truth-table-Reduzierbarkeit, \leq_{tt}^P , wurde von Ladner, Lynch und Selman [LLS75] eingeführt und gründlich untersucht. Ihre Arbeit stellt noch immer eine der tiefgründigsten Quellen in Bezug auf polynomialzeit-beschränkte Reduzierbarkeiten dar, während die beste Quelle für logarithmisch raum-beschränkte Reduzierbarkeiten die Arbeit [LL76] ist; siehe auch die Arbeit von Buhrman, E. Hemaspaandra und Torenvliet [BST93b, BST93a] für einen Vergleich solcher Reduzierbarkeiten. Jede hier diskutierte Reduzierbarkeit ist das komplexitätsbeschränkte Analogon ihres entsprechenden Pendants in der Berechenbarkeitstheorie. Selman [Sel82b] führte den Begriff der positiven Turing-Reduzierbarkeit ein, \leq_{pos-T}^P ; Übung 5.19 ist ihm zu verdanken. Außerdem untersuchte er intensiv verschiedene polynomialzeit-beschränkte

Reduzierbarkeiten auf NP, wobei er ihre Eigenschaften mit anderen komplexitäts-theoretischen Begriffen in Zusammenhang brachte, wie etwa mit der Selbstreduzierbarkeit, der P-Selektivität und den so genannten „*tally*“ (d.h. unär codierten) Mengen; siehe [Sel79, Sel82a, Sel82b, Sel88a]. Eine wunderbare Quelle zur Theorie der P-Selektivität ist das Buch von L. Hemaspaandra und Torenvliet [HT03a].

Behauptung 5.40, Lemma 5.43 und Satz 5.44 gehen auf Wagner [Wag87a] zurück. Explizite Beweise der Θ_2^P -Vollständigkeit von IN-Equ und IN-Geq sind zwar nicht in seiner Arbeit [Wag87a] enthalten, können aber in [HR97a] bzw. in [SV00] gefunden werden. Man bemerke, dass Wagner, im Unterschied zur Aussage von Lemma 5.43, eigentlich ein Kriterium für P_{bf}^{NP} -Härte in [Wag87a] angibt, wobei P_{bf}^{NP} als die Hülle von NP unter einer Einschränkung der \leq_t^P -Reduzierbarkeit definiert ist, bei welcher die boolesche Funktion τ aus Definition 5.41 nicht als boolescher Schaltkreis, sondern als eine boolesche Formel codiert wird. Jedoch zeigte Wagner in [Wag90] neben weiteren Charakterisierungen von Θ_2^P , dass $P_{bf}^{NP} = P_{tt}^{NP} = \Theta_2^P$ gilt.

Computational Social Choice: Das Studium von Wahlsystemen hinsichtlich der Komplexität der mit ihnen assoziierten Berechnungsprobleme wurde von Bartholdi, Tovey und Trick [BTT89b, BTT89a, BTT92] initiiert. Sie zeigten, dass das Gewinner- und das Rankingproblem für Dodgson- und für Kemeny-Wahlen jeweils NP-hart ist [BTT89b]. Das Dodgson-Wahlsystem wurde 1876 von Charles L. Dodgson [Dod76] vorgeschlagen, den man heute besser unter seinem Pseudonym als Schriftsteller kennt, Lewis Carroll. Sein Wahlsystem ist dem von H. Young in dem Sinn ähnlich, dass es ebenfalls das Condorcet-Prinzip durch eine Veränderung von Präferenzprofilen erweitert. Anders als Young ist Dodgson jedoch der Meinung, dass wir dem Condorcet-Prinzip dann am treuesten folgen, wenn die Wahl von einem Kandidaten gewonnen wird, der in dem folgenden Sinn einem Condorcet-Gewinner am „nächsten“ ist: Um „Nähe“ zu definieren, erhält jeder Kandidat c in einer gegebenen Wahl $\langle C, V \rangle$ eine Zahl, den so genannten Dodgson-Score. Der Dodgson-Score von c ist die kleinste Zahl aufeinander folgender Vertauschungen benachbarter Kandidaten in den Wählerpräferenzen, die nötig ist, c zum Condorcet-Gewinner zu machen. Jeder Kandidat mit minimalem Dodgson-Score ist ein Dodgson-Gewinner. Das Problem, für einen gegebenen Kandidaten und ein Präferenzprofil zu entscheiden, ob der Dodgson-Score dieses Kandidaten einen vorgegebenen Wert überschreitet oder nicht, ist NP-vollständig [BTT89b].

Die Ergebnisse von Bartholdi, Tovey und Trick, dass das Gewinner- und das Rankingproblem für Dodgson-Wahlen NP-hart sind [BTT89b], wurden beide optimal zu Θ_2^P -Vollständigkeit verbessert durch E. und L. Hemaspaandra und Rothe [HHR97a], mit einer direkten Anwendung von Lemma 5.43. E. Hemaspaandra, Spakowski und Vogel [HSV05] erzielten analoge Resultate für Kemeny-Wahlen. Die entsprechenden Resultate für Young-Wahlen, die auf Rothe, Spakowski und Vogel [RSV02, RSV03] zurückgehen, wurden hier in den Sätzen 5.49 und 5.51 angegeben. Weiterhin zeigten sie, dass das Gewinner- und Rankingproblem für die „homogene“ Variante des Wahlsystems von Dodgson,¹² die von Fishburn [Fis77] vorgeschlagen wurde, mittels

¹² Homogenität ist eine weitere recht nützliche Eigenschaft von Wahlsystemen, die in der Social-Choice-Theorie intensiv untersucht wurde. Informal gesagt ist ein Wahlsystem f

linearer Programmierung effizient gelöst werden kann. Ihr Beweis beruht auf einem ganzzahligen linearen Programm von Bartholdi et al. [BTT89b]. Young [You77] definierte sein Wahlsystem ursprünglich in der homogenen Variante und zeigte ebenfalls, dass die entsprechenden Probleme effizient mit linearer Programmierung gelöst werden können. Dies sollte mit den Aussagen der Sätze 5.49 und 5.51 kontrastiert werden.

Bartholdi, Tovey und Trick [BTT89a, BTT92] untersuchten auch Komplexitätsfragen im Hinblick auf strategisches Wählen und die Manipulation und prozedurale Kontrolle von Wahlsystemen (siehe auch z.B. [CS02, CLS03, HHR05, FHH06, HH07, HHR07b, FHHR07, HHR07a, EHRS07, ENR08a, ENR08b, FHHR08]).

Dwork, Kumar, Naor und Sivakumar [DKNS01] bewiesen, dass bei Verfahren zur Präferenzaggregation, die vom Kemeny-Wahlsystem inspiriert sind, Berechnungshärte einen nützlichen Schutz gegen „Spam“ (im Sinne einer Manipulation der Website-Rankings von Meta-Suchmaschinen) leisten kann. Sie entwickelten eine effiziente Heuristik, *lokale Kemenisierung* genannt, die auf einer Erweiterung des Condorcet-Prinzips beruht.

L. Hemaspaandra, Rajasekharan, Sethupathy und Zimand [HRSZ98] untersuchten verschiedene Algorithmen für eine gerechte Aufteilung („*Apportionment*“) der Sitze im US-Kongress. Sie testeten die vorhandenen Algorithmen im Vergleich mit einem neu entworfenen Algorithmus, der eine heuristische, auf *simulated annealing* beruhende Lösungssuche mit einer exakten Auswertung der Lösung durch dynamische Programmierung kombiniert. Die empirische Auswertung anhand von historischen Daten der USA zeigt, dass dieser neue Algorithmus gerechter ist als die früher benutzten und sogar als der gegenwärtig benutzte Aufteilungsalgorithmus. Für weitere Resultate aus dem Gebiet der *Computational Social Choice* sei auf die Übersichtsartikel [FHHR, HH00] und die darin zitierten Quellen verwiesen. Für mehr Hintergrund zur klassischen Social-Choice-Theorie sind die Bücher [MU95, Bla58, Arr63, Saa95, Saa01] empfehlenswert.

Paralleler Zugriff auf NP: Es gibt noch viele weitere Anwendungen von Lemma 5.43 in der Literatur; siehe die Referenzen in den Übersichtsartikeln [RR06a, HHR97b]. Beispielsweise zeigten E. Hemaspaandra, Rothe [HR98] und Spakowski [HRS06], dass es vollständig für parallelen Zugriff auf NP ist, diejenigen Graphen zu erkennen, deren unabhängige Mengen bzw. Knotenüberdeckungen durch bestimmte effiziente Heuristiken gut approximiert werden können.

Boolesche und Fragehierarchien über NP: Köbler, Schöning, Wagner und Wechsung [Wec85, KSW87, Wag87b, Wag89, Wag90], Krentel [Kre88], Buss und Hay [BH88, BH91] und Beigel [Bei91a] untersuchten die boolesche Hierarchie und die Fragehierarchien über NP. Satz 5.53 ist aus [Bei91a], wo die erste Aussage des Satzes Wagner und Wechsung zugeschrieben wird (siehe [Wec85]) und seine dritte Aussage Köbler, Schöning und Wagner [KSW87]. Die Gleichheit $BH(NP) = P_{\text{btt}}^{NP}$ aus

homogen, falls man nach Aufspalten eines jeden Wählers $v \in V$ in n neue Wähler, die jeweils dieselbe Präferenzliste wie v haben, genau dieselbe Menge von Gewinnern erhält. Fishburn [Fis77] zeigte, dass weder das Wahlsystem von Dodgson noch das von Young (wie in Abschnitt 5.3.2 definiert) homogen sind.

Korollar 5.54 geht auf Cai et al. [CGH⁺88] zurück. Die Gleichheit $P^{NP[\log]} = P_{tt}^NP$ aus Korollar 5.55 wurde unabhängig von L. Hemaspaandra [Hem87, Hem89], Buss und Hay [BH88, BH91] sowie von Köbler, Schöning und Wagner [KSW87, Wag90] entdeckt. Weitere Charakterisierungen von $\Theta_2^P = P^{NP[\log]}$ findet man in [Wag90].

Eiter und Gottlob [EG97] fanden logische Charakterisierungen der Klasse Θ_2^P und zeigten die Θ_2^P -Vollständigkeit einiger Probleme der Künstlichen Intelligenz und der modalen Logik. Castro und Seara [CS92] charakterisierten die Klassen $P^{NP[\mathcal{O}(\log^k)]}$, die zwischen Θ_2^P und Δ_2^P liegen. Rohatgi [Roh95] bewies scharfe Schranken für die Schwellwerte von polynomialzeit-beschränkten randomisierten Reduktionen für die Klassen der booleschen und der Fragehierarchien über NP.

Boolesche und Polynomialzeit-Hierarchie: Kadin [Kad88] untersuchte als Erster die Verbindung der booleschen und der Polynomialzeit-Hierarchie, die in Abschnitt 5.5 präsentiert wurde. Satz 5.57 ist ihm zu verdanken. Das Resultat von Yap, hier als Lemma 5.59 angegeben und im Beweis von Satz 5.57 verwandt, ist auch an sich, unabhängig von dieser Anwendung, interessant. Es gibt eine Reihe von Ergebnissen, die Lemma 5.59 ähnlich sind und es verstärken. Die ursprüngliche Motivation von Lemma 5.59 hat mit dem Satz von Karp und Lipton [KL80] zu tun: Gibt es für NP eine \leq_T^p -harte dünne Menge S (d.h., gilt $NP \subseteq P^S$ oder, äquivalent dazu,¹³ hat NP Schaltkreise polynomialer Größe), dann gilt $PH = \Sigma_2^P \cap \Pi_2^P$. Vergleicht man diese beiden Resultate, so hat Lemma 5.59, nach welchem aus der Inklusion $coNP \subseteq NP^S$ der Kollaps $PH = \Sigma_3^P \cap \Pi_3^P$ folgt, sowohl eine schwächere Annahme als auch eine schwächere Konklusion als das Karp–Lipton-Theorem. Somit sind die beiden Resultate zwar miteinander verwandt, aber nicht unmittelbar vergleichbar.

Konsequenzen aus der Existenz dünner vollständiger Mengen: Mahaney [Mah82] bewies, dass wenn eine dünne Menge nicht nur \leq_T^p -hart, sondern \leq_T^p -vollständig (also in NP) ist, dann kollabiert PH sogar auf $\Delta_2^P = P^{NP}$. Long [Lon82a] verallgemeinerte dieses Ergebnis von Mahaney: Wenn es eine dünne Menge S in Δ_2^P gibt, für die $NP \subseteq P^S$ gilt, dann ist $PH = \Delta_2^P$. Indem er Mahaneys Ergebnis verstärkte, bewies Kadin [Kad89] den optimalen Kollaps: Wenn es eine dünne Menge S in NP gibt, so dass $coNP \subseteq NP^S$ gilt, dann ist $PH = \Theta_2^P = P^{NP[\log]}$. Gibt es also eine dünne \leq_T^p -vollständige Menge in NP, dann folgt $PH = \Theta_2^P = P^{NP[\log]}$. Dieser Kollaps stellt das beste mit relativierbaren Techniken erreichbare Resultat dar, denn Kadin [Kad89] bewies auch, dass es für jede Funktion f in $o(\log n)$ eine relativierte Welt gibt, in der $P^{NP[\log]} \not\subseteq P^{NP[f(n)]}$ gilt, doch in der NP eine dünne \leq_T^p -vollständige Menge hat.

Reduktionen auf dünne Mengen: Das Studium der Reduktionen auf dünne Mengen hat eine lange und reiche Tradition in der Komplexitätstheorie; siehe die Übersichtsartikel [Mah86, Mah89, HOW92, You92, AHH⁺93]. Insbesondere stehen die dabei untersuchten Fragen im Zusammenhang mit der berühmten Isomorphie-Vermutung von Berman und Hartmanis; siehe Vermutung 3.73 in Abschnitt 3.6.2. Zur Erinnerung: In diesem Abschnitt wurde erwähnt, dass Mahaney [Mah82] eine damit

¹³ Diese Äquivalenz, die erstmalig von A. Meyer bemerkt wurde, erscheint in [BH77]. Das Karp–Lipton-Theorem selbst geht in erster Linie auf Karp und Lipton [KL80] zurück, die allerdings auch einen Beitrag Sipsers dazu anerkennen.

verwandte Vermutung von Berman und Hartmanis löste, da er nämlich bewies, dass aus der Existenz einer dünnen \leq_m^p -vollständigen Menge in NP die Gleichheit $P = NP$ folgt; siehe Satz 3.75. Vergleicht man dieses Ergebnis mit Kadins Resultat oben, so sieht man, dass Mahaneys Implikation sowohl eine stärkere Annahme (\leq_m^p -Vollständigkeit versus \leq_T^p -Vollständigkeit) als auch eine stärkere Kollapskonsequenz hat ($P = NP = PH$ versus $PH = \Theta_2^P$) als Kadins Implikation. Was ist nun aber mit den Reduktionen zwischen \leq_m^p und \leq_T^p ? Was folgt insbesondere aus der Annahme, jede NP-Menge wäre \leq_{tt}^p -reduzierbar (mit einer unbeschränkten oder aber mit einer beschränkten Anzahl von Fragen) auf eine dünne Menge? Watanabe untersuchte polynomialzeit-beschränkte 1-truth-table-Reduktionen von NP-Mengen auf dünne Mengen. Insbesondere bewies er, dass wenn $NP \subseteq P_{1-tt}^S$ für eine dünne Menge S gilt, dann ist $P = \text{FewP}$ und $NP = RP$, wobei RP die Klasse „Random Polynomial Time“ bezeichnet, die in Kapitel 6 definiert wird. Ogihara und Watanabe [OW91] verstärkten Mahaneys Resultat, indem sie zeigten, dass wenn $NP \subseteq P_{\text{bit}}^S$ für eine dünne Menge S gilt, dann ist $P = NP$. Homer und Longpré [HL94] vereinfachten ihren Beweis und erweiterten ihr Resultat auf \leq_{tt}^p - und \leq_T^p -Reduktionen von NP-Mengen auf dünne Mengen mit einer logarithmischen Anzahl von Fragen. Arvind und Torán [AT99] bewiesen, dass wenn eine NP-vollständige Menge oder eine coNP-vollständige Menge in Polynomialzeit disjunktiv-truth-table-reduzierbar auf eine dünne Menge ist, dann fallen die Funktionenanaloga von P_{tt}^{NP} und $P^{NP[\log]}$ zusammen. Die disjunktive Truth-table-Reduzierbarkeit ist eine spezielle \leq_{tt}^p -Reduzierbarkeit, die genau dann akzeptiert, wenn mindestens eine Frage positiv beantwortet wurde. Arvind und Torán [AT99] zeigten außerdem, dass wenn eine NP-vollständige Menge oder eine coNP-vollständige Menge in Polynomialzeit disjunktiv-truth-table-reduzierbar auf eine polylogarithmisch-dünne Menge ist, dann gilt $P = NP$.

Kadins Resultat, dass der Kollaps $PH = \Theta_2^P$ aus der Existenz einer dünnen \leq_T^p -vollständigen Menge in NP folgt, hat sowohl eine stärkere Annahme als auch eine stärkere Konklusion als das Karp-Lipton-Theorem, welches sagt, dass der Kollaps $PH = \Sigma_2^P \cap \Pi_2^P$ aus der Existenz einer dünnen \leq_T^p -harten Menge für NP folgt. Köbler und Watanabe [KW98] hingegen erzielten echte Verbesserungen des Karp-Lipton-Theorems: Gilt $NP \subseteq P^S$ für eine dünne Menge S , so ist $PH = ZPP^{NP}$, wobei ZPP die Klasse „Zero-error Probabilistic Polynomial Time“ bezeichnet, die in Abschnitt 6.2.1 definiert wird. Es gilt $ZPP \subseteq NP$. Der stärkste derzeit bekannte Karp-Lipton-artige Satz geht auf Cai, Chakaravarthy, L. Hemaspaandra und Ogihara [CCHO03] zurück: Gilt $NP \subseteq P^S$ für eine dünne Menge S , so ist $PH \subseteq S_2^P$. Die Klasse S_2^P wurde unabhängig von Canetti [Can96] und von Russell und Sundaram [RS98] eingeführt. S_2^P ist Σ_2^P insofern ähnlich, als es auch mittels zweier alternierender Quantoren definiert ist. Der Unterschied ist, dass S_2^P auf der *symmetrischen Alternierung* von \exists^P - und \forall^P -Quantoren beruht: Eine Menge A ist in S_2^P , falls es eine Menge B in P und ein Polynom p gibt, so dass für jedes $x \in \Sigma^*$ gilt: $x \in A$ impliziert $(\exists^p y)(\forall^p z)[\langle x, y, z \rangle \in B]$, und $x \notin A$ impliziert $(\exists^p z)(\forall^p y)[\langle x, y, z \rangle \notin B]$. Man vergleiche diese Definition von S_2^P mit der Quantorencharakterisierung von Σ_2^P aus Satz 5.31. S_2^P steht auch in Beziehung zum Begriff konkurrierender Beweiser im Gebiet der interaktiven Beweissysteme. Nach Definition gilt $S_2^P \subseteq \Sigma_2^P \cap \Pi_2^P$. Cai [Cai01] bewies, dass $S_2^P \subseteq ZPP^{NP}$ gilt.

Verbesserungen der „Easy-Hard“-Technik: Wenden wir uns erneut Kadins Resultat zu, dass die Polynomialzeit-Hierarchie kollabiert, falls die boolesche Hierarchie kollabiert. Die diesem Resultat zugrunde liegende „Easy-Hard“-Technik wurde in einer langen Reihe von Arbeiten mehr und mehr verbessert. Wagner erzielte stärkere Kollapse der PH unter derselben Annahme: zunächst, in [Wag87b], einen Kollaps bis auf Δ_3^P ; dann, in [Wag89], einen Kollaps bis auf $\text{BH}(\Sigma_2^P)$, welches in $\Theta_3^P \subseteq \Delta_3^P$ enthalten ist. Chang und Kadin [Cha91, CK96] gelang es, noch etwas weiter zu gehen, indem sie zeigten, dass $\text{BH}_k(\text{NP}) = \text{coBH}_k(\text{NP})$ sogar einen Kollaps der PH auf $\text{BH}_k(\Sigma_2^P)$ impliziert. Beigel, Chang und Ogihara [BCO93] bewiesen, dass die Gleichheit $\text{BH}_k(\text{NP}) = \text{coBH}_k(\text{NP})$ den Kollaps $\text{PH} = (\text{P}_{(k-1)-\text{tt}}^{\text{NP}})^{\text{NP}}$ bewirkt; zu beachten ist hier, dass aus Satz 5.53 die Inklusion $(\text{P}_{(k-1)-\text{tt}}^{\text{NP}})^{\text{NP}} \subseteq \text{BH}_k(\Sigma_2^P)$ folgt. E. und L. Hemaspaandra und Hempel [HHH98b] (siehe auch Hempels Dissertation [Hem98]) und, unabhängig, Reith und Wagner [RW01] bewiesen, dass für jedes $k > 0$ und jedes $i > 0$ aus $\text{BH}_k(\Sigma_i^P) = \text{coBH}_k(\Sigma_i^P)$ der Kollaps $\text{PH} = \text{BH}_k(\Sigma_i^P) \Delta \text{BH}_{k-1}(\Sigma_{i+1}^P)$ folgt, wobei Δ die komplexe symmetrische Differenz von Mengenklassen bezeichnet.¹⁴ Insbesondere gilt für $i = 1$, dass aus $\text{BH}_k(\text{NP}) = \text{coBH}_k(\text{NP})$ der Kollaps $\text{PH} = \text{BH}_k(\text{NP}) \Delta \text{BH}_{k-1}(\Sigma_2^P)$ folgt. Derzeit ist dies das stärkste unter den Resultaten, die eine Beziehung zwischen einem Kollaps der Polynomialzeit-Hierarchie und einem Kollaps der booleschen Hierarchie über NP herstellen.

„Downward Collapse“-Ergebnisse: E. und L. Hemaspaandra und Hempel [HHH99] initiierten eine verwandte Forschungslinie, die eine als „downward collapse“ bezeichnete Eigenschaft der Polynomialzeit-Hierarchie untersucht. Eine unmittelbare Konsequenz der Sätze 5.53 und 5.57 ist, dass $\text{P}^{\text{NP}[1]} = \text{P}^{\text{NP}[2]}$ den Kollaps $\text{PH} = \Sigma_3^P \cap \Pi_3^P$ impliziert. Kann man unter derselben Annahme einen Kollaps sogar bis auf $\text{PH} = \Sigma_1^P \cap \Pi_1^P = \text{NP} \cap \text{coNP}$ erwarten? Ein Resultat in [HHH99] sagt, dass ein so starker Kollaps nach unten tatsächlich für die höheren Stufen der Polynomialzeit-Hierarchie gezeigt werden kann: Für jedes $i > 2$ folgt aus $\text{P}^{\Sigma_i^P[1]} = \text{P}^{\Sigma_i^P[2]}$ der Kollaps $\Sigma_i^P = \Pi_i^P$. Kurz darauf bewiesen Buhrman und Fortnow [BF98] das analoge Resultat für $i = 2$: $\text{P}^{\Sigma_2^P[1]} = \text{P}^{\Sigma_2^P[2]}$ impliziert $\Sigma_2^P = \Pi_2^P$. Im Gegensatz dazu lieferten sie ein relativiertes Gegenbeispiel für den Fall $i = 1$: Es gibt ein Orakel, relativ zu dem $\text{P}^{\text{NP}[1]} = \text{P}^{\text{NP}[2]}$ und aber $\text{NP} \neq \text{coNP}$ gilt. Indem sie die Techniken der vorherigen Arbeiten vereinheitlichten und verbesserten, gelang E. und L. Hemaspaandra und Hempel [HHH01] schließlich das allgemeinste und stärkste Resultat, das in dieser Hinsicht derzeit bekannt ist: Für jedes $k > 0$ und jedes $i > 1$ gilt

$$\text{P}_{k-\text{tt}}^{\Sigma_i^P} = \text{P}_{(k+1)-\text{tt}}^{\Sigma_i^P} \implies \text{BH}_k(\Sigma_i^P) = \text{coBH}_k(\Sigma_i^P).$$

Diese Translation der Gleichheit nach unten stellt eine sehr straffe Verbindung her zwischen einem Kollaps der parallelen Fragehierarchie über Σ_i^P und einem Kollaps der booleschen Hierarchie über Σ_i^P .

¹⁴ Für Klassen \mathcal{C} und \mathcal{D} von Mengen ist $\mathcal{C} \Delta \mathcal{D}$ die Klasse der symmetrischen Differenzen von je zwei Mengen A und B , wobei $A \in \mathcal{C}$ und $B \in \mathcal{D}$.

„*Query Order*“-Ergebnisse: Eine weitere hiermit verwandte Forschungslinie wurde von L. Hemaspaandra, Hempel und Wechsung [HHW99] initiiert, die die Frage studierten, ob und in welchem Ausmaß die Reihenfolge eine Rolle spielt, in der auf verschiedene Orakelmengen zugegriffen wird. Insbesondere charakterisierten sie für DPOTMs, die jeweils höchstens eine Frage an zwei Orakelmengen aus verschiedenen Stufen der booleschen Hierarchie über NP stellen, genau diejenigen Fälle, in denen die Reihenfolge der Fragen (englisch „*query order*“) eine Rolle spielt, außer wenn die Polynomialzeit-Hierarchie kollabiert. Im Gegensatz zu Orakelmengen aus der booleschen Hierarchie zeigten E. und L. Hemaspaandra und Hempel [HHH98a], dass die Reihenfolge der Fragen nie eine Rolle spielt, wenn die Orakelmengen aus der Polynomialzeit-Hierarchie sind.

Weitere Hierarchien: Es wurden auch Hierarchien eingeführt, die der Polynomialzeit-Hierarchie ähnlich sind, aber auf anderen Klassen als NP beruhen. So wurden beispielsweise verschiedene Hierarchien, die jeweils auf der „*promise class*“ UP und auf unterschiedlichen Arten des Orakelzugriffs beruhen, von Lange, Niedermeier und Rossmanith [LR94, NR98], von Cai, L. Hemaspaandra und Vyskoč [CHV92, CHV93] und von L. Hemaspaandra und Rothe [HR97b] untersucht.

Alternierende Turingmaschinen: Das Modell der alternierenden Turingmaschine wurde von Chandra, Kozen und Stockmeyer [CKS81] eingeführt. Die Sätze 5.65, 5.66, 5.68 und 5.69 sowie die Korollare 5.67 und 5.70 sind ihnen zu verdanken. Zu beachten ist hier, dass auf die Annahme der Konstruierbarkeit der Ressourcenfunktionen, die in all diesen Resultaten außer in Satz 5.69 gemacht wurde, verzichtet werden kann, sofern man die Argumentation dann sorgfältig geeignet modifiziert. Zu beachten ist weiter, dass die Sätze 5.65 und 5.66 für Ressourcenfunktionen f mit $f \geq \log$ formuliert sind, wohingegen die entsprechenden Resultate in [CKS81] für Ressourcenfunktionen f mit $f \geq \text{id}$ bewiesen worden sind.

Die Low- und die High-Hierarchie: Die Begriffe der Lowness und Highness wurden in der Komplexitätstheorie von Schöning [Sch83] eingeführt, und Abschnitt 5.7 folgt in großen Teilen seiner Darstellung. Die Sätze 5.76, 5.77, 5.83 und 5.88, die Lemmata 5.80, 5.86 und 5.87 sowie die Korollare 5.78, 5.81, 5.82 und 5.84 sind ihm zu verdanken. Lowness und Highness in der Komplexitätstheorie sind von den entsprechenden Begriffen der Berechenbarkeitstheorie inspiriert; siehe [Soa77] und die Bücher von H. Rogers [Rog67] und Soare [Soa87]. Die in Definition 5.74 eingeführte starke nichtdeterministische polynomialzeit-beschränkte Turing-Reduzierbarkeit, \leq_{sT}^{NP} , geht auf Long [Lon82b] zurück. Dieser Reduzierbarkeitsbegriff beruht auf der von Adleman und Manders [AM77] definierten γ -Reduzierbarkeit. Diese ist die Many-one-Version von \leq_{sT}^{NP} . Die Charakterisierung von \leq_{sT}^{NP} in Lemma 5.75 wurde von Selman [Sel78] gefunden.

Im Anschluss an die wegweisende Arbeit von Schöning [Sch83] wurde das Studium der Lowness intensiv verfolgt. Insbesondere wurden die Lowness-Eigenschaften des Graphisomorphie-Problems und bestimmter probabilistischer Komplexitätsklassen in [Sch88, Sch87] untersucht. Einige Resultate dieser Arbeit werden später in Kapitel 6 vorgestellt. Ko und Schöning [KS85] bewiesen die Lowness von Mengen mit Schaltkreisen polynomialer Größe, einschließlich der p-selektiven Mengen.

Sie studierten auch eine Verfeinerung der Low-Hierarchie bezüglich der Δ_k^P -Stufen der Polynomialzeit-Hierarchie. Book, Orponen, Russo und Watanabe [BORW88] untersuchten den Begriff der Lowness in der Exponentialzeit-Hierarchie. Köbler, Schöning, Toda und Torán [KSTT92, KST92] zeigten, dass Probleme, die sich durch NPTMs mit „wenigen“ akzeptierenden Pfaden lösen lassen, insbesondere die Probleme in FewP, sowie das Graphisomorphie-Problem low für PP sind, „Probabilistic Polynomial Time“. Weitere Lowness-Ergebnisse für das Graphisomorphie-Problem werden in Kapitel 6 gegeben, wo einige dieser Resultate bewiesen werden.

Die Erweiterte Low-Hierarchie: Balcázar, Book und Schöning studierten Lowness und Highness im Zusammenhang mit dünnen Mengen. In ihrer Arbeit führten sie interessante Verallgemeinerungen der Begriffe Lowness und Highness ein, einschließlich der *Erweiterten Low-Hierarchie* [BBS86b]. Die k -te Stufe dieser Hierarchie ist definiert durch

$$\text{ELow}_k = \{L \mid \Sigma_k^{P,L} = \Sigma_{k-1}^{P,L \oplus \text{SAT}}\}.$$

Hier bezeichnet \oplus die *disjunkte* (oder *markierte*) *Vereinigung* zweier Mengen A und B , die durch $A \oplus B = \{0a \mid a \in A\} \cup \{1b \mid b \in B\}$ definiert ist. Zu den bemerkenswertesten Resultaten über erweiterte Lowness gehören das Ergebnis von Sheu und Long [SL94], dass die Erweiterte Low-Hierarchie echt unendlich ist, und Köblers Resultat, dass die Klasse der Mengen mit Schaltkreisen polynomiale Größe optimal in der Erweiterten Low-Hierarchie lokalisiert [Köb94]. Allender und Hemaspaandra [AH92b] bewiesen untere Schranken für die Klassen Low_k und ELow_k der Low- und der Erweiterten Low-Hierarchie.

Die Klassen ELow_k verhalten sich in mehr als einer Hinsicht anders als die meisten anderen Komplexitätsklassen. Beispielsweise ist die Klasse ELow_2 , anders als die meisten Standard-Komplexitätsklassen, nicht unter der \leq_m^P -Reduzierbarkeit abgeschlossen [AH92b]. Weiterhin bewiesen L. Hemaspaandra, Jiang, Rothe und Watanabe [HJR98], dass (a) ELow_2 unter bestimmten booleschen Operationen nicht abgeschlossen ist, so etwa nicht unter Vereinigung und nicht unter Durchschnitt, und dass (b) die oben definierte disjunkte Vereinigung die Komplexität bezüglich der erweiterten Lowness „verringern“ kann: Es gibt Mengen, die nicht in ELow_2 sind, doch ihre disjunkte Vereinigung ist in ELow_2 . Dieses Resultat erscheint auf den ersten Blick konterintuitiv, sofern man eine auf Reduktionen beruhende Intuition besitzt. Erweiterte Lowness ist jedoch kein reduktionsbasiertes Komplexitätsmaß. Stattdessen ist sie ein Maß dafür, wie schwer es ist, nützliche Information aus einem Orakel zu extrahieren. In [HJR97] werden untere Schranken hinsichtlich ELow_2 für gewissen Klassen bewiesen, die Selmans P-Selektivität verallgemeinern [Sel79]. Mehr Hintergrund zur Lowness und erweiterten Lowness findet der interessierte Leser in den exzellenten Übersichtsartikeln von L. Hemaspaandra [Hem93] und Köbler [Köb95].

Einige Relativierungsergebnisse: Zum Abschluss dieses Kapitels sollen noch einige Relativierungsergebnisse erwähnt werden, also Resultate der Form: „Es gibt ein Orakel A , so dass $\mathcal{C}^A \neq \mathcal{D}^A$ gilt“, oder: „Es gibt ein Orakel B , so dass $\mathcal{C}^B = \mathcal{D}^B$ gilt“, oder Kombinationen dieser beiden Aussagen. Hier sind \mathcal{C} und \mathcal{D} relativierbare Komplexitätsklassen, d.h., sie sind mittels eines Typs von Orakel-Turingmaschine

definiert. Beweist man solche Aussagen, so ist einem oft auch die Qualität der erreichten relativierten Separation von Komplexitätsklassen wichtig; beispielsweise hinsichtlich der Eigenschaften der separierenden Menge, die konstruiert wird, oder hinsichtlich der Eigenschaften der konstruierten Orakelmenge. So wird etwa eine „starke Separation“ durch eine „immune“ Menge bezeugt. Eine Menge heißt *immune gegen eine Mengenklasse \mathcal{C}* , falls sie eine unendliche Menge enthält, die keine unendliche Teilmenge in \mathcal{C} hat. Viele starke relativierte Separationsergebnisse sind bekannt; siehe z.B. [HM83, SB84, Bal85, BR88, TvEB89, BGY90, Ko90, Bru92, EHTY92, BCS92, HRW97a, Rot99]. Der Begriff der Immunität kommt ursprünglich aus der Berechenbarkeitstheorie. Ohne in die Einzelheiten zu gehen seien hier noch stärkere Varianten der Immunität erwähnt, wie etwa die „Bi-Immunität“ oder Müllers „balancierte Immunität“ [Mül93]. Lischke [Lis86, Lis99] studierte relativierte Beziehungen zwischen NP und Exponentialzeitklassen.

Bezüglich der Eigenschaften der Orakelmenge seien die Ergebnisse von Cai und Watanabe über Kollaps durch „strengen“ (englisch „*stringent*“) Orakelzugriff erwähnt [CW04] sowie Separationen durch „generische“ Orakel (siehe z.B. Fortnow und Yamakami [FY96]) und auch durch „zufällige“ (englisch „*random*“) Orakel.

Um ein Beispiel der letztgenannten Orakelseparationen zu nennen: Baker, Gill und Solovay [BGS75] bewiesen, dass relativ zu einem zufälligen Orakel R die Klasse NP^R eine P^R -bi-immune Menge mit Wahrscheinlichkeit eins enthält. Insbesondere gilt $\text{NP} \neq \text{P} = \text{NP} \cap \text{coNP}$ in dieser Welt. L. Hemaspaandra und Zimand [HZ96] zeigten, dass relativ zu einem zufälligen Orakel R die Klasse NP^R sogar eine P^R -balanciert-immune Menge mit Wahrscheinlichkeit eins enthält.

Randomisierte Algorithmen und Komplexitätsklassen

*,,Yet in vain a paynim foe
Armed with fate the mighty blow;
For when he fell, the Elfin queen,
All in secret and unseen,
O'er the fainting hero threw
Her mantle of ambrosial blue,
And bade her spirits bear him far,
In Merlin's agate-axed car,
To her green isle's enamelled steep,
Far in the navel of the deep.
O'er his wounds she sprinkled dew
From flowers that in Arabia grew.*

*There he reigns a mighty king,
Thence to Britain shall return,
If right prophetic rolls I learn,
Borne on victory's spreading plume,
His ancient sceptre to resume,
His knightly table to restore,
And brave the tournaments of yore“.*

*,,When Arthur bowed his haughty crest,
No princess veiled in azure vest
Snatched him, by Merlin's powerful spell,
In groves of golden bliss to dwell;
But when he fell, with winged speed,
His champions, on a milk-white steed,
From the battle's hurricane
Bore him to Joseph's towered fane,*

*In the fair vale of Avalon;
There, with chanted orison
And the long blaze of tapers clear,
The stoled fathers met the bier;
Through the dim aisles, in order dread
Of martial woe, the chief they led,
And deep entombed in holy ground,
Before the altar's solemn bound“.*

(Aus „Wharton's Ode“)

Dieses Kapitel behandelt randomisierte Algorithmen und die entsprechenden probabilistischen Komplexitätsklassen. Randomisierte Algorithmen sind oft effizienter als die besten bekannten deterministischen Algorithmen für dasselbe Problem, zahlen dafür jedoch den Preis, Fehler zu machen. Dies ist das Thema in Abschnitt 6.1, in dem ausgewählte deterministische und randomisierte Exponentialzeit-Algorithmen für das Erfüllbarkeitsproblem vorgestellt werden.

Randomisierte Algorithmen mit beschränktem Fehler, einschließlich so genannter Monte-Carlo- und Las-Vegas-Algorithmen, ermöglichen sehr nützliche Techniken zur Wahrscheinlichkeitsverstärkung, mittels derer die Fehlerwahrscheinlichkeit exponentiell klein in der Eingabegröße gemacht werden kann. In Abschnitt 6.2 werden verwandte probabilistische Komplexitätsklassen untersucht, wie etwa PP, RP, ZPP und BPP. Effiziente Monte-Carlo-Algorithmen für das Primzahlproblem werden später in Kapitel 7 präsentiert. Diese Algorithmen sind für praktische Zwecke sehr wichtig, besonders in kryptographischen Anwendungen, welche oft große zufällig gewählte Primzahlen verwenden. In solchen Anwendungen werden daher große zufällige Zahlen erzeugt und effizient auf Primheit getestet.

Abschnitt 6.3 führt den Begriff der Arthur-Merlin-Spiele ein und untersucht die Arthur-Merlin-Hierarchie. Genau wie die Polynomialzeit-Hierarchie kann die Arthur-Merlin-Hierarchie durch polynomiell längenbeschränkte Quantoren beschrieben werden, wobei die Randomisierung durch den der Klasse BPP entsprechenden Mehrheitsquantor ausgedrückt wird. Arthur-Merlin-Spiele sind eng mit dem Begriff der interaktiven Beweissysteme verwandt. Beide Begriffe kombinieren Randomisierung mit Nichtdeterminismus, woraus ihre Berechnungskraft erwächst, und sind sowohl in komplexitätstheoretischer als auch in kryptographischer Hinsicht interessant. Insbesondere sind Zero-Knowledge-Protokolle interaktive Beweissysteme mit bestimmten kryptographisch nützlichen Eigenschaften. So können sie zum Beispiel zum Zweck der Authentikation eingesetzt werden. Ein Zero-Knowledge-Protokoll für das Graphisomorphie-Problem wird später in Abschnitt 8.4 vorgestellt.

In Abschnitt 6.4 werden die Zählklassen #P und GapP eingeführt, mit deren Hilfe probabilistische Komplexitätsklassen durch die Anzahl der akzeptierenden und ablehnenden Berechnungspfade nichtdeterministischer Turingmaschinen charakterisiert werden können. In Abschnitt 6.5 werden bestimmte Lowness-Eigenschaften des Graphisomorphie-Problems bewiesen, einschließlich der Lowness für probabilistische Klassen. Diese Beweise verwenden Arthur-Merlin-Spiele, universelles Hashing, gruppentheoretische Algorithmen und andere Begriffe, in denen Randomisierung eine Rolle spielt.

6.1 Das Erfüllbarkeitsproblem der Aussagenlogik

Nach den Sätzen 3.49 und 3.51 sind das Erfüllbarkeitsproblem SAT und seine Einschränkung 3-SAT NP-vollständig. Deshalb würde, wäre SAT in P, sofort $P = NP$ folgen, was als sehr unwahrscheinlich angesehen wird. Als ebenso unwahrscheinlich gilt also die Existenz effizienter deterministischer Algorithmen für SAT oder 3-SAT.

Aber was ist eigentlich die beste bekannte Laufzeit eines deterministischen Algorithmus für das Erfüllbarkeitsproblem? Offenbar hängt diese von der Struktur der gegebenen Formel ab. Der Einfachheit halber werden wir uns hier auf 3-SAT konzentrieren, also nur solche Probleminstanzen betrachten, bei denen jede Klausel genau drei Literale hat. Die in diesem Abschnitt vorgestellten Ergebnisse können unmittelbar auf k -SAT verallgemeinert werden, wobei jede Klausel genau k Literale hat.

Der „naive“ deterministische Algorithmus für 3-SAT arbeitet so: Gegeben eine boolesche Formel φ mit n Variablen, überprüfe nacheinander alle möglichen Belegungen und akzeptiere, falls und sobald eine erfüllende Belegung gefunden ist. Eine erfüllbare Formel φ wird also auf jeden Fall akzeptiert. Ist φ jedoch unerfüllbar, dann lehnt der Algorithmus seine Eingabe ab, nachdem sich sämtliche 2^n Belegungen als nicht erfüllend erwiesen haben. Dieser Algorithmus benötigt $\mathcal{O}(n^2 \cdot 2^n)$ Schritte; er läuft also in der Zeit $\tilde{\mathcal{O}}(2^n)$.¹ Gibt es eine bessere obere Schranke?

¹ In diesem Abschnitt vernachlässigen wir, wie üblich, polynomiale Faktoren in den Zeitschranken von Exponentialzeit-Algorithmen, was durch den Gebrauch der $\tilde{\mathcal{O}}$ -Notation angezeigt wird.

Und ob. Aber bevor wir uns ansehen, *wie* man die $\tilde{\mathcal{O}}(2^n)$ -Schranke schlägt, stellen wir eine andere Frage: *Warum?* Warum sollte man die triviale obere Zeitschranke $t \in \tilde{\mathcal{O}}(2^n)$ für 3-SAT zu einer etwas besseren, aber immer noch exponentiellen Schranke verbessern, etwa zu $\hat{t} \in \tilde{\mathcal{O}}(c^n)$ für eine Konstante c mit $1 < c < 2$? Der Grund ist, dass für kleine Eingabegrößen sogar Exponentialzeit-Algorithmen als praktisch machbar betrachtet werden dürfen, auch wenn an einem Punkt n_0 das exponentielle Wachsum zuschlägt und die absolute Laufzeit des Algorithmus zu groß wird, um noch praktikabel zu sein. Zur Illustration nehmen wir an, dass T eine Konstante ist, die die absolute Laufzeit angibt, die wir uns leisten können. Verbessert man die Zeitschranke $t \in \tilde{\mathcal{O}}(2^n)$ zur Schranke $\hat{t} \in \tilde{\mathcal{O}}(c^n)$, wobei $1 < c < 2$ ist, dann kann die maximale Eingabegröße $n_{\hat{t}}$ mit $\hat{t}(n_{\hat{t}}) \leq T$ substanziell größer sein als die maximale Eingabegröße n_t mit $t(n_t) \leq T$.

Kann man zum Beispiel die triviale $\tilde{\mathcal{O}}(2^n)$ -Schranke des „naiven“ deterministischen Algorithmus für 3-SAT schlagen und zu einer $\tilde{\mathcal{O}}(c^n)$ -Schranke für die Konstante $c = \sqrt{2} \approx 1.4142$ verbessern, so kann man wegen $\tilde{\mathcal{O}}(\sqrt{2}^{2n}) = \tilde{\mathcal{O}}(2^n)$ in derselben Zeit Eingaben doppelter Größe bearbeiten. Das Verdoppeln der Eingabegröße, für die ein Algorithmus noch in annehmbarer Zeit läuft, ist nicht nur von theoretischer Bedeutung, sondern auch in der Praxis sehr wichtig. Ob ein Ingenieur durch eine neue Technologie in der Lage ist, eine doppelt so lange Brücke wie zuvor zu bauen, oder ob ein Architekt mittels neuer Materialien doppelt so hohe Gebäude wie zuvor entwerfen kann, stellt einen gravierenden Unterschied dar, und dies gilt ebenso für einen Software-Entwickler, dessen neuer Algorithmus doppelt so große Eingaben wie zuvor in vertretbarem Zeitaufwand bearbeiten kann.

Tabelle 6.1 gibt einen Überblick über ausgewählte Algorithmen für das Erfüllbarkeitsproblem und ihre Laufzeiten. Hier werden sowohl deterministische als auch randomisierte Algorithmen betrachtet, und die oberen Schranken für k -SAT sind für die Werte $k \in \{3, 4, 5, 6\}$ angegeben.

Tabelle 6.1. Laufzeiten ausgewählter Algorithmen für das Erfüllbarkeitsproblem

Algorithmus / Autoren	Typ	3-SAT	4-SAT	5-SAT	6-SAT
Backtracking	det.	$\tilde{\mathcal{O}}(1.913^n)$	$\tilde{\mathcal{O}}(1.968^n)$	$\tilde{\mathcal{O}}(1.987^n)$	$\tilde{\mathcal{O}}(1.995^n)$
Monien und Speckenmeyer [MS85]	det.	$\tilde{\mathcal{O}}(1.618^n)$	$\tilde{\mathcal{O}}(1.839^n)$	$\tilde{\mathcal{O}}(1.928^n)$	$\tilde{\mathcal{O}}(1.966^n)$
Dantsin et al. [DGH ⁺ 02]	det.	$\tilde{\mathcal{O}}(1.481^n)$	$\tilde{\mathcal{O}}(1.6^n)$	$\tilde{\mathcal{O}}(1.667^n)$	$\tilde{\mathcal{O}}(1.714^n)$
Brueggemann und Kern [BK04]	det.	$\tilde{\mathcal{O}}(1.473^n)$	—	—	—
Paturi et al. [PPSZ98]	rand.	$\tilde{\mathcal{O}}(1.362^n)$	$\tilde{\mathcal{O}}(1.476^n)$	$\tilde{\mathcal{O}}(1.569^n)$	$\tilde{\mathcal{O}}(1.637^n)$
Schöning [Sch99]	rand.	$\tilde{\mathcal{O}}(1.334^n)$	$\tilde{\mathcal{O}}(1.5^n)$	$\tilde{\mathcal{O}}(1.6^n)$	$\tilde{\mathcal{O}}(1.667^n)$
Iwama und Tamaki [IT03]	rand.	$\tilde{\mathcal{O}}(1.324^n)$	$\tilde{\mathcal{O}}(1.474^n)$	—	—
Rolf [Rol05]	rand.	$\tilde{\mathcal{O}}(1.32216^n)$	—	—	—

6.1.1 Deterministische Zeitkomplexität

In diesem Abschnitt wird ein deterministischer Algorithmus für 3-SAT präsentiert, der nach dem algorithmischen Prinzip „*Backtracking*“ arbeitet. Diese Algorithmentwurfstechnik ist für Probleme geeignet, deren Lösungen aus n Komponenten bestehen, so dass es für jede Komponente mehr als eine Wahlmöglichkeit gibt. Beispielsweise besteht eine Lösung einer 3-SAT-Instanz φ mit n Variablen aus den n Wahrheitswerten einer erfüllenden Belegung, und für jeden solchen Wahrheitswert gibt es zwei Wahlmöglichkeiten: *wahr* (durch 1 dargestellt) oder *falsch* (durch 0 dargestellt).

Die Idee ist nun, ausgehend von der leeren Lösung, welche keiner Variablen einen Wert zuweist, rekursiv immer größere Teillösungen zu konstruieren, Schritt für Schritt, bis schließlich eine vollständige Lösung gefunden ist, falls eine solche existiert. Im Falle von 3-SAT sind Teillösungen partielle Belegungen, die also nur einige der Variablen in der gegebenen Formel mit Wahrheitswerten belegen (vergleiche dazu den Begriff der partiellen Permutation in Beispiel 5.26). In jedem Rekursionsschritt dieser Prozedur werden die bisher konstruierten partiellen Belegungen um einen Wahrheitswert erweitert, der einer neuen, noch unbelegten Variablen zugewiesen wird.

Die Knoten im resultierenden Rekursionsbaum entsprechen den (verschachtelten) rekursiven Aufrufen der Prozedur und sind mit den bisher konstruierten partiellen Belegungen markiert. Insbesondere entspricht die Wurzel dem ersten rekursiven Aufruf und ist mit der leeren Belegung markiert. Die inneren Knoten des Rekursionsbaumes entsprechen den weiteren rekursiven Aufrufen. Ein Knoten \tilde{v} im Rekursionsbaum ist genau dann ein Kind eines Knoten v , wenn \tilde{v} innerhalb der durch v ausgelösten Berechnung aufgerufen wird. Schließlich terminiert der Algorithmus auf der Blattebene ohne weitere rekursive Aufrufe, und einige der Blätter sind mit vollständigen erfüllenden Belegungen markiert, sofern welche existieren.

Andere Blätter stellen „tote“ Zweige dar, weshalb die Berechnung hier erfolglos abgebrochen werden kann. Wird nämlich während der Berechnung entdeckt, dass der aktuelle Zweig im Rekursionsbaum „tot“ ist, d.h., dass die bisher konstruierte partielle Lösung unmöglich zu einer vollständigen Lösung der gegebenen 3-SAT-Instanz fortgesetzt werden kann, dann wird diese Rekursion abgebrochen und der gesamte Teilbaum darunter abgeschnitten, und die Prozedur zieht sich auf die darüberliegende Rekursionsstufe zurück (auf englisch als „*backtracking*“ bezeichnet) und versucht, eine andere, aussichtsreichere Erweiterung der zuvor konstruierten Teillösung zu finden. Durch das Abschneiden großer „toter“ Teile des Rekursionsbaumes, die nicht durchlaufen werden müssen, kann Rechenzeit gespart werden.

Abbildung 6.1 zeigt den Algorithmus BACKTRACKING-SAT. Bei Eingabe einer booleschen Formel φ und einer partiellen Belegung β , die einigen der Variablen von φ Wahrheitswerte zuweist, gibt BACKTRACKING-SAT einen booleschen Wert zurück: nämlich 1, falls die partielle Belegung β zu einer vollständigen Belegung aller Variablen von φ fortgesetzt werden kann, und andernfalls den Wert 0. Hier werden partielle Belegungen als Wörter der Länge höchstens n über dem Alphabet $\{0, 1\}$ aufgefasst.

```

BACKTRACKING-SAT( $\varphi, \beta$ ) {
    if ( $\beta$  belegt alle Variablen von  $\varphi$ ) return  $\varphi(\beta)$ ;
    else if ( $\beta$  macht eine Klausel von  $\varphi$  falsch) return 0;           // „toter“ Zweig
    else if (BACKTRACKING-SAT( $\varphi, \beta 0$ ) return 1;
              else return BACKTRACKING-SAT( $\varphi, \beta 1$ ));
}

```

Abb. 6.1. Backtracking-Algorithmus für 3-SAT

Der erste Aufruf des Algorithmus ist $\text{BACKTRACKING-SAT}(\varphi, \varepsilon)$, wobei ε die leere Belegung ist. Stellt der Algorithmus fest, dass die bisher konstruierte partielle Belegung β eine der Klauseln von φ falsch macht, so kann β nicht zu einer erfüllenden Belegung von φ fortgesetzt werden. Folglich ist dieser Zweig im Rekursionsbaum „tot“ und der Teilbaum darunter kann gefahrlos abgeschnitten werden. Siehe auch Übung 6.1.

Für die Laufzeitabschätzung von BACKTRACKING-SAT ist zunächst zu bemerken, dass der Algorithmus in Abbildung 6.1 so spezifiziert werden kann, dass er die Variablen in einer „intelligenten“ Reihenfolge wählt, die die Anzahl der zur Auswertung der Variablen in einer jeden Klausel erforderlichen Schritte minimiert. Betrachte eine beliebige feste Klausel C_j der gegebenen Formel φ . Jede erfüllende Belegung β von φ weist insbesondere den drei in C_j vorkommenden Variablen Wahrheitswerte zu. Von den $2^3 = 8$ Möglichkeiten, diese Variablen mit 0 oder 1 zu belegen, kann eine definitiv ausgeschlossen werden: die Belegung, die C_j falsch macht. Der entsprechende Knoten im Rekursionsbaum von $\text{BACKTRACKING-SAT}(\varphi, \beta)$ führt also zu einem „toten“ Zweig, und der darunterliegende Teilbaum kann abgeschnitten werden. Abhängig von der Struktur von φ könnte es weitere „tote“ Zweige geben, deren Teilbäume man abschneiden dürfte. Da wir hier aber eine obere Schranke im schlechtesten Fall finden wollen, vernachlässigen wir diese weiteren „toten“ Teilbäume. Somit erhalten wir

$$\tilde{\mathcal{O}}\left((2^3 - 1)^{n/3}\right) = \tilde{\mathcal{O}}(\sqrt[3]{7}^n) \approx \tilde{\mathcal{O}}(1.9129^n)$$

als eine obere Schranke für BACKTRACKING-SAT im schlechtesten Fall. Diese Schranke verbessert die $\tilde{\mathcal{O}}(2^n)$ -Schranke des „naiven“ Algorithmus für 3-SAT leicht.

Die deterministische Zeitkomplexität von 3-SAT kann noch weiter verbessert werden. Beispielsweise liefert der Teile-und-Herrsche-Algorithmus von Monien und Speckenmeyer [MS85] eine obere Schranke von $\tilde{\mathcal{O}}(1.618^n)$. Mit einer lokalen Suchtechnik erreichten Dantsin et al. [DGH⁺02] eine $\tilde{\mathcal{O}}(1.481^n)$ -Schranke, welche von Brueggemann und Kern [BK04] noch leicht zu $\tilde{\mathcal{O}}(1.473^n)$ verbessert wurde; siehe Tabelle 6.1.

6.1.2 Probabilistische Zeitkomplexität

Nun wenden wir uns einem randomisierten Algorithmus für 3-SAT zu, der von Schöning vorgeschlagen wurde. Sein „Random-Walk“-Algorithmus, der auf einer

„eingeschränkten lokalen Suche mit Neustart“ beruht, schlägt alle bekannten deterministischen Algorithmen für 3-SAT, auch wenn es inzwischen noch effizientere randomisierte Algorithmen für dieses Problem gibt; siehe Abschnitt 6.7.

Eine Irrfahrt (englisch als „*random walk*“ bezeichnet) kann auf einer Vielzahl von Strukturen stattfinden; etwa im Euklidischen Raum, auf einem endlichen oder unendlichen Gitter oder auf einem endlichen oder unendlichen Graphen. Hier betrachten wir Irrfahrten auf einem gerichteten endlichen Graphen, der als der Überführungsgraph eines stochastischen Automaten aufgefasst werden kann. Ein stochastischer Automat ist ein spezieller endlicher Automat, siehe die Definitionen 2.11 und 2.12 in Abschnitt 2.2.

Zur Erinnerung: Die Kanten im Überführungsgraphen eines endlichen Automaten sind mit den Symbolen eines gegebenen Alphabets markiert. Die Kanten eines stochastischen Automaten \mathcal{S} sind zusätzlich mit reellen Zahlen zwischen 0 und 1 markiert, die Wahrscheinlichkeiten darstellen. Die Markierung einer Kante von x nach y mit p_{xy} , wobei $0 \leq p_{xy} \leq 1$, zeigt an, dass der Übergang vom Zustand x zum Zustand y in \mathcal{S} mit der Wahrscheinlichkeit p_{xy} erfolgt. Die Wahrscheinlichkeiten aller Kanten, die aus irgendeinem Knoten von \mathcal{S} auslaufen, summieren sich zu eins. In der Sprache der Wahrscheinlichkeitstheorie heißt der Prozess der zufälligen Bewegung von Zustand zu Zustand gemäß der angegebenen Wahrscheinlichkeiten von \mathcal{S} eine Markoff-Kette. Endzustände, von denen aus es keinen Übergang zu einem anderen Zustand mit nicht verschwindender Wahrscheinlichkeit gibt, heißen dann *absorbierende Zustände*. Genau wie endliche Automaten kann ein stochastischer Automat \mathcal{S} zur Erkennung von Wörtern und zur Akzeptierung von Sprachen verwendet werden, wenn auch natürlich nur mit einer gewissen Wahrscheinlichkeit.

Hier sind wir jedoch nicht am Akzeptieren von Sprachen durch stochastische Automaten interessiert. Stattdessen verwenden wir sie, um die Berechnung des in Abbildung 6.2 dargestellten randomisierten Algorithmus RANDOM-SAT zu illustrieren. Bei Eingabe einer booleschen Formel φ mit n Variablen versucht RANDOM-SAT, eine erfüllende Belegung von φ zu finden, falls eine existiert. Die Berechnung von RANDOM-SAT(φ) kann man sich als eine Irrfahrt auf dem Überführungsgraphen eines stochastischen Automaten \mathcal{S} vorstellen, wie sie in Abbildung 6.3 gezeigt wird. Die Kanten sind hier allerdings nicht mit den Symbolen eines Alphabets markiert, sondern lediglich durch die entsprechenden Übergangswahrscheinlichkeiten.

Bei Eingabe φ wählt RANDOM-SAT zunächst einen zufälligen Anfangsvektor β , dessen sämtliche Bits unabhängig und gleichverteilt sind. Das heißt, jedes Bit von β nimmt den Wert 0 oder 1 mit Wahrscheinlichkeit $1/2$ an.

Ist φ nicht erfüllbar, so kann RANDOM-SAT(φ) niemals eine erfüllende Belegung von φ ausgeben; also macht der Algorithmus in diesem Fall keinen Fehler.

Sei nun angenommen, dass φ erfüllbar ist. Sei α eine beliebige, fest gewählte erfüllende Belegung von φ . (RANDOM-SAT muss α nicht kennen oder bestimmen; α wird lediglich zur Erklärung der Arbeitsweise von RANDOM-SAT und für die Analyse benötigt.) Sei \mathbf{X} eine Zufallsvariable, die die *Hammingdistanz zwischen α und β* beschreibt, also die Anzahl der Bits, in denen sich α und β unterscheiden. Offenbar kann \mathbf{X} die Werte $j \in \{0, 1, \dots, n\}$ annehmen und ist gemäß der Binomial-

```

RANDOM-SAT( $\varphi$ ) {
    for ( $i = 1, 2, \dots, \lceil(4/3)^n\rceil$ ) { //  $n$  ist die Anzahl der Variablen in  $\varphi$ 
        Wähle zufällig eine Belegung  $\beta \in \{0, 1\}^n$  unter Gleichverteilung;
        for ( $j = 1, 2, \dots, n$ ) {
            if ( $\varphi(\beta) = 1$ ) return die erfüllende Belegung  $\beta$  von  $\varphi$  und halte;
            else {
                Wähle eine Klausel  $C = (x \vee y \vee z)$  mit  $C(\beta) = 0$ ;
                Wähle zufällig ein Literal  $\ell \in \{x, y, z\}$  unter Gleichverteilung;
                Bestimme das Bit  $\beta_\ell \in \{0, 1\}$  in  $\beta$ , das  $\ell$  belegt;
                Modifiziere  $\beta_\ell$  zu  $1 - \beta_\ell$  in  $\beta$ ;
            }
        }
    }
    return „ $\varphi$  ist nicht erfüllbar“;
}

```

Abb. 6.2. Algorithmus RANDOM-SAT

Verteilung mit den Parametern n und $1/2$ verteilt. Das heißt, die Wahrscheinlichkeit für das Ereignis $\mathbf{X} = j$ ist $\binom{n}{j} 2^{-n}$.

Den oben beschriebenen Anfangsschritt von $\text{RANDOM-SAT}(\varphi)$ kann man sich nun als den ersten Schritt der Irrfahrt auf dem Übergangsgraphen von \mathcal{S} vorstellen, der gemäß der Verteilung von \mathbf{X} vom Anfangszustand s zu einem der Zustände $j \in \{0, 1, \dots, n\}$ führt. Ist man im Zustand j , so haben die zufällig gewählte Anfangsbelegung β und die feste erfüllende Belegung α die Hammingdistanz j . Abbildung 6.3 zeigt den Übergangsgraphen von \mathcal{S} für $n = 6$.

Danach überprüft $\text{RANDOM-SAT}(\varphi)$, ob die Anfangsbelegung β die Formel φ bereits erfüllt, und falls ja, akzeptiert der Algorithmus. Andernfalls, wenn also β die Formel φ nicht erfüllt, muss es eine von β nicht erfüllte Klausel in φ geben. $\text{RANDOM-SAT}(\varphi)$ wählt eine beliebige solche Klausel und wählt in dieser Klausel zufällig ein Literal unter Gleichverteilung. Das Bit in der aktuellen Belegung β , das das gewählte Literal belegt, wird dann in der Hoffnung umgedreht, dass die so modifizierte Belegung der erfüllenden Belegung α „näher“ ist. „Näher“ zu sein, bedeutet, dass die Hammingdistanz zu α kleiner ist. Das Umdrehen eines Bits β_ℓ zu $1 - \beta_\ell$ in der aktuellen Belegung kann man sich, wie oben beschrieben, als einen Schritt nach links oder nach rechts in der Irrfahrt auf dem Übergangsgraphen von \mathcal{S} vorstellen, bei dem nach sich also vom Zustand $j > 0$ entweder in den Zustand $j - 1$ oder in den Zustand $j + 1$ bewegt. Dabei können natürlich nur Zustände kleiner als oder gleich n erreicht werden.

In jeder Klausel gibt es drei Literale. Die feste Belegung α erfüllt φ ; folglich erfüllt sie wenigstens ein Literal in jeder Klausel. Betrachten wir in jeder Klausel *genau* eines der von α erfüllten Literale, etwa ℓ , dann macht $\text{RANDOM-SAT}(\varphi)$ genau dann einen Schritt nach links, wenn ℓ von $\text{RANDOM-SAT}(\varphi)$ gewählt wurde. Somit erfolgt ein Übergang vom Zustand $j > 0$ zum Zustand $j - 1$ mit der Wahr-

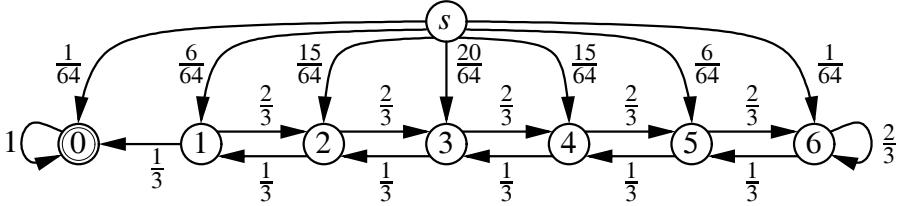


Abb. 6.3. Übergangsgraph eines stochastischen Automaten für RANDOM-SAT

scheinlichkeit $1/3$ und ein Übergang vom Zustand $j > 0$ zum Zustand $j + 1$ mit der Wahrscheinlichkeit $2/3$.

Dieser Prozess wird so oft wiederholt, wie $j \neq 0$ ist, und höchstens n -mal. Sobald der Zustand $j = 0$ erreicht ist, haben die aktuelle Belegung β und die feste Belegung α die Hammingdistanz 0. Also hat RANDOM-SAT(φ) eine erfüllende Belegung von φ gefunden und akzeptiert seine Eingabe. Natürlich könnte man auch in einem Zustand $j \neq 0$ auf eine (von α verschiedene) erfüllende Belegung treffen. Dies würde jedoch die Akzeptierungswahrscheinlichkeit nur erhöhen, weshalb wir diese Möglichkeit hier vernachlässigen wollen.

Wurde dieser Prozess n -mal ohne Erfolg ausgeführt, dann war die Anfangsbelegung β offenbar so unglücklich gewählt, dass RANDOM-SAT(φ) sie nun doch fallen lässt und den ganzen Prozess von Anfang an neu startet, wobei eine neue Anfangsbelegung gewählt wird. Diese ganze Prozedur wird höchstens t -mal wiederholt, wobei $t = \lceil (4/3)^n \rceil$ ist. Wenn RANDOM-SAT(φ) auch nach t Versuchen noch erfolglos ist, wird die Eingabe verworfen.

Nun soll grob skizziert werden, wie man – sofern φ erfüllbar ist – die Akzeptierungswahrscheinlichkeit und die Laufzeit von RANDOM-SAT(φ) abschätzen kann, wobei viele technische Details vernachlässigt werden und das Hauptaugenmerk auf die wichtigsten Ideen gerichtet wird. Zur Vereinfachung nehmen wir an, dass n durch 3 teilbar ist.

Die Wahrscheinlichkeit, einen Schritt nach rechts zu gehen, ist größer als die Wahrscheinlichkeit, einen Schritt nach links – zum Endzustand 0 hin – zu gehen. Man könnte sich davon zu der Annahme verleiten lassen, dass die Akzeptierungswahrscheinlichkeit von RANDOM-SAT(φ) ziemlich klein ist. Jedoch sollte man die Chance, vom Anfangszustand s unmittelbar in einen Zustand j nahe 0 zu springen, nicht unterschätzen. Je näher am Zustand 0 man die Irrfahrt beginnt, desto höher ist die Wahrscheinlichkeit dafür, schließlich diesen Zustand 0 während der verbleibenden n Schritte zu erreichen.

Sei p_i die Wahrscheinlichkeit für das Ereignis, dass RANDOM-SAT(φ) den Zustand 0 in höchstens n Schritten nach dem ersten Schritt erreicht, unter der Bedingung, dass mit diesem ersten Schritt ein Zustand $i \leq n/3$ erreicht wurde. Erreicht man zum Beispiel den Zustand $n/3$ mit dem ersten Schritt und geht dann nicht mehr als $n/3$ Schritte nach rechts, so kann man den Endzustand 0 immer noch mit insgesamt höchstens n Schritten erreichen. Geht man vom Zustand $n/3$ aus mehr als $n/3$

Schritte nach rechts, dann kann der Endzustand jedoch nicht innerhalb von n Schritten erreicht werden. Allgemein gilt, dass ausgehend vom Zustand i nach dem ersten Schritt nicht mehr als $(n - i)/2$ Schritte nach rechts gemacht werden dürfen, will man erfolgreich zur 0 kommen. Da ein Schritt nach rechts mit Wahrscheinlichkeit $2/3$ und ein Schritt nach links mit Wahrscheinlichkeit $1/3$ gemacht wird, ergibt sich:

$$p_i = \binom{n}{\frac{n-i}{2}} \left(\frac{2}{3}\right)^{\frac{n-i}{2}} \left(\frac{1}{3}\right)^{n-\frac{n-i}{2}}. \quad (6.1)$$

Sei q_i die Wahrscheinlichkeit für das Ereignis, dass $\text{RANDOM-SAT}(\varphi)$ mit dem ersten Schritt einen Zustand $i \leq n/3$ erreicht. Offenbar gilt:

$$q_i = \binom{n}{i} \cdot 2^{-n}. \quad (6.2)$$

Sei p die Wahrscheinlichkeit für das Ereignis, dass $\text{RANDOM-SAT}(\varphi)$ während der inneren for-Schleife den Endzustand 0 erreicht. Natürlich kann dieses Ereignis auch dann eintreten, wenn man von einem Zustand $j > n/3$ aus startet. Folglich ist

$$p \geq \sum_{i=0}^{n/3} p_i \cdot q_i.$$

Diese Summe kann mittels der in Definition 4.26 von Abschnitt 4.3.2 eingeführten Entropiefunktion geeignet approximiert werden. Weiterhin können die Binomialkoeffizienten in (6.1) und (6.2) mittels der Stirling-Formel geeignet abgeschätzt werden. Ohne hier in die Details zu gehen, erhält man eine untere Schranke von mindestens $(3/4)^n$ für p , wobei konstante Faktoren vernachlässigt werden.

Die Fehlerwahrscheinlichkeit von $\text{RANDOM-SAT}(\varphi)$ kann dann hinreichend klein gemacht werden, indem man $t = \lceil (4/3)^n \rceil$ unabhängige Versuche durchführt, die jeweils von einer neuen Anfangsbelegung β ausgehen. Für jeden Versuch ist die Akzeptierungswahrscheinlichkeit mindestens $(3/4)^n$, und somit ist der Fehler durch $1 - (3/4)^n$ beschränkt. Da die Versuche unabhängig sind, multiplizieren sich diese Fehlerwahrscheinlichkeiten und man erhält insgesamt einen Fehler von höchstens

$$(1 - (3/4)^n)^t \leq e^{-1}.$$

Also ist die Akzeptierungswahrscheinlichkeit von $\text{RANDOM-SAT}(\varphi)$ insgesamt mindestens $1 - 1/e \approx 0.632$, falls φ erfüllbar ist, und andernfalls (wenn φ unerfüllbar ist) macht $\text{RANDOM-SAT}(\varphi)$ überhaupt keinen Fehler. Der Hinweis für Übung 6.2 erklärt die spezielle Wahl dieses Wertes von t , aus welchem sich auch die Laufzeit $\tilde{\mathcal{O}}(1.334^n)$ von RANDOM-SAT ergibt, wobei in n polynomielle Faktoren wie üblich vernachlässigt werden. Als Daumenregel gilt: Um eine hinreichend kleine Fehlerwahrscheinlichkeit zu erzielen, sollte die Anzahl der Wiederholungen etwa reziprok zur Erfolgswahrscheinlichkeit eines einzelnen Versuches sein.

6.2 Probabilistische Polynomialzeit-Klassen

6.2.1 PP, RP und ZPP: Monte-Carlo- und Las-Vegas-Algorithmen

Deterministische und nichtdeterministische Turingmaschinen wurden in Kapitel 3, alternierende Turingmaschinen in Kapitel 5 eingeführt. In diesem Kapitel beschäftigen wir uns mit noch einem weiteren Maschinenmodell, der probabilistischen Turingmaschine, welche ein weiteres Berechnungsparadigma verkörpert: *Randomisierung*. Die probabilistische Turingmaschine formalisiert den Begriff des randomisierten Algorithmus. Ein solcher Algorithmus ist in der Lage, „Münzen zu werfen“ und seine Berechnungen unter Verwendung dieser Zufallswahlen auszuführen. Ein Beispiel eines randomisierten Algorithmus wurde in Abschnitt 6.1.2 gegeben. Randomisierung ermöglicht oft effizientere Algorithmen für ein gegebenes Problem. Doch dafür ist ein Preis zu zahlen: Randomisierte Algorithmen können Fehler machen.

Das Modell der probabilistischen Turingmaschine kann durch geeignete Abwandlungen der nichtdeterministischen Turingmaschine (NTM) beschrieben werden. Syntaktisch ist eine *probabilistische Turingmaschine* einfach eine NTM N , wobei wir per Konvention vereinbaren, dass der nichtdeterministische Verzweigungsgrad von N höchstens zwei ist. Außerdem ist es oft nützlich, zu verlangen, dass NTMs *normalisiert* sein mögen, d.h., für jede Eingabe x wird verlangt, dass alle Berechnungspfade in $N(x)$ dieselbe Anzahl nichtdeterministischer Verzweigungen haben und somit durch Binärwörter derselben Länge beschrieben werden können. Anders gesagt, lässt man einmal die deterministischen Schritte solcher Maschinen außer Acht, so haben normalisierte Maschinen stets einen vollen binären Berechnungsbaum.

Die Semantik von probabilistischen Turingmaschinen definiert man, indem man ein Akzeptierungsverhalten von NTMs spezifiziert, das für Randomisierung geeignet ist. Zu diesem Zweck definieren wir für eine gegebene NTM N und eine Eingabe x ein Wahrscheinlichkeitsmaß μ_T auf der Menge der Berechnungspfade im Baum $T = N(x)$, dessen Knoten die Konfigurationen von N bei Eingabe x darstellen. Es gibt eine injektive Korrespondenz zwischen den Berechnungspfaden und den Blättern des Baumes T .

Definition 6.1 (Semantik probabilistischer Turingmaschinen). *Betrachte für eine NTM N und eine Eingabe x einen beliebigen Teilbaum T des Berechnungsbaums $N(x)$, so dass die Wurzel von T die Wurzel von $N(x)$ ist. Das Wahrscheinlichkeitsmaß μ_T auf der Menge der Blätter von T ist induktiv wie folgt definiert:*

- Besteht T nur aus der Wurzel r von $N(x)$ (d.h., r ist die Startkonfiguration von $N(x)$), dann setze $\mu_T(r) = 1$.
- Solange $T \neq N(x)$ ist, fixiere ein Blatt ℓ von T , das kein Blatt von $N(x)$ ist, und betrachte den neuen Teilbaum T_ℓ von $N(x)$, den man aus T durch Hinzufügen der unmittelbaren Folgekonfiguration(en) von ℓ erhält. Definiere nun, ausgehend vom Wahrscheinlichkeitsmaß μ_T auf den Blättern von T , das Wahrscheinlichkeitsmaß μ_{T_ℓ} auf den Blättern von T_ℓ wie folgt:

$$\mu_{T_\ell}(c) = \begin{cases} \mu_T(\ell)/2 & \text{falls } c \text{ eine von zwei Folgekonfigurationen von } \ell \text{ ist} \\ \mu_T(\ell) & \text{falls } c \text{ die einzige Folgekonfiguration von } \ell \text{ ist} \\ \mu_T(c) & \text{falls } c \text{ keine Folgekonfiguration von } \ell \text{ ist.} \end{cases}$$

Wir werden uns ausschließlich mit probabilistischen Turingmaschinen befassen, die in Polynomialzeit arbeiten. Jeder Berechnungspfad einer gegebenen NPTM N bei einer Eingabe x wird durch ein Binärwort α der Länge $p(|x|)$ dargestellt, für ein $p \in \mathbb{P}\text{ol}$. Dabei entspricht das i -te Bit von α der i -ten nichtdeterministischen Verzweigung von $N(x)$ entlang α . Führt α zu einem Blatt ℓ von $T = N(x)$, so schreiben wir $\mu_T(\alpha) = \mu_T(\ell)$. Man sieht leicht, dass μ_T tatsächlich ein Wahrscheinlichkeitsmaß ist, denn für jeden endlichen Baum T gilt:

$$\sum_{\alpha \text{ ist ein Pfad von } T} \mu_T(\alpha) = 1.$$

Jede Teilmenge der Pfade von T ist ein Ereignis. Die Wahrscheinlichkeit dafür, dass ein Ereignis E bezüglich μ_T eintritt, ist bestimmt durch:

$$\Pr(E) = \sum_{\alpha \in E} \mu_T(\alpha) = \sum_{\alpha \in E} 2^{-|\alpha|}.$$

Mit Hilfe dieser Notation können wir nun die ersten beiden probabilistischen Komplexitätssklassen definieren: die Klasse PP, „Probabilistic Polynomial Time“, und die Klasse RP, „Random Polynomial Time“.

Definition 6.2 (Probabilistic Polynomial Time und Random Polynomial Time).

1. Probabilistic Polynomial Time ist definiert durch

$$\text{PP} = \left\{ A \left| \begin{array}{l} \text{es gibt eine NPTM } M, \text{ so dass für jede Eingabe } x \text{ gilt:} \\ x \in A \iff \Pr(\{\alpha \mid M \text{ akzeptiert } x \text{ auf Pfad } \alpha\}) \geq 1/2 \end{array} \right. \right\}.$$

2. Random Polynomial Time ist definiert durch

$$\text{RP} = \left\{ A \left| \begin{array}{l} \text{es gibt eine NPTM } M, \text{ so dass für jede Eingabe } x \text{ gilt:} \\ x \in A \implies \Pr(\{\alpha \mid M \text{ akzeptiert } x \text{ auf Pfad } \alpha\}) \geq 1/2; \\ x \notin A \implies \Pr(\{\alpha \mid M \text{ akzeptiert } x \text{ auf Pfad } \alpha\}) = 0 \end{array} \right. \right\}.$$

Anmerkung 6.3 (Normalisierte Turingmaschinen und Schwellwertberechnung).

1. Jede NPTM, deren Akzeptierungskriterium gemäß den Definitionen 6.1 und 6.2 auf Wahrscheinlichkeitsgewichten beruht, kann leicht normalisiert werden: Setze einfach jeden Berechnungspfad durch Anhängen eines vollen binären Teilbaums bis zu einer vorgegebenen polynomiellen Länge fort und akzeptiere auf einem jeden so erzeugten Pfad genau dann, wenn der ursprüngliche Pfad akzeptiert hat. Die so modifizierte normalisierte Maschine hat dieselbe Akzeptierungs-wahrscheinlichkeit wie die ursprüngliche Maschine.

2. Das Akzeptierungskriterium der probabilistischen Turingmaschinen für PP und RP aus Definition 6.2 wird durch das Wahrscheinlichkeitsgewicht der akzeptierenden Pfade gemäß Definition 6.1 bestimmt. Alternativ dazu kann das Akzeptierungskriterium für PP und RP über die *Anzahl* der akzeptierenden Pfade von NPTMs festgelegt werden; siehe auch die Definitionen 6.31 und 6.34. Definiere für eine NPTM M die Funktion $\text{acc}_M : \Sigma^* \rightarrow \mathbb{N}$ durch

$$\text{acc}_M(x) = \|\{\alpha \mid M \text{ akzeptiert } x \text{ auf Pfad } \alpha\}\|.$$

Dann kann man die folgenden Charakterisierungen von PP und RP durch so genannte Schwellwertberechnungen (englisch „*threshold computations*“) zeigen:

- a) A ist genau dann in PP, wenn es eine normalisierte NPTM M und ein Polynom p gibt, so dass für jedes x gilt:

$$x \in A \iff \text{acc}_M(x) \geq 2^{p(|x|)-1}.$$

- b) A ist genau dann in RP, wenn es eine normalisierte NPTM M und ein Polynom p gibt, so dass für jedes x gilt:

$$\begin{aligned} x \in A &\implies \text{acc}_M(x) \geq 2^{p(|x|)-1}; \\ x \notin A &\implies \text{acc}_M(x) = 0. \end{aligned}$$

Diese Äquivalenz der Interpretationen über das Wahrscheinlichkeitsgewicht bzw. über die Anzahl acc_M akzeptierender Pfade setzt normalisierte Maschinen voraus. Für PP könnte diese Normalisierungsforderung fallen gelassen werden; siehe Problem 6.1(a). Für RP jedoch scheint es wesentlich zu sein, normalisierte Maschinen zu verlangen; siehe Problem 6.1(b).

Anders als PP ist die Klasse RP eine so genannte „*Promise-Klasse*“. Komplexitätsklassen werden üblicherweise durch Maschinen repräsentiert (z.B. durch NPTMs), die die Klasse durch ihre Akzeptierungs- und Ablehnungskriterien definieren. In vielen Fällen gilt genau eines der beiden Kriterien für jede Eingabe. Beispielsweise akzeptieren PP-Maschinen, wenn mindestens die Hälfte ihrer Berechnungspfade akzeptieren, und andernfalls lehnen sie ab. Im Gegensatz dazu hat eine *Promise-Klasse* wie RP ein Ablehnungskriterium, das *restriktiver* ist als die logische Negation ihres Akzeptierungskriteriums,² was die Möglichkeit offen lässt, dass für einige Eingaben keines der beiden Kriterien anwendbar ist. Die Last, dieses Hindernis zu umgehen, wird den Maschinen aufgebürdet, die die Promise-Klasse repräsentieren: Jede Maschine „verspricht“ (englisch „*promises*“), dass für jede Eingabe genau eines der beiden Kriterien gilt. Zum Beispiel „versprechen“ RP-Maschinen

² Etwas sorgfältiger formuliert, haben *alle bekannten* Paare von Akzeptierungs- und Ablehnungskriterien für diese Klasse die Eigenschaft, dass das Ablehnungskriterium restriktiver als die logische Negation ihres Akzeptierungskriteriums ist. Zum Beispiel wird die Klasse RP_{path} in Problem 6.1 zwar über ein Ablehnungskriterium definiert, das restriktiver als die logische Negation des entsprechenden Akzeptierungskriteriums ist, aber dennoch ist bekannt, dass RP_{path} nichts anderes als NP ist. Somit ist RP_{path} *keine* Promise-Klasse.

gemäß Definition 6.2, dass sie niemals eine Akzeptierungswahrscheinlichkeit von $1/4$ haben (beachte jedoch Satz 6.6). Andere Beispiele von Promise-Klassen sind UP und FewP, definiert in Kapitel 3. Promise-Klassen scheinen ganz andere Eigenschaften zu haben als die Komplexitätsklassen, die keine Promise-Klassen sind. Zum Beispiel fehlen den Promise-Klassen RP, UP und FewP anscheinend vollständige Mengen; siehe auch Korollar 3.83.

RP- und coRP-Algorithmen haben beide eine einseitige Fehlerwahrscheinlichkeit. Solche Algorithmen heißen auch *Monte-Carlo-Algorithmen*. Sei L eine Menge in RP, und sei A ein RP-Algorithmus für L . Nach Definition macht A womöglich Fehler für Instanzen x in L , aber A „lügt“ nie für Instanzen x , die nicht in L liegen. Somit kann die Antwort „ja“ (d.h. ein akzeptierender Pfad) von A bei Eingabe x nur dann auftreten, wenn x in L ist, und ist daher stets korrekt, wohingegen die Antwort „nein“ (also ein ablehnender Pfad) sowohl fälschlich (falls nämlich x in L ist) als auch korrekt (falls nämlich x nicht in L ist) auftreten kann. Aus diesem Grund werden RP-Algorithmen manchmal „no-biased“ Monte-Carlo-Algorithmen genannt. Ebenso geben coRP-Algorithmen stets verlässliche „Nein“-Antworten, aber womöglich falsche „Ja“-Antworten. Daher werden coRP-Algorithmen manchmal „yes-biased“ Monte-Carlo-Algorithmen genannt. Anders als PP scheint die Klasse RP nicht unter Komplementbildung abgeschlossen zu sein (außer, natürlich, es stellt sich irgendwann z.B. die Gleichheit $RP = P$ heraus).

Satz 6.4. 1. $P \subseteq RP \subseteq NP \subseteq PP \subseteq PSPACE$.

2. PP ist unter Komplement abgeschlossen.

Beweis. 1. Die ersten beiden Inklusionen, $P \subseteq RP \subseteq NP$, folgen unmittelbar aus den Definitionen. Die Inklusion $PP \subseteq PSPACE$ kann ähnlich wie die Inklusion $NP \subseteq PSPACE$ aus Satz 3.27 bewiesen werden: Sei eine PP-Maschine M gegeben, die bei Eingabe x arbeitet. Die simulierende PSPACE-Maschine führt eine Tiefensuche durch den Berechnungsbaum von $M(x)$ aus. Statt jedoch wie im Beweis von $NP \subseteq PSPACE$ nach nur *einem* akzeptierenden Pfad zu suchen, zählt die PSPACE-Maschine nun *sämtliche* akzeptierende Pfade von $M(x)$, und sie akzeptiert die Eingabe genau dann, wenn diese Anzahl mindestens gleich der Hälfte aller Pfade ist.

Für die Inklusion $NP \subseteq PP$ sei A eine beliebige Menge in NP, und M sei eine gegebene NP-Maschine, die A akzeptiert. Wir nehmen an, dass M normalisiert ist, so dass also bei jeder Eingabe x der Berechnungsbaum $M(x)$ ein vollständiger Binärbaum der Tiefe $p(|x|)$ für ein $p \in \mathbb{N}$ ist. Konstruiere wie folgt eine neue NPTM N , die A im Sinne von PP akzeptiert. Bei Eingabe x verzweigt N nichtdeterministisch. Auf dem linken Zweig simuliert N die Berechnung von $M(x)$. Auf dem rechten Zweig erzeugt N einen vollständigen Binärbaum der Tiefe $p(|x|)$, akzeptiert dort auf $2^{p(|x|)} - 1$ Pfade und lehnt auf dem verbleibenden dieser Pfade ab.

Ist $x \in A$, so akzeptiert $M(x)$ auf mindestens einem ihrer $2^{p(|x|)}$ Pfade. Somit akzeptiert $N(x)$ auf mindestens $2^{p(|x|)}$ ihrer $2^{p(|x|)+1}$ Pfade, d.h., N akzeptiert x mit einer Wahrscheinlichkeit von mindestens ein halb.

Ist $x \notin A$, dann lehnen sämtliche $2^{p(|x|)}$ Pfade von $M(x)$ ab. Also akzeptiert $N(x)$ auf höchstens $2^{p(|x|)} - 1$ ihrer $2^{p(|x|)+1}$ Pfade, d.h., N akzeptiert x mit einer Wahrscheinlichkeit kleiner als ein halb. Es folgt, dass A in PP ist.

2. Der Beweis von $\text{PP} = \text{coPP}$ wird dem Leser als Übung 6.3 überlassen. \square

Der Akzeptanzschwellwert von $1/2$ in der Definition von RP ist willkürlich festgelegt. Andere Schwellwerte würden ebenso gut klappen und würden dieselbe Mengenklasse definieren. Wir werden nun sehen, dass der Akzeptanzschwellwert einer RP-Berechnung sehr klein sein kann: Es genügt, zu verlangen, dass die Akzeptierungswahrscheinlichkeit mindestens reziprok polynomiell in der Eingabegröße ist. Somit kann erreicht werden, dass die Fehlerwahrscheinlichkeit von RP-Algorithmen kleiner als $1/q(|x|)$ ist, für ein beliebiges fixiertes Polynom q .

Definition 6.5. Sei q ein Polynom. Definiere die Klasse

$$\text{RP}_q = \left\{ A \left| \begin{array}{l} \text{es gibt eine NPTM } M, \text{ so dass für jede Eingabe } x \text{ gilt:} \\ x \in A \implies \Pr(\{\alpha \mid M \text{ akzeptiert } x \text{ auf Pfad } \alpha\}) \geq 1/q(|x|); \\ x \notin A \implies \Pr(\{\alpha \mid M \text{ akzeptiert } x \text{ auf Pfad } \alpha\}) = 0 \end{array} \right. \right\}.$$

Satz 6.6. Sei q ein (nicht notwendig streng) monoton wachsendes Polynom, so dass $q(n) \geq 2$ für jedes n gilt. Dann ist $\text{RP}_q = \text{RP}$.

Beweis. Da $q(n) \geq 2$ für jedes n gilt, folgt die Inklusion $\text{RP} \subseteq \text{RP}_q$ aus der Definition.

Um umgekehrt $\text{RP}_q \subseteq \text{RP}$ zu beweisen, sei A eine beliebige Menge in RP_q , für das Polynom q , und M sei eine NPTM für A gemäß Definition 6.5. Konstruiere wie folgt eine NPTM N , die A im Sinne von RP akzeptiert. Bei einer Eingabe x der Länge n simuliert N sukzessive die Berechnung von $M(x)$ genau q -mal hintereinander, in $q = q(n)$ unabhängigen Versuchen. Also besteht jeder Pfad $\alpha = \alpha_1 \alpha_2 \cdots \alpha_q$ von $N(x)$ aus einer Folge von q Pfaden α_i von $M(x)$, und wir definieren, dass α genau dann akzeptiert, wenn mindestens einer seiner Teilstufen α_i akzeptiert.

Da q ein Polynom ist und M in Polynomialzeit arbeitet, ist auch die Rechenzeit von N durch ein Polynom beschränkt. Es bleibt zu zeigen, dass N die Zugehörigkeit von A zu RP bezeugt. Sei x das Eingabewort. Ist $x \notin A$, so akzeptiert kein Pfad von $M(x)$. Also hat auch $N(x)$ keine akzeptierenden Pfade und die Akzeptierungswahrscheinlichkeit ist null.

Wir schätzen nun die Fehlerwahrscheinlichkeit $E_N(x)$ von N für $x \in A$ ab, welche gegeben ist durch

$$E_N(x) = \Pr(\{\alpha \mid N \text{ lehnt } x \text{ auf dem Pfad } \alpha \text{ ab}\}).$$

Die Fehlerwahrscheinlichkeit von $M(x)$ ist nach oben durch $1 - 1/q$ beschränkt. Für jeden Pfad α von $N(x)$ werden alle Teilstufen α_i von α unabhängig gewählt. Daraus folgt für die Fehlerwahrscheinlichkeit von N :

$$E_N(x) < \left(1 - \frac{1}{q(n)}\right)^{q(n)} < \frac{1}{2}. \quad (6.3)$$

Die zweite der Ungleichungen von (6.3) ergibt sich aus $\lim_{k \rightarrow \infty} (1 + \frac{a}{k})^k = e^a$, wobei $e = 2.71828 \dots$ die Basis des natürlichen Logarithmus ist. Denn es gilt für $a = -1$, dass $(1 - 1/q(n))^{q(n)}$ nahe an e^{-1} ist, welches kleiner als ein halb ist. Folglich gilt:

$$\Pr(\{\alpha \mid N \text{ akzeptiert } x \text{ auf Pfad } \alpha\}) \geq \frac{1}{2},$$

womit der Beweis abgeschlossen ist. \square

Satz 6.6 hat eine einfache Folgerung, deren Beweis dem Leser als Übung 6.4 überlassen wird.

Korollar 6.7. RP ist unter Vereinigung und Durchschnitt abgeschlossen.

PP ist ebenfalls unter Vereinigung und Durchschnitt abgeschlossen, und sogar unter \leq_u^P -Reduktionen. Der Abschluss unter Komplement ist für PP zwar einfach zu sehen, der Beweis des Abschlusses unter Durchschnitt ist für diese Klasse jedoch alles andere als trivial; siehe Problem 6.2.

RP-Algorithmen haben einen einseitigen Fehler, und Satz 6.6 sagt, dass dieser Fehler sehr klein gemacht werden kann. Diese Eigenschaft ist ein klarer Vorteil von RP- gegenüber PP-Algorithmen. Wie oben erwähnt können RP-Algorithmen falsche „Nein“-Antworten geben, wohingegen coRP-Algorithmen stets verlässliche „Nein“-Antworten geben. Umgekehrt können coRP-Algorithmen falsche „Ja“-Antworten geben, doch RP-Algorithmen geben stets verlässliche „Ja“-Antworten. Die Klasse ZPP besteht aus den Problemen, die sich durch polynomialzeit-beschränkte randomisierte Algorithmen mit nullseitigem Fehler lösen lassen, wodurch die Vorteile der „yes-biased“ und der „no-biased“ Monte-Carlo-Algorithmen kombiniert werden.

Definition 6.8 (Zero-Error Probabilistic Polynomial Time). Definiere die Klasse Zero-error Probabilistic Polynomial Time durch $ZPP = RP \cap coRP$.

Genau wie RP ist die Klasse ZPP eine Promise-Klasse. ZPP-Algorithmen, welche man auch als *Las-Vegas-Algorithmen* bezeichnet, geben niemals eine falsche Antwort, wenn es auch passieren könnte, dass sie überhaupt keine brauchbare Antwort geben. Dies rechtfertigt das „zero-error“ in ihrem Namen. Ähnlich wie in Definition 5.74 kann ein ZPP-Algorithmus als eine NPTM M mit drei (Typen von) Endzuständen aufgefasst werden: einem akzeptierenden Zustand s_a , einem ablehnenden Zustand s_r und einem „weiß nicht“-Zustand $s_?$.

Tabelle 6.2. Eine ZPP-Berechnung

	Pfad α of $M(x)$	Pfad β von $N(x)$	Pfad $\langle \alpha, \beta \rangle$ von $(M \circ N)(x)$
$x \in A$	+	—	$(+, -) = s_a$
	—	—	$(-, -) = s_?$
$x \notin A$	—	+	$(-, +) = s_r$
	—	—	$(-, -) = s_?$

Sei A eine beliebige Sprache in ZPP, und seien M und N NPTMs, die bezeugen, dass $A \in RP$ bzw. $\overline{A} \in RP$ gilt. Betrachte die Maschine $M \circ N$, die folgendermaßen definiert ist: Bei Eingabe x simuliert $M \circ N$ erst $M(x)$ und dann $N(x)$. Somit hat jeder

Pfad von $(M \circ N)(x)$ die Form $\langle \alpha, \beta \rangle$, wobei α ein Pfad von $M(x)$ und β ein Pfad von $N(x)$ ist. Für die Pfade α und β bezeichnen wir Akzeptierung mit $+$ und Ablehnung mit $-$. Tabelle 6.2 zeigt alle Möglichkeiten für die Pfade α und β und wie $M \circ N$ diese Fälle behandelt, indem sie ihre Endzustände s_a , s_r und $s_?$ jedem möglichen Paar $\langle \alpha, \beta \rangle$ zuweist. Daraus ergibt sich das folgende Korollar.

Korollar 6.9. *A ist genau dann in ZPP, wenn es eine NPTM M mit drei Typen von Endzuständen gibt (nämlich mit akzeptierenden und ablehnenden Zuständen und mit einem „weiß nicht“-Zustand), so dass für jedes x gilt:*

$$\begin{aligned} x \in A &\implies \Pr(\{\alpha \mid M \text{ akzeptiert } x \text{ auf Pfad } \alpha\}) \geq 1/2 \text{ und} \\ &\quad \Pr(\{\alpha \mid M \text{ lehnt } x \text{ auf Pfad } \alpha \text{ ab}\}) = 0; \\ x \notin A &\implies \Pr(\{\alpha \mid M \text{ lehnt } x \text{ auf Pfad } \alpha \text{ ab}\}) \geq 1/2 \text{ und} \\ &\quad \Pr(\{\alpha \mid M \text{ akzeptiert } x \text{ auf Pfad } \alpha\}) = 0. \end{aligned}$$

Zum Schluss dieses Abschnitts werden zwei Probleme vorgestellt, die vollständig für PP sind.

Definition 6.10 (Majority-SAT und Threshold-SAT). *Definiere die folgenden beiden Probleme:*

$$\begin{aligned} \text{Majority-SAT} &= \left\{ \varphi \left| \begin{array}{l} \varphi \text{ ist eine boolesche Formel mit } n \text{ Variablen} \\ \text{und mindestens } 2^{n-1} \text{ erfüllenden Belegungen} \end{array} \right. \right\}; \\ \text{Threshold-SAT} &= \left\{ \langle \varphi, i \rangle \left| \begin{array}{l} \varphi \text{ ist eine boolesche Formel mit} \\ \text{mindestens } i \text{ erfüllenden Belegungen} \end{array} \right. \right\}. \end{aligned}$$

Beispielsweise ist $\varphi(x,y) = x \wedge y$ nicht in Majority-SAT, da nur eine von vier möglichen Belegungen von φ erfüllt ist. Andererseits ist $\psi(x,y) = x \vee y$ in Majority-SAT, da diese Formel drei erfüllende Belegungen hat. Für diese beiden Formeln, φ und ψ , gehören die Instanzen $\langle \varphi, 0 \rangle$, $\langle \varphi, 1 \rangle$, $\langle \psi, 0 \rangle$, $\langle \psi, 1 \rangle$, $\langle \psi, 2 \rangle$ und $\langle \psi, 3 \rangle$ jeweils zu Threshold-SAT, aber keine der Instanzen $\langle \varphi, 2 \rangle$, $\langle \varphi, 3 \rangle$, $\langle \varphi, 4 \rangle$ und $\langle \psi, 4 \rangle$ ist in Threshold-SAT.

Satz 6.11. Majority-SAT und Threshold-SAT sind beide \leq_m^p -vollständig für PP.

Beweis. Dass Majority-SAT in PP liegt, ist leicht zu sehen. Wir beweisen nun, dass

1. Threshold-SAT ein \leq_m^p -hartes Problem für PP ist und dass
2. Threshold-SAT \leq_m^p Majority-SAT gilt.

Da PP unter \leq_m^p -Reduktionen abgeschlossen ist (siehe Übung 6.5), folgen daraus beide Aussagen des Satzes.

1. Threshold-SAT ist \leq_m^p -hart für PP: Sei A eine beliebige Menge in PP, und M sei eine NPTM, die A im Sinne von PP akzeptiert. Gemäß Anmerkung 6.3 dürfen wir davon ausgehen, dass M eine normalisierte Maschine ist und dass es ein Polynom p gibt, so dass für jedes x der Länge n gilt:

$$x \in A \iff \text{acc}_M(x) \geq 2^{p(n)-1}.$$

Sei f_M die Cook-Reduktion aus dem Beweis von Satz 3.49, und sei $\varphi_{M,x} = f_M(x)$ die entsprechende boolesche Formel. Weil die Cook-Reduktion „geizig“ ist (siehe z.B. den Beweis von Satz 3.82 für die Definition dieses Begriffs), gilt:

$$\text{acc}_M(x) = \|\{\beta \mid \beta \text{ ist eine erfüllende Belegung für } \varphi_{M,x}\}\|.$$

Somit zeigt die Reduktion $g(x) = \langle \varphi_{M,x}, 2^{p(|x|)-1} \rangle$, dass $A \leq_m^p \text{Threshold-SAT}$ gilt.

2. $\text{Threshold-SAT} \leq_m^p \text{Majority-SAT}$: Sei $\langle \varphi, i \rangle$ eine gegebene Threshold-SAT -Instanz, wobei φ eine boolesche Formel in den Variablen x_1, x_2, \dots, x_m ist. Konstruiere eine Formel $\psi = \psi(x_1, x_2, \dots, x_m)$ so, dass ψ genau $j = 2^m - i$ erfüllende Belegungen hat, wobei wir annehmen, dass $i \leq 2^m$ ist. Betrachte die Binärentwicklung von

$$j = 2^{m-s_1} + 2^{m-s_2} + \dots + 2^{m-s_k},$$

wobei $0 \leq s_1 < s_2 < \dots < s_k \leq m$ gilt. Definiere die Formel

$$\begin{aligned} \psi = & (x_1 \wedge \dots \wedge x_{s_1-1} \wedge x_{s_1}) \vee \\ & (x_1 \wedge \dots \wedge x_{s_1-1} \wedge \neg x_{s_1} \wedge x_{s_1+1} \wedge \dots \wedge x_{s_2-1} \wedge x_{s_2}) \vee \\ & \vdots \\ & (x_1 \wedge \dots \wedge x_{s_1-1} \wedge \neg x_{s_1} \wedge x_{s_1+1} \wedge \dots \wedge x_{s_2-1} \wedge \neg x_{s_2} \wedge x_{s_2+1} \\ & \wedge \dots \wedge x_{s_{k-1}-1} \wedge \neg x_{s_{k-1}} \wedge x_{s_{k-1}+1} \wedge \dots \wedge x_{s_k}). \end{aligned}$$

Zunächst ist festzustellen, dass die ℓ -te Konjunktion in ψ genau 2^{m-s_ℓ} erfüllende Belegungen beisteuert, und wegen der Negationen in ψ kann keine Belegung, die eine Konjunktion erfüllt, auch eine andere erfüllen. Es wird also keine der erfüllenden Belegungen doppelt gezählt. Folglich summiert sich die Anzahl der Belegungen, die ψ erfüllen, wie gewünscht zu:

$$2^{m-s_1} + 2^{m-s_2} + \dots + 2^{m-s_k} = j.$$

Fixiere nun eine Formel $\gamma \notin \text{Majority-SAT}$; zum Beispiel könnte man $\gamma = x \wedge y$ wählen. Definiere die Reduktion $\text{Threshold-SAT} \leq_m^p \text{Majority-SAT}$ durch

$$f(\langle \varphi, i \rangle) = \begin{cases} \gamma & \text{falls } i > 2^m \\ (x_0 \wedge \varphi(x_1, \dots, x_m)) \vee (\neg x_0 \wedge \psi(x_1, \dots, x_m)) & \text{falls } i \leq 2^m. \end{cases}$$

Es ist leicht zu sehen, dass sowohl $\langle \varphi, i \rangle \notin \text{Threshold-SAT}$ als auch $f(\langle \varphi, i \rangle) = \gamma \notin \text{Majority-SAT}$ gilt, wenn $i > 2^m$ ist. Ist jedoch $i \leq 2^m$, dann gehört $\langle \varphi, i \rangle$ genau dann zu Threshold-SAT , wenn $f(\langle \varphi, i \rangle)$ durch mindestens $i + 2^m - i = 2^m$ der insgesamt 2^{m+1} möglichen Belegungen erfüllt wird. \square

6.2.2 BPP: Probabilistische Polynomialzeit mit beschränktem Fehler

Das Akzeptierungskriterium für PP-Maschinen ist nicht sehr robust, denn schon das Hinzufügen oder Wegnehmen eines einzigen akzeptierenden Pfades kann zu einem anderen Ausgang bezüglich der Akzeptierung oder Ablehnung der Eingabe führen. Anders gesagt, wächst die Eingabegröße ins Unendliche, dann kann die Fehlerwahrscheinlichkeit asymptotisch zu ein halb gehen. Unser Ziel ist es nun, die Fehlerwahrscheinlichkeit von ein halb weg zu beschränken. Zu diesem Zweck wird die Komplexitätsklasse BPP eingeführt.

Definition 6.12 (Bounded-Error Probabilistic Polynomial Time). Die Klasse Bounded-error Probabilistic Polynomial Time ist definiert durch

$$\text{BPP} = \left\{ A \middle| \begin{array}{l} \text{es gibt eine NPTM } M \text{ und eine Konstante } c, 0 < c \leq 1/2, \\ \text{so dass für jede Eingabe } x \text{ gilt:} \\ x \in A \implies \Pr(\{\alpha \mid M \text{ akzeptiert } x \text{ auf Pfad } \alpha\}) \geq 1/2 + c; \\ x \notin A \implies \Pr(\{\alpha \mid M \text{ akzeptiert } x \text{ auf Pfad } \alpha\}) \leq 1/2 - c \end{array} \right\}.$$

Das heißt, eine BPP-Berechnung akzeptiert entweder ihre Eingabe mit hoher Wahrscheinlichkeit oder sie lehnt sie mit hoher Wahrscheinlichkeit ab, wobei es eine echte Lücke um den Wert ein halb herum gibt, die von der Akzeptierungs- oder Ablehnungswahrscheinlichkeit streng vermieden wird. Deshalb ist BPP ebenfalls eine Promise-Klasse.

Sei r eine Funktion von \mathbb{N} in das reelle Intervall $[0, 1]$. Wir sagen, eine NPTM M akzeptiert eine Menge A im Sinne von BPP mit Fehlerwahrscheinlichkeit höchstens r genau dann, wenn

$$\Pr(\{\alpha \mid M(x) = c_A(x) \text{ auf Pfad } \alpha\}) \geq 1 - r(|x|)$$

gilt, wobei c_A die charakteristische Funktion von A bezeichnet.

Nun wird gezeigt, dass die Fehlerwahrscheinlichkeit von BPP-Berechnungen exponentiell klein in der Eingabegröße gemacht werden kann. Dann ist der Fehler einer solchen BPP-Berechnung „vernachlässigbar klein“.

Satz 6.13. Sei p ein Polynom und sei A eine beliebige Menge in BPP. Dann gibt es eine NPTM N , die A im Sinne von BPP mit Fehlerwahrscheinlichkeit höchstens $2^{-p(n)}$ akzeptiert.

Beweis. Fixiere ein Polynom p . Sei eine beliebige Menge A in BPP gegeben, und seien M eine NPTM und c eine Konstante mit $0 < c \leq 1/2$, so dass gilt:

$$\Pr(\{\alpha \mid M(x) = c_A(x) \text{ auf Pfad } \alpha\}) \geq \frac{1}{2} + c.$$

Setze $k = 2q(n) + 1$, für eine Eingabe x der Länge n und für ein Polynom q , das später spezifiziert werden wird. Wie im Beweis von Satz 6.6 wird nun eine NPTM N konstruiert, die bei Eingabe x die Berechnung von $M(x)$ k -mal hintereinander in unabhängigen Versuchen simuliert. Also besteht jeder Pfad $\alpha = \alpha_1 \alpha_2 \cdots \alpha_k$ von $N(x)$

aus einer Folge von k Pfaden α_i von $M(x)$. Anders als in dem früheren Satz definieren wir nun jedoch, dass α genau dann akzeptieren möge, wenn eine Mehrheit (also mindestens $q(n) + 1$) der Pfade α_i von $M(x)$ entlang α akzeptieren.

Zu zeigen ist, dass man ein Polynom q finden kann, so dass die Fehlerwahrscheinlichkeit von $N(x)$, die gegeben ist durch

$$E_N(x) = \Pr(\{\alpha \mid N(x) \neq c_A(x) \text{ auf Pfad } \alpha\}),$$

nach oben durch $2^{-p(n)}$ beschränkt ist. Damit α eine solche fehlerhafte Berechnung ist, muss nach unserer Festlegung, wann ein Pfad $\alpha = \alpha_1 \alpha_2 \cdots \alpha_k$ akzeptiert, ein j existieren, für das gilt:

- $j \leq q(n)$,
- j Teilstufen α_i entlang α sind korrekt im Sinne von $M(x) = c_A(x)$, und
- die übrigen $k - j$ Teilstufen α_i von α sind inkorrekt im Sinne von $M(x) \neq c_A(x)$.

Bezeichne die Erfolgswahrscheinlichkeit von $M(x)$ mit $\sigma = 1/2 + c$ und die Fehlerwahrscheinlichkeit von $M(x)$ mit $\varepsilon = 1/2 - c$. Wählt man unter den k möglichen Teilstufen α_i von α die j korrekten aus und summiert über alle möglichen j , so kann die Fehlerwahrscheinlichkeit von $N(x)$ wie folgt abgeschätzt werden:

$$E_N(x) \leq \sum_{j=0}^{q(n)} \binom{k}{j} \sigma^j \varepsilon^{k-j}. \quad (6.4)$$

Sei $m > 0$ so gewählt, dass $j = \frac{k}{2} - m$ und $k - j = \frac{k}{2} + m$ gilt. Da $\varepsilon < \sigma$ ist, folgt

$$\sigma^j \varepsilon^{k-j} = (\sigma \cdot \varepsilon)^{\frac{k}{2}} \cdot \sigma^{-m} \cdot \varepsilon^m = (\sigma \cdot \varepsilon)^{\frac{k}{2}} \cdot \left(\frac{\varepsilon}{\sigma}\right)^m < (\sigma \cdot \varepsilon)^{\frac{k}{2}}. \quad (6.5)$$

Der binomische Satz, nach welchem $(a + b)^k = \sum_{j=0}^k \binom{k}{j} a^j b^{k-j}$ gilt, impliziert für die spezielle Wahl von $a = b = 1$:

$$\sum_{j=0}^k \binom{k}{j} = 2^k. \quad (6.6)$$

Setzt man (6.5) und (6.6) in (6.4) ein, so ergibt sich:

$$\begin{aligned} E_N(x) &< (\sigma \cdot \varepsilon)^{\frac{k}{2}} \cdot 2^k \\ &= (4\sigma\varepsilon)^{\frac{k}{2}} \\ &= (1 - 4c^2)^{\frac{k}{2}}, \quad \text{da } \sigma\varepsilon = (\frac{1}{2} + c)(\frac{1}{2} - c) = \frac{1}{4} - c^2 \\ &\leq (1 - 4c^2)^{q(n)}, \end{aligned}$$

wobei die letztere Ungleichung aus

$$1 - 4c^2 < 1 \quad \text{und} \quad \frac{k}{2} = q(n) + \frac{1}{2} > q(n)$$

folgt. Weil $1 - 4c^2 < 1$ ist, ergibt sich weiter $(1 - 4c^2)^t \leq 1/2$ für eine ganze Zahl t . Setzen wir nun $q(n) = t \cdot p(n)$, so erhalten wir:

$$E_N(x) \leq (1 - 4c^2)^{t \cdot p(n)} \leq \left(\frac{1}{2}\right)^{p(n)} \leq 2^{-p(n)},$$

wie gewünscht. \square

Aus den Definitionen ergeben sich unmittelbar die folgenden Inklusionen und der Abschluss von BPP unter Komplement. Somit ist Satz 6.13 insbesondere auch auf RP und coRP anwendbar.

Fakt 6.14 $RP \subseteq BPP = coBPP \subseteq PP$.

Welche Beziehung besteht zwischen BPP und NP und zwischen BPP und der Polynomialzeit-Hierarchie? Wir werden später sehen, dass BPP in der zweiten Stufe der Polynomialzeit-Hierarchie enthalten ist. BPP und NP jedoch sind höchstwahrscheinlich unvergleichbar, d.h., man geht allgemein davon aus, dass weder $BPP \subseteq NP$ noch $NP \subseteq BPP$ gilt. Das folgende Resultat liefert ein gewisses Indiz dafür, dass $NP \subseteq BPP$ wahrscheinlich nicht gilt. Die Umkehrung von Satz 6.15 folgt übrigens unmittelbar aus Fakt 6.14.

Satz 6.15. Ist $NP \subseteq BPP$, so gilt $NP = RP$.

Beweis. Angenommen, $NP \subseteq BPP$. Nach Satz 6.13 gibt es eine NPTM M , die SAT im Sinne von BPP mit Fehlerwahrscheinlichkeit höchstens 2^{-n} akzeptiert:

$$\Pr(\{\alpha \mid M(\varphi) = c_{\text{SAT}}(\varphi) \text{ auf Pfad } \alpha\}) \geq 1 - 2^{-n}, \quad (6.7)$$

wobei $n = |\varphi|$ die Länge der booleschen Formel φ in einer geeigneten Codierung ist. Durch Konstruktion einer RP-Maschine für SAT wollen wir $SAT \in RP$ zeigen. Da SAT in $NP \leq_m^p$ -vollständig und da $RP \leq_m^p$ -abgeschlossen ist, folgt dann $NP = RP$.

Sei $\varphi = \varphi(x_1, x_2, \dots, x_m)$ eine gegebene boolesche Formel und sei $s \in \{0, 1\}^*$, $|s| \leq m$, ein Wort. Definiere die Formel φ_s in $m - |s|$ Variablen, die man aus φ erhält, indem man das i -te Bit von s als Wahrheitswert der i -ten Variable in φ einsetzt:

$$\begin{aligned} \varphi_0(x_2, x_3, \dots, x_m) &= \varphi(0, x_2, x_3, \dots, x_m) \\ \varphi_1(x_2, x_3, \dots, x_m) &= \varphi(1, x_2, x_3, \dots, x_m) \\ \varphi_{00}(x_3, x_4, \dots, x_m) &= \varphi(0, 0, x_3, x_4, \dots, x_m) \\ &\vdots \end{aligned}$$

In Abhängigkeit von der verwendeten Codierung kann das Vereinfachen von φ zu einer durch ein kürzeres Wort codierten Formel φ_s führen, so dass dann $|\varphi_s| < |\varphi|$ gelten würde. Da aber die Fehlerwahrscheinlichkeit von M von der Größe der Eingabe abhängt, wollen wir $|\varphi_s| \geq |\varphi|$ erreichen. Daher wird φ_s mit einer hinreichend großen Anzahl neuer Variablen $v_1, v_2, \dots, v_{k(s)}$ „ausgestopft“. Definiere also für jedes

φ_s , wobei $s \in \{0, 1\}^*$ und $|s| \leq m$ gilt, die entsprechend „ausgestopfte“ Formel ψ_s in den Variablen $x_{|s|+1}, \dots, x_m, v_1, \dots, v_{k(s)}$ durch

$$\psi_s = \varphi_s(x_{|s|+1}, \dots, x_m) \wedge v_1 \wedge \dots \wedge v_{k(s)},$$

wobei $k(s)$ groß genug gewählt ist, so dass $|\psi_s| \geq |\varphi|$ gilt. Offenbar ist ψ_s genau dann erfüllbar, wenn φ_s es ist.

Um $SAT \in RP$ zu zeigen, beschreiben wir eine NPTM N , die SAT im Sinne von RP akzeptiert. Bei einer Eingabe $\varphi(x_1, x_2, \dots, x_m)$ der Länge n versucht N , eine erfüllende Belegung für φ zu finden, falls eine existiert. Dazu verzweigt N zunächst nichtdeterministisch und verwendet M , um Schritt für Schritt auf jedem ihrer nichtdeterministischen Berechnungspfade Kandidaten erfüllender Belegungen für φ zu konstruieren. Auf jedem solchen Pfad verifiziert N dann deterministisch, ob der konstruierte Kandidat tatsächlich φ erfüllt oder nicht.

Etwas genauer lässt sich die Arbeit von N bei Eingabe $\varphi(x_1, x_2, \dots, x_m)$ so beschreiben:

Schritt 1: Simuliere $M(\psi_0)$. Da $|\psi_0| \geq |\varphi| = n$ gilt, impliziert (6.7):

$$\begin{aligned} \Pr(\{\alpha \mid M(\psi_0) = c_{SAT}(\psi_0) \text{ auf Pfad } \alpha\}) &\geq 1 - 2^{-|\psi_0|} \\ &\geq 1 - 2^{-n}. \end{aligned} \quad (6.8)$$

- Auf den akzeptierenden Pfaden α von $M(\psi_0)$ speichert N die Belegung 0 für die Variable x_1 , indem sie das erste Bit von s_α auf 0 setzt, und fährt dann rekursiv mit der Simulation von $M(\psi_{00})$ fort.
- Auf den ablehnenden Pfaden α von $M(\psi_0)$ speichert N die Belegung 1 für die Variable x_1 , indem sie das erste Bit von s_α auf 1 setzt, und fährt dann rekursiv mit der Simulation von $M(\psi_{10})$ fort.

Nach m solchen Schritten hat N auf jedem Pfad α einen Kandidaten s_α einer erfüllenden Belegung für φ konstruiert.

Schritt 2: Auf jedem Pfad α überprüft N deterministisch, ob s_α tatsächlich φ erfüllt.

Falls ja, so akzeptiert N auf α ; andernfalls lehnt N auf α ab.

Ist $\varphi \notin SAT$, so lehnt $N(\varphi)$ wegen der Überprüfung in Schritt 2 auf allen Pfaden ab. Es gilt also:

$$\Pr(\{\alpha \mid N \text{ akzeptiert } \varphi \text{ auf Pfad } \alpha\}) = 0.$$

Sei nun $\varphi \in SAT$ angenommen. Gemäß (6.8) ist die Fehlerwahrscheinlichkeit von M in jeder der m Simulationen in Schritt 1 höchstens 2^{-n} . Da alle diese m Versuche unabhängig sind, kann die Akzeptierungswahrscheinlichkeit von $N(\varphi)$ folgendermaßen abgeschätzt werden:

$$\begin{aligned} \Pr(\{\alpha \mid N \text{ akzeptiert } \varphi \text{ auf Pfad } \alpha\}) &\geq (1 - 2^{-n})^m \\ &\geq (1 - m2^{-n}) \\ &\geq \frac{1}{2}, \end{aligned}$$

wobei die letztere Ungleichung aus der offensichtlichen Tatsache folgt, dass $m \leq n$ ist, was $m2^{-n} \leq 2^{-1} = 1/2$ impliziert. \square

6.3 Quantoren und Arthur-Merlin-Spiele

6.3.1 Quantoren und BPP

Nach Satz 5.24 kann die Klasse NP mittels polynomiell längenbeschränkter existenzieller Quantoren charakterisiert werden. Nämlich ist eine Menge A genau dann in NP, wenn es eine Menge B in P und ein Polynom p gibt, so dass für jedes $x \in \Sigma^*$ gilt: $x \in A \iff (\exists^p w) [\langle x, w \rangle \in B]$. Dies kann man auch so schreiben:

$$\begin{aligned} x \in A &\implies (\exists^p w) [\langle x, w \rangle \in B]; \\ x \notin A &\implies (\forall^p w) [\langle x, w \rangle \notin B]. \end{aligned}$$

Akzeptierung wird also durch den \exists^p -Quantor und Ablehnung durch den \forall^p -Quantor ausgedrückt. Da alle diese Quantoren polynomiell längenbeschränkt sind, lassen wir aus Bequemlichkeit ihre oberen Indizes weg. Beispielsweise schreiben wir einfach $NP = (\exists | \forall)$, um die oben genannte Charakterisierung von NP kurz zu beschreiben. Ähnlich schreiben wir $coNP = (\forall | \exists)$. Noch allgemeiner können die in Satz 5.31 angegebenen Quantorendarstellungen der Klassen der Polynomialzeit-Hierarchie in dieser Notation kompakt geschrieben werden. Beispielsweise ist $\Sigma_3^p = (\exists \forall | \forall \exists)$ und $\Pi_3^p = (\forall \exists | \exists \forall)$.

In dieser Weise wollen wir nun Paare von Quantorenfolgen verwenden, um Komplexitätsklassen einheitlich durch ihre Akzeptierungs- bzw. Ablehnungskriterien zu definieren. Natürlich sind nicht alle Quantorenfolgen miteinander kompatibel. Es ist vernünftig, ihre Konsistenz in dem Sinn zu verlangen, dass nur solche Quantorenfolgen miteinander gepaart werden dürfen, die sich nicht „überlappen“. Im Fall von $NP = (\exists | \forall)$ zum Beispiel geschieht es nie, dass $(\exists w) [\langle x, w \rangle \in B] \wedge (\forall w) [\langle x, w \rangle \notin B]$ gilt, also ist (\exists, \forall) ein vernünftiges Paar von Quantoren.

Definition 6.16. 1. Seien Ω_1 und Ω_2 zwei Folgen von jeweils n Quantoren. Das Paar (Ω_1, Ω_2) heißt vernünftig, falls für jedes $(n+1)$ -stellige Prädikat B , für jedes x und für alle $\mathbf{y} = (y_1, y_2, \dots, y_n)$ der quantifizierte Ausdruck

$$(\Omega_1 \mathbf{y}) [B(x, \mathbf{y})] \wedge (\Omega_2 \mathbf{y}) [\neg B(x, \mathbf{y})]$$

logisch widersprüchlich (also falsch) ist. Hier ist y_i die Variable, die durch den i -ten Quantor in Ω_1 bzw. in Ω_2 quantifiziert wird.

2. Sei (Ω_1, Ω_2) ein vernünftiges Paar von Folgen, die aus jeweils n (polynomiell längenbeschränkten) Quantoren bestehen. Definiere die Komplexitätsklasse $(\Omega_1 | \Omega_2)$ wie folgt: Eine Menge L ist genau dann in $(\Omega_1 | \Omega_2)$, wenn es ein $(n+1)$ -stelliges Prädikat B in P gibt, so dass für jedes $x \in \Sigma^*$ gilt:

$$\begin{aligned} x \in L &\implies (\Omega_1 \mathbf{y}) [B(x, \mathbf{y})]; \\ x \notin L &\implies (\Omega_2 \mathbf{y}) [\neg B(x, \mathbf{y})], \end{aligned}$$

wobei $\mathbf{y} = (y_1, y_2, \dots, y_n)$ und $|y_i| \leq p(|x|)$ für ein geeignetes Polynom p ist. Wieder ist y_i die durch den i -ten Quantor in Ω_1 bzw. in Ω_2 quantifizierte Variable.

Nun wird ein neuer Quantor eingeführt, nämlich der *Mehrheitsquantor*, mit dessen Hilfe probabilistische Klassen wie z.B. BPP und RP dargestellt werden können.

Definition 6.17 (Polynomiell längenbeschränkter Mehrheitsquantor). Seien B ein Prädikat, p und q Polynome und x ein fest gewähltes Wort. Der durch den Mehrheitsquantor \exists^+ quantifizierte Ausdruck $(\exists^+ y) [B(x, y)]$ ist genau dann wahr, wenn der Anteil aller Wörter y mit $|y| \leq p(|x|)$, die das Prädikat $B(x, y)$ erfüllen, mindestens $(1 - 2^{-q(|x|)})$ ist.

Der oben definierte Mehrheitsquantor hat nach Definition einen exponentiell kleinen Fehler. Für viele Zwecke (und zur Vereinfachung mancher Beweise) würde es auch genügen, nur einen Anteil von mindestens, sagen wir, drei Viertel aller Wörter y mit $|y| \leq p(|x|)$, die das Prädikat $B(x, y)$ erfüllen, zu verlangen. Im Sinne der in Satz 6.13 gezeigten Wahrscheinlichkeitsverstärkung für BPP kann man die Äquivalenz dieser beiden möglichen Definitionen des Mehrheitsquantors beweisen, wobei das zugrunde liegende Prädikat B geeignet zu modifizieren ist.

In der Notation von Definition 6.16 ist $\text{BPP} = (\exists^+ | \exists^+)$ und $\text{RP} = (\exists^+ | \forall)$. Man sieht leicht, dass das Vertauschen der Quantorenfolgen für Akzeptierung und Ablehnung die komplementäre Klasse hervorbringt. Der einfache Beweis von Fakt 6.18 wird dem Leser als Übung 6.6 überlassen.

Fakt 6.18 Für jedes vernünftige Paar $(\mathfrak{Q}_1, \mathfrak{Q}_2)$ von Quantorenfolgen gilt:

$$(\mathfrak{Q}_1 | \mathfrak{Q}_2) = \text{co}(\mathfrak{Q}_2 | \mathfrak{Q}_1).$$

Korollar 6.19. BPP ist unter Komplement abgeschlossen.

Der oben beschriebene einheitliche Zugang zur Definition von Komplexitätsklassen mittels Quantoren ist sehr zweckmäßig beim Studium der Inklusionsbeziehungen zwischen bestimmten Komplexitätsklassen und quantorenbasierten Hierarchien wie der Arthur-Merlin-Hierarchie, die in Abschnitt 6.3.2 eingeführt und untersucht wird. Unser Ziel hier ist es zunächst, weitere Charakterisierungen von BPP durch Quantoren bereitzustellen. Dies wird es uns ermöglichen, zu beweisen, dass BPP in der Polynomialzeit-Hierarchie enthalten ist. Zuerst brauchen wir jedoch einige technische Lemmata.

Lemma 6.20. Seien B ein beliebiges Prädikat in P und x ein beliebiges Wort. Angenommen, es gilt: $(\forall y) (\exists^+ z) [B(x, y, z)]$. Dann gelten die folgenden beiden Aussagen:

1. $(\exists^+ Z) (\forall y) (\exists z \in Z) [B(x, y, z)]$.
2. $(\forall Y) (\exists^+ z) (\forall y \in Y) [B(x, y, z)]$.

Dabei hängen die Polynome, die implizit die Längen der quantifizierten Variablen beschränken, von der Länge n von x ab. Gilt in der ersten Aussage $|z| \leq p(n)$ für ein Polynom p , dann wird Z als eine Variable aufgefasst, die über Mengen von Wörtern der Länge höchstens $p(n)$ variiert. Damit Z selbst durch ein Wort einer Länge polynomiell in n dargestellt werden kann, verlangen wir $\|Z\| = q(n)$ für ein geeignet gewähltes Polynom q . Außerdem sei r ein Polynom, das die Länge von y beschränkt, d.h., $|y| \leq r(n)$. Ein analoger Kommentar gilt für die Mengenvariable Y in der zweiten Aussage.

Beweis. Seien B eine Menge in P und x ein Wort der Länge n . Wir setzen voraus, dass

$$(\forall y) (\exists^+ z) [B(x, y, z)] \quad (6.9)$$

gelte. Wir zeigen nur die erste Aussage des Lemmas; der Beweis der zweiten Aussage wird weggelassen, siehe Übung 6.7. (Zu beachten ist dabei, dass es für den Beweis der zweiten Aussage wesentlich ist, den Mehrheitsquantor wie in Definition 6.17 mit exponentiell kleinem Fehler zu verwenden.)

Weiter seien p , q und r Polynome, die, wie im Lemma erklärt, die Variablenlängen bzw. die Kardinalität der Mengenvariable Z beschränken.

Sei $S = \{0, 1\}^{\leq p(n)}$ die Menge aller Binärwörter einer Länge höchstens $p(n)$. Offenbar ist $\|S\| = 2^{p(n)+1} - 1$. Unsere Mengenvariable Z variiert über die Teilmengen von S , die genau $q(n)$ Elemente haben. Es gibt genau $\binom{\|S\|}{q(n)}$ solcher Teilmengen. Nun schätzen wir die Anzahl der Teilmengen Z von S mit genau $q(n)$ Elementen ab, die

$$(\forall y) (\exists z \in Z) [B(x, y, z)]$$

nicht erfüllen. Äquivalent dazu könnte man danach fragen, wie viele Teilmengen Z von S mit genau $q(n)$ Elementen es gibt, so dass

$$(\exists y) (\forall z \in Z) [\neg B(x, y, z)] \quad (6.10)$$

erfüllt ist. Betrachte ein beliebiges, festes Wort \tilde{y} . Zur Vereinfachung des Beweises gehen wir hier von der „schwächeren“ Definition des Mehrheitsquantors \exists^+ aus; siehe den Absatz nach Definition 6.17. Das heißt also, nach unserer Annahme (6.9) erfüllen höchstens ein Viertel aller Wörter z mit $|z| \leq p(n)$ das Prädikat $B(x, \tilde{y}, z)$ nicht. Für dieses feste \tilde{y} ist somit die Anzahl der Teilmengen $Z \subseteq S$ mit $\|Z\| = q(n)$, für die $(\forall z \in Z) [\neg B(x, \tilde{y}, z)]$ gilt, höchstens

$$\binom{\lceil 2^{-2}(2^{p(n)+1} - 1) \rceil}{q(n)} \leq \binom{2^{p(n)-1}}{q(n)}.$$

Es gibt genau $2^{r(n)+1} - 1$ Wörter y einer Länge höchstens $r(n)$. Daraus folgt, dass die Anzahl der Teilmengen $Z \subseteq S$ mit $\|Z\| = q(n)$, für die $(\exists y) (\forall z \in Z) [\neg B(x, y, z)]$ gilt, höchstens

$$(2^{r(n)+1} - 1) \cdot \binom{2^{p(n)-1}}{q(n)} \leq 2^{r(n)+1} \cdot \binom{2^{p(n)-1}}{q(n)}$$

ist. Da es insgesamt $\binom{2^{p(n)+1}-1}{q(n)}$ Teilmengen von S mit $q(n)$ Elementen gibt, kann der Anteil solcher Teilmengen $Z \subseteq S$ mit $\|Z\| = q(n)$, für die (6.10) gilt, nach oben durch

$$\frac{2^{r(n)+1} \cdot \binom{2^{p(n)-1}}{q(n)}}{\binom{2^{p(n)+1}-1}{q(n)}} \leq \frac{2^{r(n)+1}}{2^{q(n)}} = \frac{1}{4} \quad (6.11)$$

abgeschätzt werden, wobei die Wahl von $q(n) = r(n) + 3$ sichert, dass die Gleichheit in (6.11) gilt. Die Ungleichung in (6.11) folgt aus der Produktformel für Binomialkoeffizienten, $\binom{m}{k} = \prod_{i=1}^k \frac{m+1-i}{i}$, und weil $2^{p(n)+1} \geq 2(2^{p(n)-1} + 1)$ in jedem der

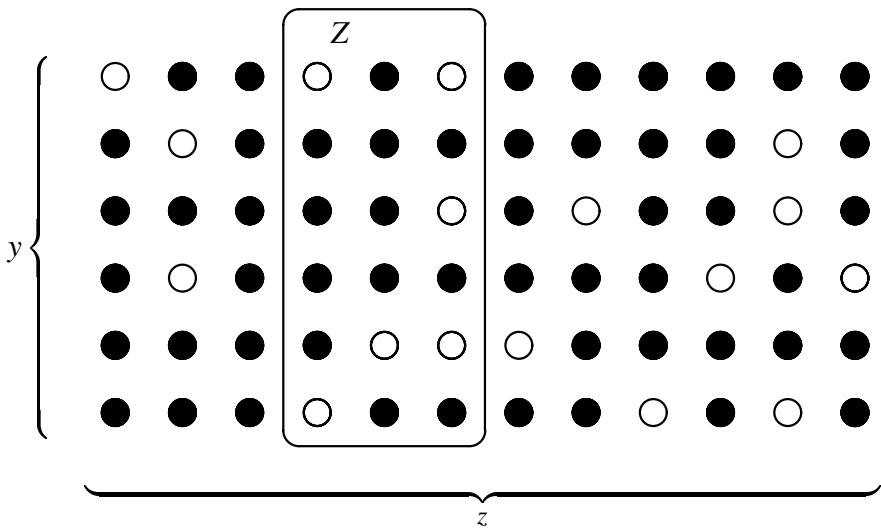


Abb. 6.4. Illustration der ersten Aussage von Lemma 6.20

$q(n)$ Faktoren der Produkte für $\binom{2^{p(n)}-1}{q(n)}$ bzw. $\binom{2^{p(n)+1}-1}{q(n)}$ gilt. Es folgt

$$(\exists^+ Z) (\forall y) (\exists z \in Z) [B(x, y, z)],$$

wie gewünscht. \square

Abbildung 6.4 zeigt eine illustrative Interpretation der ersten Aussage von Lemma 6.20:

$$(\forall y) (\exists^+ z) [B(x, y, z)] \implies (\exists^+ Z) (\forall y) (\exists z \in Z) [B(x, y, z)]. \quad (6.12)$$

Abbildung 6.4 kann man als eine Matrix verstehen, deren Einträge die Werte des Prädikats $B(x, y, z)$ für ein festes x und veränderliche y und z darstellen. Ein voller Kreis bedeutet, dass $B(x, y, z)$ wahr ist, und ein leerer Kreis bedeutet, dass $B(x, y, z)$ falsch ist. Die linke Seite von (6.12) sagt, dass jede Zeile y „viele“ (nämlich, im Sinne der „schwächeren“ Definition des Mehrheitsquantors \exists^+ , mindestens drei Viertel) volle Kreise besitzt. Die rechte Seite von (6.12) sagt, dass für viele „Fenster“ Z jede Zeile in diesem Fenster mindestens einen vollen Kreis hat. (Die Spalten eines solchen Fensters müssen natürlich nicht nebeneinander liegen.)

Lemma 6.20 wird unten in den Beweisen von Lemma 6.21 und Satz 6.22 angewandt. Die Behauptung von Lemma 6.21 ist „fast“ die für unwahrscheinlich gehaltene Inklusion

$$\Sigma_2^p = (\exists | \forall) (\exists | \forall) \subseteq (\forall | \exists) = \Pi_2^p,$$

nur dass die Ablehnungsbedingungen der beiden Klassen in Lemma 6.21 einen \exists^+ -Quantor anstatt eines \exists -Quantors haben.

Lemma 6.21. $(\exists \forall | \forall \exists^+) \subseteq (\forall \exists | \exists^+ \forall)$.

Beweis. Sei A eine beliebige Menge in $(\exists \forall | \forall \exists^+)$. Nach Definition 6.16 gibt es ein Prädikat B in P , so dass für jedes x gilt:

$$\begin{aligned} x \notin A &\implies (\forall y) (\exists^+ z) [\neg B(x, y, z)] && \text{nach Definition 6.16} \\ &\implies (\exists^+ Z) (\forall y) (\exists z \in Z) [\neg B(x, y, z)] && \text{nach Aussage 1 von Lemma 6.20} \\ &\implies (\exists Z) (\forall y) (\exists z \in Z) [\neg B(x, y, z)] \\ &\implies (\forall y) (\exists z) [\neg B(x, y, z)] \\ &\implies x \notin A, \end{aligned}$$

wobei die letzte Implikation wieder aus Definition 6.16 folgt, wenn man die Kontraposition von „ $x \in A \implies (\exists y) (\forall z) [B(x, y, z)]$ “ anwendet. Definieren wir nun das Prädikat C durch

$$C(x, y, Z) \equiv (\forall z \in Z) [B(x, y, z)],$$

so folgt:

$$x \notin A \iff (\exists^+ Z) (\forall y) [\neg C(x, y, Z)]; \quad (6.13)$$

$$x \notin A \iff (\exists Z) (\forall y) [\neg C(x, y, Z)]. \quad (6.14)$$

Durch Negation von (6.14), erhalten wir:

$$x \in A \iff (\forall Z) (\exists y) [C(x, y, Z)]. \quad (6.15)$$

Da Z nur polynomiell viele Elemente enthält, impliziert $B \in P$, dass auch C in P ist. Nach Definition 6.16 folgt aus (6.13) und (6.15), dass A eine Menge in $(\forall \exists | \exists^+ \forall)$ ist. \square

Das folgende Resultat ist als das „BPP-Theorem“ bekannt, da es zwei weitere nützliche Charakterisierungen der Klasse BPP hinsichtlich quantorenbasierter Komplexitätsklassen zur Verfügung stellt. Wichtig ist die Bemerkung, dass Satz 6.22 in jedem (vernünftigen) Quantorenkontext gilt.

Satz 6.22 (Zachos und Heller).

$$\text{BPP} = (\exists^+ | \exists^+) = (\forall \exists^+ | \exists^+ \forall) = (\exists^+ \forall | \forall \exists^+).$$

Beweis. Es genügt, die Gleichheit $(\exists^+ | \exists^+) = (\forall \exists^+ | \exists^+ \forall)$ zu beweisen. Die andere Gleichheit folgt unmittelbar aus Fakt 6.18 und Korollar 6.19, denn es ist

$$(\forall \exists^+ | \exists^+ \forall) = \text{co}(\exists^+ \forall | \forall \exists^+).$$

Um die Inklusion $(\forall \exists^+ | \exists^+ \forall) \subseteq (\exists^+ | \exists^+)$ zu zeigen, sei A eine beliebige Menge in $(\forall \exists^+ | \exists^+ \forall)$. Nach Definition 6.16 gibt es eine Menge B in P , so dass für jedes x gilt:

$$\begin{aligned} x \in A &\implies (\forall y) (\exists^+ z) [B(x, y, z)] \\ &\implies (\exists^+ \langle y, z \rangle) [C(x, \langle y, z \rangle)], \end{aligned}$$

wobei das Prädikat C durch $C(x, \langle y, z \rangle) \equiv B(x, y, z)$ definiert ist. Offenbar ist C in P. Außerdem gilt für jedes x :

$$\begin{aligned} x \notin A &\implies (\exists^+ y) (\forall z) [\neg B(x, y, z)] \\ &\implies (\exists^+ \langle y, z \rangle) [\neg C(x, \langle y, z \rangle)]. \end{aligned}$$

Somit ist A in $(\exists^+ | \exists^+)$.

Um umgekehrt die Inklusion $(\exists^+ | \exists^+) \subseteq (\forall \exists^+ | \exists^+ \forall)$ zu beweisen, sei A eine beliebige Menge in $(\exists^+ | \exists^+)$. Also existiert eine Menge B in P, so dass für jedes x gilt:

$$x \in A \implies (\exists^+ y) [B(x, y)]; \quad (6.16)$$

$$x \notin A \implies (\exists^+ y) [\neg B(x, y)]. \quad (6.17)$$

Tabelle 6.3. Definition des Prädikats C im Beweis von Satz 6.22

y	s_0	s_1	s_2	\dots	s_{m-2}	s_{m-1}
z	b_0	b_1	b_2	\dots	b_{m-2}	b_{m-1}
s_0	b_0	b_1	b_2	\dots	b_{m-2}	b_{m-1}
s_1	b_1	b_2	b_3	\dots	b_{m-1}	b_0
s_2	b_2	b_3	b_4	\dots	b_0	b_1
\vdots	\vdots	\vdots	\vdots	\ddots	\vdots	\vdots
s_{m-2}	b_{m-2}	b_{m-1}	b_0	\dots	b_{m-4}	b_{m-3}
s_{m-1}	b_{m-1}	b_0	b_1	\dots	b_{m-3}	b_{m-2}

Sei p ein Polynom, das die Länge der Variablen y beschränkt, die in (6.16) und (6.17) quantifiziert wird. Sei x ein beliebiges, fest gewähltes Wort der Länge n . Sei $S = \{0, 1\}^{\leq p(n)}$ die Menge aller Binärwörter einer Länge höchstens $p(n)$. Weiter nehmen wir an, dass die Wörter in S lexikographisch geordnet sind, d.h., $S = \{s_0, s_1, \dots, s_{m-1}\}$, wobei $m = 2^{p(n)+1} - 1$ ist. Für das feste Wort x sei ein Prädikat $C(x, y, z)$ wie folgt definiert:

$$C(x, y, z) \equiv C(x, s_i, s_j) \equiv B(x, s_{i+j \bmod m}),$$

wobei die Variablen y und z über die Wörter in S variieren und i und j aus der Menge $\mathbb{Z}_m = \{0, 1, \dots, m-1\}$ sind, d.h., sie sind Reste modulo m . Tabelle 6.3 illustriert die Definition von C : Mit $b_i = B(x, s_i)$ für $i \in \mathbb{Z}_m$ gibt die i -te Zeile von Tabelle 6.3 die Werte von $C(x, y, s_i)$ für variierendes y an, und die j -te Spalte von Tabelle 6.3 gibt die Werte von $C(x, s_j, z)$ für variierendes z an. Wie man sieht, ist $C(x, y, z)$ symmetrisch in den letzten beiden Argumenten. Das heißt, die Zeilen werden zyklisch um je eine Position verschoben und die Spalten ebenso. Folglich hat jede Zeile und jede Spalte in Tabelle 6.3 dieselbe Anzahl von Einsen und dieselbe Anzahl von Nullen.

Weil B in P ist, gehört auch C zu P . Aus (6.16) folgt, dass für jedes x gilt:

$$\begin{aligned} x \in A &\implies (\forall z) (\exists^+ y) [C(x, y, z)] && \text{da jede Zeile dieselbe} \\ &&& \text{Anzahl von Einsen hat} \\ &\implies (\forall y) (\exists^+ z) [C(x, y, z)] && \text{da } C \text{ symmetrisch in den} \\ &&& \text{letzten beiden Argumenten ist} \\ &\implies (\forall Y) (\exists^+ z) \underbrace{(\forall y \in Y) [C(x, y, z)]}_{D(x, z, Y)} && \text{nach Aussage 2 von Lemma 6.20,} \end{aligned}$$

wobei das Prädikat D durch $D(x, z, Y) \equiv (\forall y \in Y) [C(x, y, z)]$ definiert ist. (Zu beachten ist wie gesagt hier, dass der Mehrheitsquantor in der zweiten Aussage von Lemma 6.20 wie in Definition 6.17 mit exponentiell kleinem Fehler verwendet wird.)

Da der Allquantor in D über einen Bereich polynomialer Größe variiert, impliziert $C \in P$, dass auch D in P ist.

Ähnlich folgt aus (6.17), dass für jedes x gilt:

$$\begin{aligned} x \notin A &\implies (\forall z) (\exists^+ y) [\neg C(x, y, z)] && \text{da jede Zeile dieselbe} \\ &&& \text{Anzahl von Nullen hat} \\ &\implies (\forall y) (\exists^+ z) [\neg C(x, y, z)] && \text{da } C \text{ symmetrisch in den} \\ &&& \text{letzten beiden Argumenten ist} \\ &\implies (\exists^+ Z) (\forall y) \underbrace{(\exists z \in Z) [\neg C(x, y, z)]}_{\neg D(x, y, Z)} && \text{nach Aussage 1 von Lemma 6.20} \\ &\implies (\exists^+ Z) (\forall y) [\neg D(x, y, Z)]. \end{aligned}$$

Somit ist A in $(\forall \exists^+ | \exists^+ \forall)$. □

Korollar 6.23. $\text{BPP} \subseteq \Sigma_2^P \cap \Pi_2^P$.

Beweis. Da BPP gemäß Korollar 6.19 bezüglich Komplement abgeschlossen ist, genügt es, die Inklusion $\text{BPP} \subseteq \Sigma_2^P$ zu zeigen. Sei also A eine beliebige Menge in BPP . Nach Satz 6.22 ist $\text{BPP} = (\exists^+ \forall | \forall \exists^+)$. Somit gibt es eine Menge B in P , so dass für jedes x gilt:

$$\begin{aligned} x \in A &\implies (\exists^+ y) (\forall z) [B(x, y, z)] \\ &\implies (\exists y) (\forall z) [B(x, y, z)] \end{aligned}$$

und

$$\begin{aligned} x \notin A &\implies (\forall y) (\exists^+ z) [\neg B(x, y, z)] \\ &\implies (\forall y) (\exists z) [\neg B(x, y, z)]. \end{aligned}$$

Nach Satz 5.31 ist A in Σ_2^P . □

6.3.2 Die Arthur-Merlin-Hierarchie

So höret nun die Geschichte von König Artus und Merlin, dem mächtigen Zauberer.

Story 6.24 (Arthur-Merlin-Spiele) *Merlin und Artus spielen das folgende Spiel. Das Ziel des Spiels ist es, dass sie gemeinsam ein Problem lösen. Nehmen wir zum Beispiel an, sie wollen das Graphisomorphie-Problem lösen. Für ein gegebenes Graphenpaar, G und H , müssen sie entscheiden, ob G und H isomorph sind oder nicht. Sie machen abwechselnd ihren Zug, und sie spielen um jede Problemeingabe. Merlins Absicht dabei ist es, Artus davon zu überzeugen, dass die gegebenen Graphen isomorph sind, selbst wenn sie in Wirklichkeit nicht isomorph sind. Ein Zug von Merlin besteht also in der Präsentation eines Beweises für die Isomorphie der gegebenen Graphen, und ein solcher Beweis ist ein Isomorphismus zwischen den Graphen.*

Artus jedoch ist argwöhnisch und misstraut dem hinterlistigen Zauberer. Zwar kann er sich selbst natürlich nicht solch mächtige Beweise wie dieser ausdenken. Schließlich hat Merlin übernatürliche nichtdeterministische Kräfte und Sprüche zu seiner Verfügung, und Artus nicht. Dennoch bezweifelt Artus die Gültigkeit der Beweise, wirft ein paar Münzen und in Abhängigkeit von diesen Zufallswahlen fordert er den Zauberer und seine Beweise heraus. Dies ist einer von Artus' Zügen in diesem Spiel. Nun ist Merlin wieder am Zug und so weiter. Schließlich, nach einer endlichen Zahl von Zügen, wird die Eingabe entweder akzeptiert oder verworfen. Es ist auch möglich, dass Artus in einem Spiel den ersten Zug macht.

Um nun etwas formaler und allgemeiner zu sein, nehmen wir an, dass Merlin durch eine NP-Maschine M und dass Artus durch eine randomisierte polynomialzeitbeschränkte Turingmaschine A repräsentiert wird. Sei x die Probleminstanz, um die es geht, und sei L das Problem, das sie lösen wollen. Ein Zug Merlins ist ein Beweis (oder Zeuge; vergleiche Beispiel 5.22 und Definition 5.23) für „ $x \in L$ “, und Merlin kann einen solchen Beweis durch Simulation von $M(x, y)$ finden, wobei y die Geschichte der bisher gemachten Züge codiert. Das heißt, das Wort y beschreibt alle nichtdeterministischen Wahlen von Merlin und alle Zufallswahlen von Artus, die zuvor in diesem Spiel gemacht wurden. Um den misstrauischen König zufrieden zu stellen, muss Merlin ihn mit überwältigender Wahrscheinlichkeit überzeugen. Ein Zug Artus' ist durch die Berechnung von $A(x, y)$ gegeben, die von Artus' Zufallswahlen abhängt, wobei y wieder die vorherige Geschichte des Spiels codiert.

Genau wie die Polynomialzeit-Hierarchie durch alternierende (polynomiell längenbeschränkte) existenzielle und universelle Quantoren beschrieben werden kann (siehe Satz 5.31 in Abschnitt 5.2), können Arthur-Merlin-Spiele durch alternierende (polynomiell längenbeschränkte) existenzielle und probabilistische Quantoren ausgedrückt werden. Der \exists -Quantor stellt dabei einen Zug Merlins dar, also eine NP-Berechnung, und der \exists^+ -Quantor stellt einen Zug Arthurs dar (den wir jetzt bei seinem englischen Namen nennen, da der Begriff der „Arthur-Merlin-Spiele“ der übliche Fachausdruck ist), also eine BPP-Berechnung. Der Bezeichnung aus Definition 6.16 entsprechend erhält man eine Hierarchie von Komplexitätsklassen, die so genannte Arthur-Merlin-Hierarchie.

Definition 6.25 (Arthur-Merlin-Hierarchie). Die folgenden Klassen bilden die Stufen der Arthur-Merlin-Hierarchie:

$$\begin{aligned} A &= (\exists^+ | \exists^+), \quad AM = (\exists^+ | \exists^+ \forall), \quad AMA = (\exists^+ \exists^+ | \exists^+ \forall \exists^+), \dots \\ M &= (\exists | \forall), \quad MA = (\exists^+ | \forall \exists^+), \quad MAM = (\exists \exists^+ | \forall \exists^+), \dots \end{aligned}$$

Definiere die Arthur-Merlin-Hierarchie AMH als die Vereinigung all dieser Klassen.

Beispiel 6.26 (Arthur-Merlin-Hierarchie). Betrachte die Klasse MA, welche aus genau den Problemen L besteht, für die es eine NPTM M und eine randomisierte Polynomialzeit-Turingmaschine A gibt, so dass für jede Eingabe x gilt:

- Ist $x \in L$, so existiert ein Pfad y von $M(x)$, so dass $A(x, y)$ mit einer Wahrscheinlichkeit von mindestens $3/4$ akzeptiert. Das heißt, Arthur kann Merlins korrekten Beweis y für „ $x \in L$ “ nicht widerlegen, und Merlin gewinnt.
- Ist $x \notin L$, so gilt für jeden Pfad y von $M(x)$, dass $A(x, y)$ mit einer Wahrscheinlichkeit von mindestens $3/4$ ablehnt. Das heißt, Arthur kann durch keinen von Merlins falschen Beweisen für „ $x \in L$ “ getäuscht werden und gewinnt deshalb.

Analog kann man die Klassen AM, MAM, AMA, … beschreiben, siehe Übung 6.8.

Da die in Definition 6.25 eingeführten Klassen mittels des Mehrheitsquantors \exists^+ definiert wurden, bestimmt dessen Fehlerschwellwert auch Arthurs' Fehlertoleranz. Wie nach Definition 6.17 erwähnt wurde, ist die Festlegung dieses Schwellwerts willkürlich. Selbst wenn man (wie in Beispiel 6.26) bei der Definition von \exists^+ – und somit auch in Definition 6.25 – eine an sich inakzeptabel hohe Fehlerwahrscheinlichkeit von z.B. $1/4$ zulassen würde (oder auch von $1/2 - \epsilon$ für eine beliebig kleine, feste Konstante $\epsilon > 0$), so könnte man doch mit den Techniken zur Verstärkung der Erfolgswahrscheinlichkeit aus Abschnitt 6.2 einen exponentiell kleinen Fehler erreichen, ohne die Polynomialzeit-Beschränkung zu verletzen. Ein so kleiner Fehler kann aber als vernachlässigbar gelten, da sogar das Auftreten von Hardware-Fehlern mit einer größeren Wahrscheinlichkeit erwartet werden kann.

Das folgende Resultat zeigt, dass für eine konstante Anzahl von Zügen die Arthur-Merlin-Hierarchie auf AM kollabiert. Ob irgendeine der Inklusionen $NP \subseteq MA \subseteq AM$ oder $BPP \subseteq MA$ echt ist oder nicht, ist eine offene Frage. Ebenso ist es offen, ob MA bzw. AM unter Komplement abgeschlossen ist oder nicht.

Satz 6.27. $NP \cup BPP \subseteq MA \subseteq AM = AMA = MAM = \dots = AMH$.

Beweis. Nach Definition 6.25 gilt $NP \subseteq MA$ und $BPP \subseteq MA$. Um $MA \subseteq AM$ zu beweisen, charakterisieren wir diese Klassen zunächst hinsichtlich geeigneter quantorenbasierter Klassen und wenden dann Lemma 6.21 an, d.h., wir beweisen nun:

$$MA = (\forall | \exists^+) \subseteq (\exists | \exists^+) = AM. \quad (6.18)$$

Im Folgenden werden Lemma 6.21 und Satz 6.22 in bestimmten Quantorenkontexten angewandt. Die Quantoren, auf die das Lemma oder das BPP-Theorem genau angewandt werden, sind zur besseren Unterscheidung vom jeweiligen Quantorenkontext unterstrichen.

Die erste Gleichheit in (6.18) sieht man wie folgt:

$$\begin{aligned}
 MA &= (\exists \underline{\exists}^+ | \forall \underline{\exists}^+) && \text{nach Definition 6.25} \\
 &= (\exists \underline{\exists}^+ | \forall \underline{\exists}^+ \underline{A}(\underline{\exists}^+)) && \text{nach Satz 6.22 im Quantorenkontext} \\
 &\subseteq (\exists \underline{\exists}^+ | \forall \underline{\exists}^+ \underline{A}(\underline{\exists}^+)) && \text{durch Abschwächen von } (\exists^+ v) [\dots] \text{ zu } (\exists v) [\dots] \\
 &= (\exists | \forall \underline{\exists}^+) && \text{durch Zusammenfassen gleicher Quantoren} \\
 &\subseteq (\exists \underline{\exists}^+ | \forall \underline{\exists}^+) && \text{durch Abschwächen von } (\forall v) [\dots] \text{ zu } (\exists^+ v) [\dots] \\
 &= MA && \text{nach Definition 6.25.}
 \end{aligned}$$

Die letzte Gleichheit in (6.18) sieht man wie folgt:

$$\begin{aligned}
 AM &= (\underline{\exists}^+ \underline{\exists} | \underline{\exists}^+ \underline{A}) && \text{nach Definition 6.25} \\
 &= (\underline{\exists}^+ \underline{\exists} | \underline{\exists}^+ \underline{A} \underline{A}(\underline{\exists}^+)) && \text{nach Satz 6.22 im Quantorenkontext} \\
 &\subseteq (\underline{\exists}^+ \underline{\exists} | \underline{\exists}^+ \underline{A} \underline{A}(\underline{\exists}^+)) && \text{durch Abschwächen von } (\exists^+ v) [\dots] \text{ zu } (\exists v) [\dots] \\
 &= (\underline{\exists} | \underline{\exists}^+ \underline{A}) && \text{durch Zusammenfassen gleicher Quantoren} \\
 &\subseteq (\underline{\exists}^+ \underline{\exists} | \underline{\exists}^+ \underline{A}) && \text{durch Abschwächen von } (\forall v) [\dots] \text{ zu } (\exists^+ v) [\dots] \\
 &= AM && \text{nach Definition 6.25.}
 \end{aligned}$$

Nach Lemma 6.21 ist die Inklusion (6.18) bewiesen, also gilt $MA \subseteq AM$.

Wir zeigen nun, dass die gesamte Arthur-Merlin-Hierarchie auf AM kollabiert. Dass die Inklusionen $AM \subseteq MAM$, $AM \subseteq AMA$ usw. gelten, ist klar. Wendet man umgekehrt die Inklusion $MA \subseteq AM$ aus (6.18) in einem geeigneten Quantorenkontext an, so erhält man

$$AMA \subseteq AAM \subseteq AM,$$

da zwei benachbarte \exists^+ -Quantoren genauso zu einem \exists^+ -Quantor zusammengefasst werden können, wie das für die \exists - oder \forall -Quantoren möglich ist.³

Die Inklusion $MAM \subseteq AM$ sieht man wie folgt:

$$\begin{aligned}
 MAM &= (\underline{\exists} \underline{\exists}^+ \underline{\exists} | \underline{\exists}^+ \underline{A}) && \text{nach Definition 6.25} \\
 &= (\underline{\exists} \underline{\exists}^+ \underline{\exists} | \underline{\exists}^+ \underline{A} \underline{A}(\underline{\exists}^+)) && \text{nach Satz 6.22 im Quantorenkontext} \\
 &\subseteq (\underline{\exists} \underline{\exists}^+ \underline{\exists} | \underline{\exists}^+ \underline{A} \underline{A}(\underline{\exists}^+)) && \text{durch Abschwächen von } (\exists^+ v) [\dots] \text{ zu } (\exists v) [\dots] \\
 &= (\underline{\exists} \underline{\exists}^+ | \underline{\exists}^+ \underline{A}) && \text{durch Zusammenfassen gleicher Quantoren} \\
 &\subseteq (\underline{\exists} \underline{\exists}^+ | \underline{\exists}^+ \underline{A}) && \text{nach Lemma 6.21 im Quantorenkontext} \\
 &= (\underline{\exists} | \underline{\exists}^+ \underline{A}) && \text{durch Zusammenfassen gleicher Quantoren} \\
 &\subseteq (\underline{\exists}^+ \underline{\exists} | \underline{\exists}^+ \underline{A}) && \text{durch Abschwächen von } (\forall v) [\dots] \text{ zu } (\exists^+ v) [\dots] \\
 &= AM && \text{nach Definition 6.25.}
 \end{aligned}$$

³ Beispielsweise gilt $(\exists^+ \exists^+ | \exists^+ \exists^+) = (\exists^+ | \exists^+) = \text{BPP}$. Äquivalent kann man dies als $\text{BPP}^{\text{BPP}} = \text{BPP}$ schreiben, siehe Problem 6.3(a).

Somit ist $\text{AM} = \text{MAM} = \text{AMA}$ gezeigt. Die übrigen Gleichheiten folgen nun mit Induktion. \square

Das folgende Korollar gibt die Beziehungen zwischen den Arthur-Merlin-Klassen und der Polynomialzeit-Hierarchie an. Außerdem wird das in Korollar 6.23 festgestellte Ergebnis verschärft, welches sagt, dass BPP in der zweiten Stufe der Polynomialzeit-Hierarchie enthalten ist.

Korollar 6.28. 1. $\text{MA} \subseteq \Sigma_2^P$ und $\text{AM} \subseteq \Pi_2^P$.

2. Gilt $\text{NP} \subseteq \text{coAM}$, so ist $\Sigma_2^P = \Pi_2^P = \text{PH}$.

3. $\text{BPP} \subseteq \text{MA} \cap \text{coMA}$.

Beweis. 1. Es gilt $\text{MA} = (\exists \exists^+ | \forall \exists^+) = (\exists \forall | \forall \exists^+) \subseteq (\exists \forall | \forall \exists) = \Sigma_2^P$, wobei die zweite Gleichheit aus (6.18) im Beweis von Satz 6.27 folgt. Die Inklusion $\text{AM} \subseteq \Pi_2^P$ kann analog gezeigt werden.

2. Angenommen, $\text{NP} \subseteq \text{coAM}$, d.h., $(\exists | \forall) \subseteq (\exists^+ \forall | \exists^+ \exists)$. Dann gilt:

$$\Pi_2^P = (\forall \exists | \exists \forall) \subseteq (\forall \exists^+ \forall | \exists \exists^+ \exists) = \text{coMAM} = \text{coAM} \subseteq \Sigma_2^P,$$

wobei die letzte Inklusion aus dem ersten Teil dieses Korollars folgt. Nach Satz 5.33 kollabiert die Polynomialzeit-Hierarchie auf ihre zweite Stufe.

3. Dies folgt unmittelbar aus Satz 6.27 und dem Abschluss von BPP unter Komplement, welcher in Fakt 6.14 festgestellt wurde. \square

Satz 6.29 (Schöning). $\Sigma_2^{P,\text{AM} \cap \text{coAM}} = \Sigma_2^P$.

Beweis. Der Beweis ist dem Beweis von Satz 5.31 ähnlich (oder besser gesagt dessen relativierter Version). Die Inklusion $\Sigma_2^P \subseteq \Sigma_2^{P,\text{AM} \cap \text{coAM}}$ liegt auf der Hand. Um die umgekehrte Inklusion zu beweisen, $\Sigma_2^{P,\text{AM} \cap \text{coAM}} \subseteq \Sigma_2^P$, werden wir die dazu äquivalente Inklusion $\Pi_2^{P,\text{AM} \cap \text{coAM}} \subseteq \Pi_2^P$ zeigen. Sei also L eine beliebige Menge in $\Pi_2^{P,\text{AM} \cap \text{coAM}}$. Nach der relativierten Version von Korollar 5.32, gibt es eine Menge $A \in \text{AM} \cap \text{coAM}$ und ein Prädikat B in P^A , so dass für jedes x gilt:

$$x \in L \iff (\forall y) (\exists z) [B(x, y, z)],$$

wobei wie üblich alle Quantoren polynomiell längenbeschränkt sind. Sei M eine DPOTM, die B mit Orakel A entscheidet, d.h., $L(M^A) = B$. Definiere ein Prädikat C wie folgt:

$$C(u, v, x, y, z) \equiv \begin{cases} M^A(x, y, z) \text{ akzeptiert}, u = \langle u_1, u_2, \dots, u_k \rangle, v = \langle v_1, v_2, \dots, v_\ell \rangle, \\ \text{wobei } u \text{ und } v \text{ genau die Fragen von } M^A(x, y, z) \text{ enthalten} \\ \text{und genau die Fragen in } u \text{ die Antwort „ja“ haben} \\ \text{und genau die Fragen in } v \text{ die Antwort „nein“ haben.} \end{cases}$$

Definiere die Mengen

$$\begin{aligned} A_{\text{yes}} &= \{\langle u_1, u_2, \dots, u_k \rangle \mid u_1, u_2, \dots, u_k \in A\}; \\ A_{\text{no}} &= \{\langle v_1, v_2, \dots, v_\ell \rangle \mid v_1, v_2, \dots, v_\ell \notin A\}. \end{aligned}$$

Da sowohl A als auch \bar{A} in AM sind (und da AM unter Paarung abgeschlossen ist), sind sowohl A_{yes} als auch A_{no} in AM. Definieren wir ein Prädikat D durch $D(u, v) \equiv (u \in A_{\text{yes}} \wedge v \in A_{\text{no}})$, so folgt, dass D ebenfalls in AM liegt. Somit gilt für jedes x :

$$\begin{aligned} x \in L &\iff (\forall y) (\exists z) [M^A(x, y, z) \text{ akzeptiert}] \\ &\iff (\forall y) (\exists z) (\exists u) (\exists v) [C(u, v, x, y, z) \wedge \\ &\quad u_1, u_2, \dots, u_k \in A \wedge v_1, v_2, \dots, v_\ell \notin A] \\ &\iff (\forall y) (\exists z) (\exists u) (\exists v) [C(u, v, x, y, z) \wedge D(u, v)]. \end{aligned}$$

Nach (6.18) im Beweis von Satz 6.27 gilt $\text{AM} = (\forall \exists \mid \exists^+ \forall)$. Also gibt es ein Prädikat E in P, so dass gilt:

$$\begin{aligned} D(u, v) &\implies (\forall r) (\exists s) [E(u, v, r, s)]; \\ \neg D(u, v) &\implies (\exists^+ r) (\forall s) [\neg E(u, v, r, s)]. \end{aligned}$$

Für jedes x folgt damit:

$$\begin{aligned} x \in L &\implies (\forall y) (\exists z) (\exists u) (\exists v) [C(u, v, x, y, z) \wedge (\forall r) (\exists s) [E(u, v, r, s)]] \\ &\implies (\forall y) (\exists z) (\exists u) (\exists v) (\forall r) (\exists s) [C(u, v, x, y, z) \wedge E(u, v, r, s)] \end{aligned}$$

und

$$\begin{aligned} x \notin L &\implies (\exists y) (\forall z) (\forall u) (\forall v) [\neg C(u, v, x, y, z) \vee (\exists^+ r) (\forall s) [\neg E(u, v, r, s)]] \\ &\implies (\exists y) (\forall z) (\forall u) (\forall v) (\exists^+ r) (\forall s) [\neg C(u, v, x, y, z) \vee \neg E(u, v, r, s)]. \end{aligned}$$

Fasst man nun benachbarte Quantoren gleichen Typs zu einem Quantor dieses Typs zusammen, so ergibt sich, dass L zur Klasse

$$(\underline{\text{AA}}^+ \underline{\text{EE}} \mid \underline{\text{EE}} \underline{\text{AA}}) \subseteq (\underline{\text{AA}}^+ \underline{\text{EE}} \mid \underline{\text{EE}} \underline{\text{AA}}) = \Pi_2^P$$

gehört. Die erste Inklusion oben folgt aus einer Anwendung von Lemma 6.21 im Quantorenkontext (wobei die Quantoren, auf die das Lemma angewandt wird, wieder unterstrichen sind). Die zweite Inklusion folgt durch Zusammenfassen benachbarter Quantoren gleichen Typs zu einem Quantor dieses Typs und durch Abschwächen von \exists^+ zu \exists . \square

In Problem 6.3(c) ist zu zeigen, dass $\text{AM} \cap \text{coAM}$ low für AM ist, d.h., dass $\text{AM}^{\text{AM} \cap \text{coAM}} = \text{AM}$ gilt.

Eine unmittelbare Folgerung von Satz 6.29 ist, dass jede Menge, die sowohl in coAM als auch in NP liegt, low für die zweite Stufe der Polynomialzeit-Hierarchie ist. Dieses Resultat werden wir später im Beweis von Satz 6.45 anwenden, welcher sagt, dass das Graphisomorphie-Problem in Low_2 liegt.

Korollar 6.30 (Schöning). $\text{NP} \cap \text{coAM} \subseteq \text{Low}_2$.

6.4 Zählklassen

In Abschnitt 6.2 wurden probabilistische Klassen wie PP, RP, ZPP und BPP über das Wahrscheinlichkeitsgewicht akzeptierender Berechnungen eingeführt. In Anmerkung 6.3 wurde eine alternative Sichtweise auf diese Klassen erwähnt, die auf dem Zählen akzeptierender und ablehnender Berechnungen von NPTMs beruht. In diesem Abschnitt nun wird eine Theorie der so genannten Zählklassen entwickelt, die zum Beispiel später in Abschnitt 6.5 angewandt werden kann, um bestimmte Lowness-Eigenschaften des Graphisomorphie-Problems zu beweisen.

Wir beginnen mit der Definition der Funktionenklassen #P und GapP und der Erkundung ihrer grundlegenden Eigenschaften.

Definition 6.31 (#P und GapP). Für eine NPTM M und für ein Eingabewort x bezeichne $\text{acc}_M(x)$ die Anzahl der akzeptierenden Berechnungspfade von $M(x)$ und $\text{rej}_M(x)$ die Anzahl der ablehnenden Berechnungspfade von $M(x)$. Sowohl acc_M als auch rej_M bilden von Σ^* in \mathbb{N} ab. Definiere die Funktion $\text{gap}_M : \Sigma^* \rightarrow \mathbb{Z}$ durch

$$\text{gap}_M(x) = \text{acc}_M(x) - \text{rej}_M(x).$$

Einen solchen Funktionswert $\text{gap}_M(x)$ nennt man auch die Lücke von $M(x)$ (englisch als „gap“ bezeichnet), da dieser Wert die Differenz zwischen der Anzahl der akzeptierenden und der Anzahl der ablehnenden Pfade von M bei Eingabe x angibt.

Definiere die folgenden zwei Funktionenklassen:

$$\begin{aligned} \#P &= \{\text{acc}_M \mid M \text{ ist eine NPTM}\}; \\ \text{GapP} &= \{\text{gap}_M \mid M \text{ ist eine NPTM}\}. \end{aligned}$$

In Definition 6.31 wird nicht verlangt, dass die NPTMs normalisiert sein sollen. Das ist sinnvoll, denn da die Lücken normalisierter NP-Maschinen stets gerade Zahlen sind, wäre dies eine unnötige und schwer zu rechtfertigende Einschränkung.

Behauptung 6.32. 1. $\text{GapP} = \#P - \#P = \#P - \text{FP} = \text{FP} - \#P$, wobei sich das Minuszeichen auf die elementweise Subtraktion bezieht, nicht auf die mengentheoretische Differenz, d.h., für Funktionenklassen \mathcal{F} und \mathcal{G} definieren wir:

$$\mathcal{F} - \mathcal{G} = \{f - g \mid f \in \mathcal{F} \text{ und } g \in \mathcal{G}\}.$$

2. $\text{FP} \subseteq \#P \subseteq \text{GapP}$.

Der Beweis von Behauptung 6.32 wird dem Leser als Übung 6.11 überlassen.

#P und GapP besitzen gemeinsam einige nützliche Abschlusseigenschaften, wie etwa den Abschluss unter Addition und Multiplikation, die wir später benötigen. Andererseits ist GapP sogar unter Subtraktion abgeschlossen, eine Eigenschaft, die #P vermutlich nicht besitzt. Einige grundlegende Abschlusseigenschaften von GapP werden unten zusammengefasst. Der Beweis von Lemma 6.33 wird dem Leser als

Übung 6.11 überlassen. Diese Aufgabe fragt außerdem, welche der Abschlusseigenschaften von GapP auch für #P gelten. Von den in Lemma 6.33 angegebenen Abschlusseigenschaften ist vielleicht die unter Binomialkoeffizienten die am wenigsten offensichtliche.

Lemma 6.33 (Abschlusseigenschaften von GapP).

1. GapP ist unter Subtraktion abgeschlossen: Ist $f \in \text{GapP}$, so ist auch $-f \in \text{GapP}$.

2. Ist $f \in \text{GapP}$ und ist $p \in \mathbb{P}$, so sind die unten definierten Funktionen a und b in GapP:

$$a(x) = \sum_{|y| \leq p(|x|)} f(\langle x, y \rangle) \quad \text{und} \quad b(x) = \prod_{0 \leq y \leq p(|x|)} f(\langle x, y \rangle).$$

3. Gilt $f, g \in \text{GapP}$ und $0 \leq g(x) \leq p(|x|)$ für ein Polynom p , so sind die unten definierten Funktionen c, d und e in GapP:

$$c(x) = f(\langle x, g(x) \rangle) \quad \text{und} \quad d(x) = \binom{f(x)}{g(x)} \quad \text{und} \quad e(x) = f(x)^{g(x)}.$$

In Anmerkung 6.3 wurde PP durch #P-Funktionen charakterisiert. Alternativ dazu kann PP auch durch GapP-Funktionen charakterisiert werden; siehe Übung 6.9(a). In Definition 6.34 werden die Zählklassen $\oplus\mathbb{P}$, $\mathbb{C}=\mathbb{P}$ und SPP mittels GapP-Funktionen eingeführt. Diese sind Beispiele so genannter *gap-definierbarer* Komplexitätsklassen. Angesichts von Behauptung 6.32 können $\oplus\mathbb{P}$, $\mathbb{C}=\mathbb{P}$ und SPP ebenso durch #P-Funktionen charakterisiert werden, genau so, wie PP in Anmerkung 6.3 durch #P-Funktionen charakterisiert wurde; siehe Übung 6.9(b).

Definition 6.34 (Zählklassen). Definiere die folgenden Mengenklassen:

$$\mathbb{C}=\mathbb{P} = \{A \mid (\exists f \in \text{GapP}) (\forall x \in \Sigma^*) [x \in A \iff f(x) = 0]\};$$

$$\oplus\mathbb{P} = \{A \mid (\exists f \in \text{GapP}) (\forall x \in \Sigma^*) [x \in A \iff f(x) \equiv 1 \pmod{2}]\};$$

$$\text{SPP} = \left\{ A \left| \begin{array}{l} (\exists f \in \text{GapP}) (\forall x \in \Sigma^*) \\ [(x \in A \implies f(x) = 1) \wedge (x \notin A \implies f(x) = 0)] \end{array} \right. \right\}.$$

SPP ist ein Akronym für das englische „*Stoic Probabilistic Polynomial Time*“. Eine SPP-Maschine ist „stoisch“ in dem Sinn, dass für jede Eingabe ihre „Lücke“ – die Differenz zwischen der Anzahl ihrer akzeptierenden und der Anzahl ihrer ablehnenden Berechnungspfade bei dieser Eingabe – nur zwei der exponentiell vielen Werte hat, die eine NPTM ansonsten normalerweise haben kann. In dieser Hinsicht ähnelt SPP der Klasse UP, siehe Definition 3.81. Ersetzt man „GapP“ durch „#P“ in der Definition von SPP, so erhält man gerade UP, also ist SPP das GapP-Analogon von UP. Genau wie UP und anders als PP, $\oplus\mathbb{P}$ oder $\mathbb{C}=\mathbb{P}$ ist SPP eine Promise-Klasse, denn eine SPP-Maschine M „verspricht“, dass $\text{gap}_M(x)$ für alle x ein Wert in $\{0, 1\}$ ist; siehe auch Übung 6.10. SPP ist die kleinste „vernünftige“ gap-definierbare Zählklasse, wobei man eine Klasse hier genau dann vernünftig nennt, wenn sie sowohl die leere Menge als auch Σ^* enthält.

Die Inklusionen zwischen PP, $\text{C}=\text{P}$, $\oplus\text{P}$, SPP und anderen Klassen werden in Behauptung 6.35 angegeben. Sie sind leicht zu beweisen, siehe Übung 6.12. Einige Beweise (etwa der von $\text{C}=\text{P} \subseteq \text{PP}$) können durch den Gebrauch von GapP-Funktionen stark vereinfacht werden. Genau wie PP sind sowohl SPP als auch $\oplus\text{P}$ unter Komplement abgeschlossen, eine Eigenschaft, die $\text{C}=\text{P}$ vermutlich nicht teilt. Von keiner der in Behauptung 6.35 aufgeführten Inklusionen weiß man, ob sie echt ist.

Behauptung 6.35. 1. $\text{SPP} \subseteq \oplus\text{P}$.

2. $\text{P} \subseteq \text{UP} \subseteq \text{SPP} \subseteq \text{C}=\text{P}$ und $\text{P} \subseteq \text{coUP} \subseteq \text{SPP} \subseteq \text{coC}=\text{P}$.

3. $\text{P} \subseteq \text{coUP} \subseteq \text{coNP} \subseteq \text{C}=\text{P} \subseteq \text{PP}$ und $\text{P} \subseteq \text{UP} \subseteq \text{NP} \subseteq \text{coC}=\text{P} \subseteq \text{PP}$.

Der in Abschnitt 5.7 eingeführte Begriff der Lowness kann für jede relativierbare Komplexitätsklasse \mathcal{C} definiert werden. In der Notation von Definition 6.36 unten enthält Low_k , die k -te Stufe der Low-Hierarchie (siehe Definition 5.73), genau die NP-Mengen, die Σ_k^p -low sind. Und Satz 6.29 sagt, dass $\text{AM} \cap \text{coAM}$ Σ_2^p -low ist.

Definition 6.36 (Lowness). Sei \mathcal{C} eine beliebige relativierbare Komplexitätsklasse.

- Eine Menge A heißt \mathcal{C} -low, falls $\mathcal{C}^A = \mathcal{C}$ gilt.
- Eine Mengenklasse \mathcal{A} heißt \mathcal{C} -low, falls $\mathcal{A} \subseteq \{A \mid \mathcal{C}^A = \mathcal{C}\}$ gilt, d.h., jede Menge A in \mathcal{A} ist \mathcal{C} -low.

Der folgende Satz sagt, dass SPP genau die Mengen enthält, die GapP-low sind. Somit sind alle Mengen in SPP low für jede Klasse, die mittels GapP-Funktionen definierbar ist, wozu nicht nur SPP selbst, sondern auch PP, $\oplus\text{P}$ und $\text{C}=\text{P}$ zählen. Komplexitätsklassen \mathcal{C} , die für sich selbst low sind – für die also $\mathcal{C}^\mathcal{C} = \mathcal{C}$ gilt – heißen *selbst-low*.

Satz 6.37. $\text{SPP} = \{A \mid \text{GapP}^A = \text{GapP}\}$.

Beweis. Um zu beweisen, dass SPP alle Mengen enthält, die GapP-low sind, nehmen wir an, dass A eine beliebige Menge mit $\text{GapP}^A = \text{GapP}$ ist. Definiere eine NPOTM M , die bei Eingabe x ihr Orakel nach x fragt. Ist die Antwort „ja“ (gehört x also zur Orakelmenge), so akzeptiert M auf genau einem Pfad; andernfalls (wenn die Antwort nämlich „nein“ ist) erzeugt M einen akzeptierenden und einen ablehnenden Pfad. Somit ist, für jedes Orakel B , die Lücke von $M^B(x)$ gleich der charakteristischen Funktion von B an der Stelle x , d.h., $\text{gap}_{M^B} \equiv c_B$. Insbesondere ergibt sich für unsere Orakelmenge A :

$$\text{gap}_{M^A}(x) = \begin{cases} 1 & \text{falls } x \in A \\ 0 & \text{falls } x \notin A. \end{cases}$$

Da $\text{GapP}^A = \text{GapP}$ ist, gibt es eine NPTM N (ohne Orakel), so dass $\text{gap}_N \equiv \text{gap}_{M^A}$ gilt. Somit bezeugt N , dass A in SPP ist.

Umgekehrt müssen wir $\text{SPP} \subseteq \{A \mid \text{GapP}^A = \text{GapP}\}$ zeigen. Sei M eine gegebene NPOTM, und sei A eine beliebige Menge in SPP. Angenommen, M stellt bei Eingabe x auf jedem ihrer Pfade genau $k(|x|)$ Fragen, für ein geeignetes Polynom k und unabhängig davon, welches Orakel verwendet wird. Sei x beliebig gewählt, und sei $k = k(|x|)$. Weiter sei S eine NPTM, die $A \in \text{SPP}$ bezeugt, d.h., $\text{gap}_S \equiv c_A$.

Definiere eine NPTM N (ohne Orakel), für die $\text{gap}_N \equiv \text{gap}_{M^A}$ gilt, wie folgt:

Schritt 1: Bei Eingabe x rät N nichtdeterministisch ein Tupel $\mathbf{a} = (a_1, a_2, \dots, a_k)$ von Antwortbits: $a_i = 1$ für die Antwort „ja“ und $a_i = 0$ für die Antwort „nein“.

Schritt 2: N rät nichtdeterministisch einen Berechnungspfad α von $M^{(\cdot)}(x)$, wobei anstelle der Orakelbefragung die entsprechenden Antworten von \mathbf{a} eingesetzt werden, d.h., ist q_i die i -te Frage von $M^{(\cdot)}(x)$ auf α , so setzt $N(x)$ die Simulation von $M^{(\cdot)}(x)$ auf α gemäß der Antwort a_i fort (ohne ein Orakel zu fragen).

Schritt 3: An jeden solchen Pfad α hängt N einen Teilbaum an, der die Lücke G_α hat, die wie folgt definiert ist: Für jedes i mit $1 \leq i \leq k$ seien

$$g_i = \begin{cases} \text{gap}_S(q_i) & \text{falls } a_i = 1 \\ 1 - \text{gap}_S(q_i) & \text{falls } a_i = 0 \end{cases} \quad \text{und} \quad G_\alpha = \begin{cases} \prod_{i=1}^k g_i & \text{falls } \alpha \text{ akzeptiert} \\ -\prod_{i=1}^k g_i & \text{falls } \alpha \text{ ablehnt.} \end{cases}$$

Aus den Abschlusseigenschaften von GapP folgt, dass G_α in GapP ist, und daher kann N die entsprechende Lücke G_α für jeden Pfad α erzeugen. Für jedes i , $1 \leq i \leq k$, gilt dabei $g_i = 1$ genau dann, wenn a_i die korrekte Antwort ist, also genau dann, wenn $a_i = c_A(q_i)$ gilt. Somit ist, für jeden Pfad α , $G_\alpha \in \{-1, 1\}$, falls alle Fragen auf α korrekt beantwortet wurden, und andernfalls gilt $G_\alpha = 0$. Es folgt, dass kein Pfad mit inkorrekten Antworten einen Beitrag zur Lücke von N bei Eingabe x leisten kann, und die verbleibende Lücke entspricht dem Pfad, auf welchem alle Orakelantworten von $M^A(x)$ korrekt simuliert wurden. Somit ist $\text{gap}_N(x) = \text{gap}_{M^A}(x)$ für jedes x , und es gilt $\text{GapP}^A = \text{GapP}$. \square

Korollar 6.38. 1. $\text{GapP}^{\text{SPP}} = \text{GapP}$.

2. SPP ist unter \leq_T^P -Reduktionen abgeschlossen, d.h., $\text{P}^{\text{SPP}} = \text{SPP}$.
3. SPP ist PP-low, d.h., $\text{PP}^{\text{SPP}} = \text{PP}$.
4. SPP ist $\oplus\text{P}$ -low, d.h., $\oplus\text{P}^{\text{SPP}} = \oplus\text{P}$.
5. SPP ist C=P-low, d.h., $\text{C}=\text{P}^{\text{SPP}} = \text{C}=\text{P}$.
6. SPP ist selbst-low, d.h., $\text{SPP}^{\text{SPP}} = \text{SPP}$.

Der Beweis von Korollar 6.38 ist einfach und wird weggelassen, siehe jedoch Übung 6.13(a). Im Gegensatz zur ersten Aussage von Korollar 6.38 ist es unwahrscheinlich, dass $\#\text{PSPP} = \#\text{P}$ oder auch nur $\#\text{P}^{\text{UP}} = \#\text{P}$ gilt; siehe Übung 6.13(b).

Lemma 6.40 unten gibt einige nützliche Eigenschaften von SPP an, die in Abschnitt 6.5 angewandt werden. Auf den Beweis von Lemma 6.40 wird hier verzichtet; er ergibt sich im Wesentlichen aus der Selbst-Lowness von SPP (d.h. aus $\text{SPP}^{\text{SPP}} = \text{SPP}$) und Satz 6.39. Auch der Beweis von Satz 6.39, der dem von Satz 6.37 ähnelt (siehe Übung 6.14), wird hier weggelassen. Für eine NPTM N mit $L(N) = A$ und eine NPOTM (oder DPOTM) M sagen wir, M stellt N nur UP-Fragen, falls $\text{acc}_N(q) \leq 1$ für jede Eingabe x und alle Fragen q in der Berechnung von $M^A(x)$ gilt.

Satz 6.39. Sei $A \in \text{NP}$ mittels einer NPTM N , und sei M eine NPOTM, die N nur UP-Fragen stellt. Dann ist die Funktion $f(x) = \text{gap}_{M^A}(x)$ in GapP.

Lemma 6.40. Sei $A \in \text{NP}$ mittels einer NPTM N .

1. Ist $L \in \text{SPP}^A$ mittels einer NPOTM M , die N nur UP-Fragen stellt, so ist L in SPP.
2. Ist $f \in \text{FP}^A$ mittels einer DPOTM M , die N nur UP-Fragen stellt, so ist f in FP^{SPP} .

6.5 Graphisomorphie und Lowness

In diesem Abschnitt werden interessante Lowness-Eigenschaften des Graphisomorphie-Problems gezeigt. Insbesondere ist GI low für die zweite Stufe der Polynomialzeit-Hierarchie und auch für probabilistische und Zählklassen, einschließlich PP, C=P, ⊕P und SPP.

6.5.1 Graphisomorphie ist in der Low-Hierarchie

Wir zeigen zunächst, dass GI in Low_2 enthalten ist. Dieses Resultat liefert starke Indizien dafür, dass das Graphisomorphie-Problem nicht NP-vollständig ist. Warum? Angenommen, GI wäre NP-vollständig. Nach Satz 5.76 enthält High_0 alle in $\text{NP} \leq_T^p$ -vollständigen Mengen und deshalb insbesondere alle \leq_m^p -vollständigen NP-Mengen. Nach Satz 5.77 ist $\text{High}_0 \subseteq \text{High}_2$; also ist GI in High_2 . Andererseits ist jedoch $\text{Low}_2 \cap \text{High}_2$ nach Satz 5.77 genau dann leer, wenn die Polynomialzeit-Hierarchie auf Σ_2^p kollabiert, was für unwahrscheinlich gehalten wird.

Der Beweis von Satz 6.45 unten, nach welchem GI in Low_2 ist, erfordert als eine technische Vorbereitung das so genannte Hashing-Lemma, siehe Lemma 6.42. Hashing ist eine Methode, die in der Informatik für die dynamische Datenverwaltung verwendet wird. Jedem Datum wird eindeutig ein (kurzer) Schlüssel zugeordnet. Die Menge aller potenziellen Schlüssel, als Universum U bezeichnet, ist normalerweise sehr groß, wohingegen die Menge $V \subseteq U$ aller tatsächlich verwendeten Schlüssel viel kleiner sein kann. Eine *Hash-Funktion* $h : U \rightarrow T$ bildet die Elemente von U auf die *Hash-Tabelle* $T = \{0, 1, \dots, k - 1\}$ ab. Hash-Funktionen sind nicht injektiv, was bedeutet, dass verschiedene Schlüssel aus U auf ein und dieselbe Adresse in T abgebildet werden können. Wenn möglich, sollten je zwei verschiedene Schlüssel aus V jedoch auf verschiedene Adressen in T abgebildet werden. Das heißt, man versucht, Kollisionen auf der Menge der tatsächlich verwendeten Schlüssel zu vermeiden. Anders gesagt sollte eine Hash-Funktion nach Möglichkeit auf V injektiv sein.

Hashing ist auch eine sehr nützliche Technik in kryptographischen Anwendungen. Kryptographische Hash-Funktionen bilden gewöhnlich große Schlüssel auf kleinere Schlüssel in einer „sicheren“ Art und Weise ab.

Unter den verschiedenen Hash-Techniken ist das *universelle Hashing* für den Beweis von Satz 6.45 von besonderem Interesse. Universelles Hashing wurde 1979 von Carter und Wegman erfunden. Die Idee besteht darin, dass man sich nicht auf eine spezielle, konkrete Hash-Funktion einschränkt, sondern stattdessen *zufällig* eine solche aus einer geeigneten Familie von Hash-Funktionen wählt. Diese Hash-Technik ist in dem Sinne universell, dass sie nicht mehr von einer spezifischen Schlüsselmenge V abhängt; stattdessen wird versucht, Kollisionen auf *allen* hinreichend kleinen Mengen V mit einer hohen Wahrscheinlichkeit zu vermeiden. Die Wahrscheinlichkeit wird über die zufällige Wahl der Hash-Funktionen genommen.

Im Folgenden stellen wir uns Schlüssel als Wörter über dem Alphabet $\Sigma = \{0, 1\}$ vor, und wir bezeichnen mit Σ^n die Menge aller Wörter der Länge n in Σ^* .

Definition 6.41 (Universelles Hashing). Sei $\Sigma = \{0, 1\}$, und seien m und t , $t > m$, natürliche Zahlen. Eine Hash-Funktion $h : \Sigma^t \rightarrow \Sigma^m$ ist eine lineare Abbildung, die durch eine boolesche $(m \times t)$ -Matrix $B_h = (b_{i,j})_{i,j}$ bestimmt wird, wobei $b_{i,j} \in \{0, 1\}$.

Für $\mathbf{x} \in \Sigma^t$ und für jedes j mit $1 \leq j \leq m$ ist das j -te Bit von $\mathbf{y} = h(\mathbf{x}) \in \Sigma^m$ gegeben durch

$$y_j = (b_{1,j} \wedge x_1) \oplus (b_{2,j} \wedge x_2) \oplus \cdots \oplus (b_{t,j} \wedge x_t).$$

Hier bezeichnet \oplus die logische Operation Exklusives-Oder (alias die Paritätsoperation), siehe Definition 4.18 und Tabelle 4.14 in Abschnitt 4.2. Weil \oplus assoziativ ist, können wir schreiben:

$$a_1 \oplus a_2 \oplus \cdots \oplus a_n = 1 \iff \|\{i \mid a_i = 1\}\| \equiv 1 \pmod{2}.$$

Sei $\mathcal{H}_{t,m} = \{h : \Sigma^t \rightarrow \Sigma^m \mid B_h \text{ ist eine boolesche } (m \times t)\text{-Matrix}\}$ eine Familie von Hash-Funktionen mit den Parametern t und m . Wir nehmen auf $\mathcal{H}_{t,m}$ die Gleichverteilung an: Eine Hash-Funktion h wird aus $\mathcal{H}_{t,m}$ gezogen, indem die Bits $b_{i,j}$ in B_h unabhängig und gleichverteilt gewählt werden.

Sei $V \subseteq \Sigma^t$. Für eine Teilfamilie $\widehat{\mathcal{H}}$ von $\mathcal{H}_{t,m}$ gibt es eine Kollision auf V , falls

$$(\exists \mathbf{v} \in V) (\forall h \in \widehat{\mathcal{H}}) (\exists \mathbf{x} \in V) [\mathbf{v} \neq \mathbf{x} \wedge h(\mathbf{v}) = h(\mathbf{x})]$$

gilt. Andernfalls heißt $\widehat{\mathcal{H}}$ kollisionsfrei auf V .

Eine Kollision auf V bedeutet, dass die Injektivität einer jeden Hash-Funktion aus der Teilstammfamilie $\widehat{\mathcal{H}}$ auf V gestört ist. Lemma 6.42 sagt, dass auf jeder hinreichend kleinen Menge V eine zufällig gewählte Teilstammfamilie von $\mathcal{H}_{t,m}$ mit hoher Wahrscheinlichkeit kollisionsfrei ist. Ist V jedoch zu groß, dann tritt eine Kollision unvermeidlich ein.

Lemma 6.42 (Hashing-Lemma). Für die fest gewählten Parameter $t, m \in \mathbb{N}$ sei $V \subseteq \Sigma^t$ und sei $\widehat{\mathcal{H}} = (h_1, h_2, \dots, h_{m+1})$ eine Familie von Hash-Funktionen, die zufällig unter Gleichverteilung aus $\mathcal{H}_{t,m}$ gezogen werden. Definiere das Kollisionsprädikat als

$$\text{Col}(V) = \{\widehat{\mathcal{H}} \mid (\exists \mathbf{v} \in V) (\forall h \in \widehat{\mathcal{H}}) (\exists \mathbf{x} \in V) [\mathbf{v} \neq \mathbf{x} \wedge h(\mathbf{v}) = h(\mathbf{x})]\}.$$

Das heißt, $\text{Col}(V)$ ist das Ereignis, dass für die gegebene Teilstammfamilie $\widehat{\mathcal{H}}$ eine Kollision auf V eintritt. Dann gelten die folgenden beiden Aussagen:

1. Ist $\|V\| \leq 2^{m-1}$, so tritt $\text{Col}(V)$ mit Wahrscheinlichkeit höchstens $1/4$ ein.
2. Ist $\|V\| > (m+1)2^m$, so tritt $\text{Col}(V)$ mit Wahrscheinlichkeit 1 ein.

Der Beweis von Lemma 6.42 wird dem Leser als Übung 6.15 überlassen.

Die folgende Definition gibt dem Komplement des Graphisomorphie-Problems einen eigenen Namen.

Definition 6.43 (Graph-Nichtisomorphie-Problem). Definiere das Graph-Nichtisomorphie-Problem als

$$\text{GNI} = \{\langle G, H \rangle \mid G \text{ und } H \text{ sind nicht isomorphe Graphen}\}.$$

Die Arthur-Merlin-Hierarchie wurde in Definition 6.25 eingeführt, und Satz 6.27 sagte, dass diese Hierarchie auf AM kollabiert. Wir zeigen nun, dass das Graph-Nichtisomorphie-Problem in AM liegt. Anders gesagt ist GI in coAM. Nach Korollar 6.30 liegt jede NP-Menge aus coAM in Low₂. Es folgt, dass GI zu Low₂ gehört.

Lemma 6.44. GNI ist in AM.

Beweis. Seien G und H zwei Graphen mit jeweils n Knoten. Wir wollen das Hashing-Lemma anwenden. Auf den ersten Blick scheint es dafür sinnvoll zu sein, die direkt vor Lemma 2.53 in Abschnitt 2.4.3 definierte Menge

$$A(G, H) = \{\langle F, \varphi \rangle \mid F \cong G \wedge \varphi \in \text{Aut}(F)\} \cup \{\langle F, \varphi \rangle \mid F \cong H \wedge \varphi \in \text{Aut}(F)\}$$

als das V aus Lemma 6.42 zu verwenden. Nach Lemma 2.53 gilt $\|A(G, H)\| = n!$, falls G und H isomorph sind, und $\|A(G, H)\| = 2n!$, falls G und H nicht isomorph sind. Die AM-Maschine für GNI, die unten definiert werden soll, muss jedoch natürlich polynomialzeit-beschränkt sein. Diese Schranke erfordert, dass die Parameter t und m aus dem Hashing-Lemma polynomiell in n sein müssen. Um dieses Lemma aber anwenden zu können, müssten wir ein Polynom $m = m(n)$ so wählen, dass gilt:

$$n! \leq 2^{m-1} < (m+1)2^m < 2n!, \quad (6.19)$$

denn andernfalls wäre die Menge $V = A(G, H)$ nicht groß genug, um zwei isomorphe Graphen G und H gemäß Lemma 6.42 mit hinreichend hoher Wahrscheinlichkeit von zwei nicht isomorphen Graphen unterscheiden zu können. Doch leider ist es nicht möglich, ein Polynom m zu finden, das (6.19) erfüllt.

Deshalb wählen wir eine andere Menge als $A(G, H)$ für unser V aus Lemma 6.42, nämlich eine Menge, die eine Lücke zwischen der oberen und der unteren Schranke in (6.19) schafft, die groß genug ist, um ein Paar isomorpher Graphen von einem Paar nicht isomorpher Graphen zu unterscheiden. Definiere

$$V = A(G, H)^n = \underbrace{A(G, H) \times A(G, H) \times \cdots \times A(G, H)}_{n\text{-mal}}.$$

Nun wird (6.19) zu:

$$(n!)^n \leq 2^{m-1} < (m+1)2^m < (2n!)^n, \quad (6.20)$$

und diese Ungleichung kann erfüllt werden, indem man $m = m(n) = 1 + \lceil n \log n \rceil$ setzt, welches eine in n polynomielle Funktion ist.

Eine AM-Maschine M für GNI wird nun folgendermaßen definiert. Seien zwei Graphen G und H mit jeweils n Knoten gegeben. M beginnt mit der Berechnung des Parameters $m = m(n)$. Die Menge $V = A(G, H)^n$ enthält n -Tupel von Paaren

der Form $\langle F, \varphi \rangle$, wobei F ein Graph mit n Knoten und φ eine Permutation aus der Automorphismengruppe $\text{Aut}(F)$ ist. Die Elemente von V können geeignet als Wörter über dem Alphabet $\Sigma = \{0, 1\}$ codiert werden, wobei $t = t(n)$ ein geeignetes Polynom ist. Bis zu diesem Punkt sind alle Berechnungen deterministisch.

Dann macht M einen probabilistischen Zug Arthurs: Sie wählt zufällig eine Familie $\widehat{\mathcal{H}} = (h_1, h_2, \dots, h_{m+1})$ von Hash-Funktionen unter Gleichverteilung. Jede solche Hash-Funktion $h_i \in \widehat{\mathcal{H}}$ ist durch eine boolesche $(m \times t)$ -Matrix gegeben, deren Einträge unabhängig und gleichverteilt gewählt werden. Die $m + 1$ Hash-Funktionen $h_i \in \widehat{\mathcal{H}}$ können dann durch ein Wort $r(\widehat{\mathcal{H}}) \in \Sigma^*$ der Länge $p(n)$ codiert werden, für ein geeignetes Polynom p . Nun wird das Kollisionsprädikat $\text{Col}(V)$ aus Lemma 6.42 folgendermaßen modifiziert:

$$B = \left\{ \langle G, H, r(\widehat{\mathcal{H}}) \rangle \middle| \begin{array}{l} G \text{ und } H \text{ sind Graphen mit je } n \text{ Knoten}, |r(\widehat{\mathcal{H}})| = p(n) \text{ und} \\ (\exists v \in V) (\forall i : 1 \leq i \leq m+1) (\exists x \in V) [v \neq x \wedge h_i(v) = h_i(x)] \end{array} \right\}.$$

Da der \forall -Quantor in B über nur polynomiell viele i quantifiziert, kann er deterministisch in Polynomialzeit ausgewertet werden. Somit können die beiden \exists -Quantoren in B zu nur *einem* polynomiell längenbeschränkten \exists -Quantor zusammengefasst werden. Nach Satz 5.31 ist B eine Menge in $\Sigma_1^p = \text{NP}$. Sei N eine NPTM für B . Ist $r(\widehat{\mathcal{H}})$ das zufällig gewählte Wort, das $m + 1$ unabhängige und gleichverteilte Hash-Funktionen aus $\mathcal{H}_{t,m}$ codiert, so entspricht die Simulation der Berechnung von $N(\langle G, H, r(\widehat{\mathcal{H}}) \rangle)$ einem Zug Merlins. Damit ist M vollständig beschrieben.

Angenommen, G und H sind nicht isomorph. Nach Lemma 2.53 gilt $\|A(G, H)\| = 2n!$. Ungleichung (6.20) impliziert $\|V\| = (2n!)^n > (m+1)2^m$. Nach Lemma 6.42 ist $\langle G, H, r(\widehat{\mathcal{H}}) \rangle$ mit Wahrscheinlichkeit eins in B , d.h., eine Kollision tritt mit Sicherheit ein. Somit gibt es für jede Wahl von $r(\widehat{\mathcal{H}})$ einen akzeptierenden Berechnungspfad von $N(\langle G, H, r(\widehat{\mathcal{H}}) \rangle)$.

Sei nun angenommen, dass G und H isomorph sind. Nach Lemma 2.53 gilt $\|A(G, H)\| = n!$. Ungleichung (6.20) impliziert $\|V\| = (n!)^n \leq 2^{m-1}$. Nach Lemma 6.42 ist $\langle G, H, r(\widehat{\mathcal{H}}) \rangle$ mit Wahrscheinlichkeit höchstens $1/4$ in B . Somit gilt für mehr als $3/4$ der möglichen Wahlen von $r(\widehat{\mathcal{H}})$, dass $N(\langle G, H, r(\widehat{\mathcal{H}}) \rangle)$ keinen akzeptierenden Berechnungspfad hat. Also ist GNI in $(\exists^+ \exists | \exists^+ \forall) = \text{AM}$ und das Lemma bewiesen. \square

Satz 6.45 (Schöning). GI ist in Low_2 .

Beweis. Nach Korollar 6.30 gilt $\text{NP} \cap \text{coAM} \subseteq \text{Low}_2$. Aus Lemma 6.44 folgt, dass GI in $\text{NP} \cap \text{coAM}$ ist. Somit ist GI in Low_2 . \square

Korollar 6.46. GI ist in keiner der Klassen High_k enthalten, außer wenn die Polynomialzeit-Hierarchie auf $\Sigma_{\max(k,2)}^p$ kollabieren würde. Insbesondere ist GI nicht \leq_m^p -vollständig in NP, außer es würde $\text{PH} = \Sigma_2^p$ gelten.

Beweis. Das Argument wurde im ersten Absatz von Abschnitt 6.5.1 gegeben. \square

6.5.2 Graphisomorphie ist in SPP

Nun beweisen wir, dass das Graphisomorphie-Problem in SPP ist. Nach Korollar 6.38 ist GI somit low für PP, \oplus P, $\mathbb{C}=\mathbb{P}$ und SPP. Um dieses Resultat zu beweisen, das später als Satz 6.50 angegeben wird, zeigen wir zunächst, dass die lexikographisch kleinste Permutation in einer rechten co-Menge effizient berechnet werden kann. Es ist an dieser Stelle zweckmäßig, sich die in den Abschnitten 2.4.2 und 2.4.3 definierten Begriffe wieder ins Gedächtnis zu rufen. Dort wurde zum Beispiel gezeigt, dass die Menge $\text{Iso}(G, H)$ der Isomorphismen zweier isomorpher Graphen G und H eine rechte co-Menge der Automorphismengruppe $\text{Aut}(G)$ in der Permutationsgruppe \mathfrak{S}_n ist. In Symbolen: $\text{Iso}(G, H) = \text{Aut}(G)\tau$, wobei $\tau \in \text{Iso}(G, H)$. Auch werden hier die Begriffe aus Definition 2.47 in Abschnitt 2.4 sowie Gleichung (2.13) gebraucht. Die lexikographische Ordnung auf \mathfrak{S}_n wurde in Beispiel 5.26 definiert.

Satz 6.47. *Sei $\mathfrak{G} \leq \mathfrak{S}_n$ eine Permutationsgruppe, repräsentiert durch einen Generator G (d.h., $\mathfrak{G} = \langle G \rangle$), und sei $\pi \in \mathfrak{S}_n$ eine Permutation. Es gibt einen deterministischen Polynomialzeit-Algorithmus, der bei Eingabe von G und π die lexikographisch kleinste Permutation in der rechten co-Menge $\mathfrak{G}\pi$ von \mathfrak{G} in \mathfrak{S}_n bestimmt.*

Beweis. Sei G ein Generator der Permutationsgruppe $\mathfrak{G} \leq \mathfrak{S}_n$ (d.h., $\mathfrak{G} = \langle G \rangle$), und sei $\pi \in \mathfrak{S}_n$ eine Permutation. Abbildung 6.5 zeigt den Algorithmus LERC: Sind G und π gegeben, so berechnet LERC die lexikographisch kleinste Permutation in der rechten co-Menge $\mathfrak{G}\pi$ von \mathfrak{G} in \mathfrak{S}_n .

```

LERC( $G, \pi$ ) {
    Berechne den Turm  $\mathfrak{G}^{(n)} \leq \mathfrak{G}^{(n-1)} \leq \dots \leq \mathfrak{G}^{(1)} \leq \mathfrak{G}^{(0)}$  von Stabilisatoren in  $\mathfrak{G}$ ;
     $\varphi_0 = \pi$ ;
    for ( $i = 0, 1, \dots, n - 1$ ) {
         $x := i + 1$ ;
        Berechne das Element  $y$  im Orbit  $\mathfrak{G}^{(i)}(x)$ , für das  $\varphi_i(y)$  minimal ist;
        Berechne eine Permutation  $\tau_i$  in  $\mathfrak{G}^{(i)}$  mit  $\tau_i(x) = y$ ;
         $\varphi_{i+1} := \tau_i \varphi_i$ ;
    }
    return  $\varphi_n$ ;
}

```

Abb. 6.5. Algorithmus LERC

Nach Satz 2.48 kann der Turm $\mathbf{id} = \mathfrak{G}^{(n)} = \mathfrak{G}^{(n-1)} \leq \dots \leq \mathfrak{G}^{(1)} \leq \mathfrak{G}^{(0)} = \mathfrak{G}$ von Stabilisatoren von \mathfrak{G} in Polynomialzeit berechnet werden. Genauer gesagt bestimmt der Algorithmus für jedes i mit $1 \leq i \leq n$ die vollständigen rechten Transversalen T_i von $\mathfrak{G}^{(i)}$ in $\mathfrak{G}^{(i-1)}$ und somit einen starken Generator $S = \bigcup_{i=1}^{n-1} T_i$ von \mathfrak{G} .

Da $\varphi_0 = \pi$ und $\mathfrak{G}^{(n-1)} = \mathfrak{G}^{(n)} = \mathbf{id}$ gilt, genügt es für den Nachweis der Korrektheit von LERC zu zeigen, dass für jedes i mit $0 \leq i \leq n - 1$ die lexikogra-

phisch kleinste Permutation von $\mathfrak{G}^{(i)}\varphi_i$ in $\mathfrak{G}^{(i+1)}\varphi_{i+1}$ enthalten ist. Mit Induktion folgt daraus, dass $\mathfrak{G}^{(n)}\varphi_n = \{\varphi_n\}$ die lexikographisch kleinste Permutation von $\mathfrak{G}\pi = \mathfrak{G}^{(0)}\varphi_0$ enthält. Also gibt der Algorithmus LERC dann tatsächlich die lexikographisch kleinste Permutation von $\mathfrak{G}\pi$ aus.

Für eine Permutationsgruppe $\mathfrak{H} \leq \mathfrak{S}_n$ bezeichne $\mathfrak{H}(x)$ den Orbit des Elements $x \in [n]$ in \mathfrak{H} . Um die obige Behauptung zu beweisen, sei τ_i die Permutation in $\mathfrak{G}^{(i)}$, die $i+1$ auf das Element y im Orbit $\mathfrak{G}^{(i)}(i+1)$ abbildet, für das $\varphi_i(y) = x$ das kleinste Element in der Menge $\{\varphi_i(z) \mid z \in \mathfrak{G}^{(i)}(i+1)\}$ ist. Nach Satz 2.48 kann der Orbit $\mathfrak{G}^{(i)}(i+1)$ in Polynomialzeit berechnet werden, und da $\mathfrak{G}^{(i)}(i+1)$ höchstens $n-i$ Elemente enthält, kann y effizient bestimmt werden. Der Algorithmus ist so entworfen worden, dass $\varphi_{i+1} = \tau_i\varphi_i$ gilt. Da jede Permutation aus $\mathfrak{G}^{(i)}$ jedes Element von $[i]$ auf sich selbst abbildet und da $\tau_i \in \mathfrak{G}^{(i)}$ gilt, folgt für jedes j mit $1 \leq j \leq i$, für jedes $\tau \in \mathfrak{G}^{(i)}$ und für jedes $\sigma \in \mathfrak{G}^{(i+1)}$:

$$(\sigma\varphi_{i+1})(j) = \varphi_{i+1}(j) = (\tau_i\varphi_i)(j) = \varphi_i(j) = (\tau\varphi_i)(j).$$

Insbesondere ergibt sich für die lexikographisch kleinste Permutation μ in $\mathfrak{G}^{(i)}\varphi_i$, dass jede Permutation aus $\mathfrak{G}^{(i+1)}\varphi_{i+1}$ mit μ in den ersten i Elementen übereinstimmt, d.h., sie stimmt mit μ auf $[i]$ überein. Außerdem gilt für jedes $\sigma \in \mathfrak{G}^{(i+1)}$ und für das oben definierte Element $x = \varphi_i(y)$:

$$(\sigma\varphi_{i+1})(i+1) = \varphi_{i+1}(i+1) = (\tau_i\varphi_i)(i+1) = x.$$

Es ist leicht zu sehen, dass $\mathfrak{G}^{(i+1)}\varphi_{i+1} = \{\varphi \in \mathfrak{G}^{(i)}\varphi_i \mid \varphi(i+1) = x\}$ gilt. Die Behauptung folgt nun aus der Tatsache, dass $\mu(i+1) = x$ für die lexikographisch kleinste Permutation μ von $\mathfrak{G}^{(i)}\varphi_i$ gilt. Wir haben also gezeigt, dass der Algorithmus LERC effizient und korrekt arbeitet. \square

Satz 6.47 kann leicht zu Korollar 6.48 verallgemeinert werden, siehe Übung 6.16.

Korollar 6.48. *Sei $\mathfrak{G} \leq \mathfrak{S}_n$ eine Permutationsgruppe, repräsentiert durch einen Generator G (d.h., $\mathfrak{G} = \langle G \rangle$), und seien π und ψ zwei Permutationen aus \mathfrak{S}_n . Es gibt einen deterministischen Polynomialzeit-Algorithmus, der bei Eingabe von $\langle G, \pi, \psi \rangle$ die lexikographisch kleinste Permutation in $\psi\mathfrak{G}\pi$ bestimmt.*

Nun definieren wir ein Problem, das im Beweis von Satz 6.50 benötigt wird; siehe auch die Begriffe in Abschnitt 2.4, insbesondere die Definitionen 2.47 und 2.49.

Definition 6.49. *Definiere das funktionale Problem `auto` wie folgt: Gegeben ein Graph G , berechne einen starken Generator der Automorphismengruppe $\text{Aut}(G)$.*

Nach einem Resultat von Mathon [Mat79] sind die Probleme `auto` und `GI` \leq_T^P -äquivalent; siehe Übung 6.17. Das heißt, es gilt $\text{auto} \in \text{FP}^{\text{GI}}$ und $\text{GI} \in \text{P}^{\text{auto}}$.

Nach diesen Vorbereitungen kann Satz 6.50 bewiesen werden, das Hauptergebnis in diesem Abschnitt.

Satz 6.50 (Arvind und Kurur). *GI ist in SPP.*

Beweis. Um den Satz zu beweisen, genügt es, $\text{auto} \in \text{FP}^{\text{SPP}}$ zu zeigen. Warum? Angenommen, es gilt $\text{auto} \in \text{FP}^{\text{SPP}}$. Wie oben erwähnt sind auto und $\text{GI} \leq_T^p$ -äquivalent, also ist GI in $\text{P}^{\text{auto}} \subseteq \text{P}^{\text{FP}^{\text{SPP}}} = \text{P}^{\text{SPP}}$. Nach Korollar 6.38 ist SPP unter \leq_T^p -Reduktionen abgeschlossen, d.h., $\text{P}^{\text{SPP}} = \text{SPP}$. Daraus folgt, dass GI in SPP ist.

Wir müssen also einen FP^{SPP} -Algorithmus für auto entwerfen. Für einen gegebenen Graphen G berechnet dieser Algorithmus einen starken Generator $S = \bigcup_{i=0}^{n-1} T_i$ von $\text{Aut}(G)$, wobei

$$\mathbf{id} = \text{Aut}(G)^{(n)} \leq \text{Aut}(G)^{(n-1)} \leq \cdots \leq \text{Aut}(G)^{(1)} \leq \text{Aut}(G)^{(0)} = \text{Aut}(G)$$

der Turm von Stabilisatoren von $\text{Aut}(G)$ ist und T_i die vollständige rechte Transversale von $\text{Aut}(G)^{(i+1)}$ in $\text{Aut}(G)^{(i)}$, $0 \leq i < n$.

Ausgehend vom trivialen Fall, $\text{Aut}(G)^{(n)} = \mathbf{id}$, konstruieren wir Schritt für Schritt einen starken Generator von $\text{Aut}(G)^{(i)}$, wobei i fallend ist, so dass wir schließlich einen solchen für $\text{Aut}(G)^{(0)} = \text{Aut}(G)$ erhalten werden. Nehmen wir also an, wir haben bereits einen starken Generator $S_i = \bigcup_{j=i}^{n-1} T_j$ von $\text{Aut}(G)^{(i)}$ gefunden. Unten wird beschrieben, wie der FP^{SPP} -Algorithmus dann eine vollständige rechte Transversale T_{i-1} von $\text{Aut}(G)^{(i)}$ in $\text{Aut}(G)^{(i-1)}$ bestimmt. Definiere zu diesem Zweck die folgende Menge:

$$A = \left\{ \langle G, S, i, j, \pi \rangle \mid \begin{array}{l} S \subseteq \text{Aut}(G) \text{ und } \langle S \rangle \text{ ist ein punktweiser Stabilisator von } [i] \\ \text{in } \text{Aut}(G), \pi \text{ ist eine partielle Permutation, die punktweise } [i-1] \text{ stabilisiert und } \pi(i) = j \text{ erfüllt, und es gibt ein } \tau \in \\ \text{Aut}(G)^{(i-1)}, \text{ so dass } \tau(i) = j \text{ und LERC}(S, \tau) \text{ erweitert } \pi \end{array} \right\}.$$

Nach Satz 6.47 kann die lexikographisch kleinste Permutation $\text{LERC}(S, \tau)$ der rechten co-Menge $\langle S \rangle \tau$ durch den Algorithmus aus Abbildung 6.5 in Polynomialzeit berechnet werden. Die partielle Permutation π ist Teil der Eingabe $\langle G, S, i, j, \pi \rangle$, da die Menge A als ein Orakel in einer Präfixsuche nach der lexikographisch kleinsten Permutation $\tau \in \text{Aut}(G)^{(i-1)}$ mit $\tau(i) = j$ verwendet werden soll. (Ein anderer Algorithmus, der eine Präfixsuche durchführt, ist in Abbildung 5.9 in Beispiel 5.26 dargestellt.)

Abbildung 6.6 zeigt eine NPTM N für die Orakelmenge A . Demnach ist A in NP. Wie man sieht, folgt aus $\tau(i) = j$, dass $\sigma(i) = j$ für jede Permutation σ in der rechten co-Menge $\langle S \rangle \tau$ gilt.

Wir zeigen nun, dass die Anzahl der akzeptierenden Berechnungspfade von N bei Eingabe $\langle G, S, i, j, \pi \rangle$ entweder 0 oder 1 ist, falls $\langle S \rangle = \text{Aut}(G)^{(i)}$ gilt. Im Allgemeinen (unabhängig davon, ob $\langle S \rangle = \text{Aut}(G)^{(i)}$ gilt oder nicht) ist $\text{acc}_N(\langle G, S, i, j, \pi \rangle)$ entweder 0 oder

$$k = \frac{\|\text{Aut}(G)^{(i)}\|}{\|\langle S \rangle\|}.$$

Angenommen, $\langle G, S, i, j, \pi \rangle$ ist in A und $\langle S \rangle = \text{Aut}(G)^{(i)}$. Falls $\tau(i) = j$ für ein $\tau \in \text{Aut}(G)^{(i-1)}$ und $j > i$ gilt, dann besteht die rechte co-Menge $\langle S \rangle \tau$ aus genau den Permutationen in $\text{Aut}(G)^{(i-1)}$, die i auf j abbilden. Folglich entspricht

```

 $N(\langle G, S, i, j, \pi \rangle) \{$ 
    Verifiziere  $S \subseteq \text{Aut}(G)^{(i)}$ ;
    Rate nichtdeterministisch eine Permutation  $\tau \in \mathfrak{S}_n$ ; //  $G$  hat  $n$  Knoten
    if ( $\tau \in \text{Aut}(G)^{(i-1)}$  und  $\tau(i) = j$  und  $\tau$  erweitert  $\pi$  und  $\tau = \text{LERC}(S, \tau)$ )
        akzeptiere und halte;
    else lehne ab und halte;
}

```

Abb. 6.6. NP-Maschine N für die Orakelmenge A

der einzige akzeptierende Berechnungspfad von $N(\langle G, S, i, j, \pi \rangle)$ der eindeutig bestimmten lexikographisch kleinsten Permutation $\tau = \text{LERC}(S, \tau)$. Ist $\langle S \rangle$ jedoch eine echte Untergruppe von $\text{Aut}(G)^{(i)}$, dann kann $\text{Aut}(G)^{(i)} \tau$ als die disjunkte Vereinigung von k rechten co-Mengen von $\langle S \rangle$ geschrieben werden. Im Allgemeinen hat $N(\langle G, S, i, j, \pi \rangle)$ also k akzeptierende Berechnungspfade, falls $\langle G, S, i, j, \pi \rangle$ zu A gehört, und andernfalls hat $N(\langle G, S, i, j, \pi \rangle)$ keinen akzeptierenden Pfad.

Abbildung 6.7 zeigt den FP^A -Algorithmus M^A , der auto berechnet.⁴ Diese DPOTM M stellt nur solche Fragen $q = \langle G, S_i, i, j, \pi \rangle$ an ihr Orakel A , für die $\langle S_i \rangle = \text{Aut}(G)^{(i)}$ gilt. Folglich ist $\text{acc}_N(q) \leq 1$ für jede Frage q , die in dieser Berechnung tatsächlich gestellt wird. Aus Aussage 2 von Lemma 6.40 folgt $\text{auto} \in \text{FP}^{\text{SPP}}$.

Dass die Ausgabe S_0 von $M^A(G)$ ein starker Generator von $\text{Aut}(G) = \text{Aut}(G)^{(0)}$ ist, kann durch Induktion über n gezeigt werden. Der Induktionsanfang ist $n = 1$, und $\text{Aut}(G)^{(n-1)} = \mathbf{id}$ wird natürlich von $S_{n-1} = \{\mathbf{id}\}$ erzeugt. Im Induktionsschritt sei angenommen, dass zu Beginn der Iteration i (siehe die for-Schleife in Abbildung 6.7) ein starker Generator S_i von $\text{Aut}(G)^{(i)}$ bereits gefunden ist. Wir zeigen, dass die Menge $S_{i-1} = S_i \cup T_{i-1}$ am Ende von Iteration i ein starker Generator von $\text{Aut}(G)^{(i-1)}$ ist. Für jedes j mit $i+1 \leq j \leq n$ überprüft die Orakelfrage „ $\langle G, S_i, i, j, \hat{\pi} \rangle \in A?$ “, ob es eine Permutation in $\text{Aut}(G)^{(i-1)}$ gibt, die i auf j abbildet.

Die sich anschließende Präfixsuche, in der geeignete Fragen an das Orakel A gestellt werden, konstruiert die lexikographisch kleinste Permutation $\hat{\pi}$ in $\text{Aut}(G)^{(i-1)}$ mit $\hat{\pi}(i) = j$. Wie oben erwähnt wird A nur über Wörter q mit $\text{acc}_N(q) \leq 1$ befragt, da S_i ein starker Generator für $\text{Aut}(G)^{(i)}$ ist, d.h., da $\langle S_i \rangle = \text{Aut}(G)^{(i)}$ gilt. Nach Konstruktion ist T_{i-1} , am Ende von Iteration i , eine vollständige rechte Transversale von $\text{Aut}(G)^{(i)}$ in $\text{Aut}(G)^{(i-1)}$. Folglich ist $S_{i-1} = S_i \cup T_{i-1}$ ein starker Generator für $\text{Aut}(G)^{(i-1)}$. Schließlich, nach n Iterationen, ist ein starker Generator S_0 für $\text{Aut}(G) = \text{Aut}(G)^{(0)}$ bestimmt worden. □

Korollar 6.38 und Satz 6.50 haben die folgende Konsequenz.

Korollar 6.51. GI ist low für SPP, PP, C=P und $\oplus P$, d.h., es gilt $\text{SPP}^{\text{GI}} = \text{SPP}$, $\text{PP}^{\text{GI}} = \text{PP}$, $\text{C}=\text{P}^{\text{GI}} = \text{C}=\text{P}$ und $\oplus \text{P}^{\text{GI}} = \oplus \text{P}$.

⁴ Eine Bemerkung zur äußeren for-Schleife in Abbildung 6.7: Für $i = 1$ ist (in der Notation von Beispiel 5.26) $\pi = \underbrace{* \dots *}_n$ die leere Permutation.

```

 $M^A(G) \{$ 
    Setze  $T_i := \{\text{id}\}$  für jedes  $i, 0 \leq i \leq n - 2$ ; //  $G$  ist ein Graph mit  $n$  Knoten
    //  $T_i$  soll eine vollständige rechte Transversale von  $\text{Aut}(G)^{(i+1)}$  in  $\text{Aut}(G)^{(i)}$  werden
    Setze  $S_i := \emptyset$  für jedes  $i, 0 \leq i \leq n - 2$ ;
    Setze  $S_{n-1} := \{\text{id}\}$ ; //  $S_i$  soll ein starker Generator von  $\text{Aut}(G)^{(i)}$  werden
    for ( $i = n - 1, n - 2, \dots, 1$ ) {
        // vor Iteration  $i$  ist  $S_i$  bereits gefunden und  $S_{i-1}$  soll berechnet werden
        Sei  $\pi : [i-1] \rightarrow [n]$  die partielle Permutation mit  $\pi(a) = a$  für jedes  $a \in [i-1]$ 
        for ( $j = i + 1, i + 2, \dots, n$ ) {
            Setze  $\hat{\pi} := \pi j$ ; //  $\hat{\pi}$  erweitert  $\pi$  um das Paar  $(i, j)$ , d.h.,  $\hat{\pi}(i) = j$ 
            if ( $\langle G, S_i, i, j, \hat{\pi} \rangle \in A$ ) {
                // Präfixsuche findet die kleinste Permutation in  $\text{Aut}(G)^{(i-1)}$ , die  $i$  auf  $j$  abbildet
                for ( $k = i + 1, i + 2, \dots, n$ ) {
                    Finde Element  $\ell$ , so dass  $\ell$  nicht im Bild von  $\hat{\pi}$  ist und  $\langle G, S_i, i, j, \hat{\pi} \ell \rangle \in A$ ;
                     $\hat{\pi} := \hat{\pi} \ell$ ;
                }
                //  $\hat{\pi}$  ist nun eine totale Permutation in  $\mathfrak{S}_n$ 
                 $T_{i-1} := T_{i-1} \cup \hat{\pi}$ ;
            }
            }
            //  $T_{i-1}$  ist nun eine vollständige rechte Transversale von  $\text{Aut}(G)^{(i)}$ 
            in  $\text{Aut}(G)^{(i-1)}$ 
             $S_{i-1} := S_i \cup T_{i-1}$ ;
        }
        return  $S_0$ ; //  $S_0$  ist ein starker Generator von  $\text{Aut}(G) = \text{Aut}(G)^{(0)}$ 
    }
}

```

Abb. 6.7. FP^{SPP}-Algorithmus zur Berechnung eines starken Generators von $\text{Aut}(G)$

6.6 Übungen und Probleme

Aufgabe 6.1 Der Algorithmus BACKTRACKING-SAT aus Abbildung 6.1 arbeite bei Eingabe der booleschen Formel

$$\varphi = (x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg u \vee y \vee z) \wedge (u \vee \neg y \vee \neg z) \wedge (u \vee \neg y \vee z).$$

Konstruiere Schritt für Schritt gemäß BACKTRACKING-SAT eine erfüllende Belegung für φ , falls eine solche existiert. Zeichne den entsprechenden Rekursionsbaum und markiere die Teilbäume, die abgeschnitten werden.

Aufgabe 6.2 Begründe, weshalb die Wahl von $t = \lceil (4/3)^n \rceil$ Wiederholungen im Algorithmus RANDOM-SAT aus Abbildung 6.2 eine hinreichend kleine Fehlerwahrscheinlichkeit erreicht.

Hinweis: Die in Abschnitt 6.1.2 und auf Seite 54 in Abschnitt 2.5 erwähnte Daumenregel sagte: Um eine hinreichend kleine Fehlerwahrscheinlichkeit zu erreichen, ist die Anzahl der Wiederholungen etwa reziprok zur Erfolgswahrscheinlichkeit eines Versuches. Genauer gesagt, wenn der Algorithmus t unabhängige Versuche macht, die jeweils eine Erfolgswahrscheinlichkeit p haben, so ist in dieser Aufgabe zu begründen, dass die Fehlerwahrscheinlichkeit dann durch

$$(1 - p)^t \leq e^{-t \cdot p}$$

abgeschätzt werden kann, wobei $e = 2.71828\cdots$ die Basis des natürlichen Logarithmus ist. Nimm weiter an, dass ε eine feste Fehlerwahrscheinlichkeit ist, die nicht überschritten werden sollte. Zeige, dass dies erreicht werden kann, indem t so groß gewählt wird, dass gilt: $t \geq \ln(1/\varepsilon)/p$.

Aufgabe 6.3 Beweise den zweiten Teil von Satz 6.4: PP = coPP.

Aufgabe 6.4 Beweise Korollar 6.7: RP ist unter Vereinigung und Durchschnitt abgeschlossen.

Aufgabe 6.5 Beweise, dass PP, BPP, RP und ZPP unter \leq_m^p -Reduktionen abgeschlossen sind.

Aufgabe 6.6 Beweise Fakt 6.18: Für jedes vernünftige Paar $(\mathcal{Q}_1, \mathcal{Q}_2)$ von Quantorenfolgen ist $(\mathcal{Q}_1 | \mathcal{Q}_2) = \text{co}(\mathcal{Q}_2 | \mathcal{Q}_1)$.

Aufgabe 6.7 Beweise die zweite Aussage von Lemma 6.20:

$$(\forall y) (\exists^+ z) [B(x, y, z)] \implies (\forall Y) (\exists^+ z) (\forall y \in Y) [B(x, y, z)].$$

Aufgabe 6.8 Beschreibe die Klassen AM, MAM und AMA der Arthur-Merlin-Hierarchie (siehe Definition 6.25) so, wie die Klasse MA in Beispiel 6.26 beschrieben wurde.

Aufgabe 6.9 (a) Charakterisiere die Klasse PP durch GapP-Funktionen so, wie $\oplus P$, $C=P$ und SPP in Definition 6.34 dargestellt wurden.

(b) Charakterisiere die Klassen $\oplus P$, $C=P$ und SPP aus Definition 6.34 so durch #P-Funktionen, wie PP in Anmerkung 6.3 dargestellt wurde.

Aufgabe 6.10 (a) Welche der folgenden Klassen sind Promise-Klassen: NP, coNP, PP, RP, coRP, ZPP, BPP, AM, coAM, MA, coMA, PP_{path}, BPP_{path} und RP_{path}? Die Definition der letzten drei Klassen findet man in Problem 6.1.

(b) Haben Promise-Klassen vollständige Probleme?

Aufgabe 6.11 (a) Beweise Behauptung 6.32.

(b) Beweise die in Lemma 6.33 genannten Abschlusseigenschaften von GapP.

Hinweis: Den Beweis findet man in [FFK94]. Hier sind ein paar spezifische Hinweise.

- **Subtraktion:** Verwende Behauptung 6.32.
- **Addition:** Gegeben $f = \text{gap}_M$ und x , rate nichtdeterministisch alle Wörter y einer Länge höchstens $p(|x|)$ und simuliere M bei Eingabe $\langle x, y \rangle$ für jedes geratene y .
- **Multiplikation:** Für gegebenes $f = \text{gap}_M$ und x rät die NPTM N nichtdeterministisch eine Folge von Berechnungspfaden von M bei den Eingaben $\langle x, 0 \rangle, \langle x, 1 \rangle, \dots, \langle x, p(|x|) \rangle$ und akzeptiert genau dann, wenn eine gerade Anzahl dieser Pfade akzeptiert. Der Beweis von

$$b(x) = \prod_{0 \leq y \leq p(|x|)} f(\langle x, y \rangle) = \text{gap}_N(x)$$

kann dann durch Induktion über $n = p(|x|)$ geführt werden.

- **Binomialkoeffizienten, eingeschränkte Komposition und Potenzierung:** Beweise zunächst eine schwächere Version des Abschlusses unter Binomialkoeffizienten: Ist $f \in \text{GapP}$ und ist $k \in \text{FP}$, wobei $k(x)$ polynomiell beschränkt in $|x|$ ist, so ist $h(x) = \binom{f(x)}{k(x)}$ eine Funktion in GapP . Für diesen Beweis ist das folgende kombinatorische Lemma nützlich.

Lemma 6.52. Für alle $r, s \in \mathbb{Z}$ und für jedes $k \in \mathbb{N}$ gilt:

$$\binom{s}{k} = \sum_{i=0}^k (-1)^i \binom{r+i}{i} \binom{r+s+1}{k-i}.$$

Die Binomialkoeffizienten sind durch

$$\binom{x}{y} = \frac{x(x-1)\cdots(x-y+1)}{y!}$$

für alle (einschließlich der negativen) reellen Zahlen x und für alle natürliche Zahlen y definiert. Per Konvention setzt man dabei $\binom{x}{0} = 1$.

Für den Beweis von Lemma 6.52 kann die Vandermondesche Konvolution [GKP89, p. 174] verwendet werden, welche sagt, dass für alle $a, b \in \mathbb{Z}$ und für jedes $k \in \mathbb{N}$ gilt:

$$\binom{a+b}{k} = \sum_{i=0}^k \binom{a}{i} \binom{b}{k-i}. \quad (6.21)$$

Die Intuition hinter (6.21) ist, dass wenn man ein Komitee mit k Mitgliedern aus einer Menge mit a Frauen und b Männern wählen möchte, so kann man ebenso gut für jedes $i \leq k$ zunächst i Frauen und dann, unabhängig, $k - i$ Männer wählen.

Definiere als Nächstes für alle ganzen Zahlen k und B mit $0 \leq k \leq B$ und für jede ganze Zahl x die Funktionen

$$\delta_k^B(x) = \binom{x}{k} \binom{B-x}{B-k}.$$

Man kann zeigen, dass diese Funktionen in FP (und somit in GapP) sind und dass gilt:

$$\delta_k^B(x) = \begin{cases} 1 & \text{falls } x = k \\ 0 & \text{falls } 0 \leq x \leq B \text{ und } x \neq k. \end{cases} \quad (6.22)$$

Unter Verwendung von (6.22) und des oben genannten schwächeren Abschlusses unter Binomialkoeffizienten kann nun gezeigt werden, dass die

Funktion $c(x) = f(\langle x, g(x) \rangle)$ in GapP ist, falls f und g GapP-Funktionen sind. Mit diesem Abschluss von GapP unter eingeschränkter Komposition wiederum kann der Beweis der stärkeren Version des Abschlusses unter Binomialkoeffizienten und des Abschlusses unter Potenzierung vollbracht werden: Sind f und g in GapP und ist $0 \leq g(x) \leq p(|x|)$ für ein Polynom p , dann sind die Funktionen d und e in GapP, die wie folgt definiert sind:

$$d(x) = \begin{pmatrix} f(x) \\ g(x) \end{pmatrix} \quad \text{und} \quad e(x) = f(x)^{g(x)}.$$

- (c) Welche der für GapP in Lemma 6.33 genannten Abschlusseigenschaften besitzt auch $\#P$? Beweise deine Vermutungen.

Hinweis: Wenn es Zweifel gibt, dass $\#P$ unter irgendeiner Eigenschaft π abgeschlossen ist, dann könnte man zum Beispiel zu beweisen versuchen, dass π „hart“ für die Abschlusseigenschaften von $\#P$ ist, nämlich in dem Sinne, dass $\#P$ genau dann unter π abgeschlossen ist, wenn es unter *jeder* polynomialzeitberechenbaren Operation abgeschlossen ist. Resulte dieser Art wurden von Ogi-hara und L. Hemaspaandra [OH93] gezeigt, die eine allgemeine Theorie der Abschlusseigenschaften von $\#P$ und verwandten Klassen entwickelten.

Aufgabe 6.12 Beweise Behauptung 6.35.

Aufgabe 6.13 (a) Beweise Korollar 6.38.

(b) Beweise, dass $UP = coUP$ aus $\#P^{UP} = \#P$ folgt. **Hinweis:** Siehe [KST89].

Aufgabe 6.14 (a) Beweise Satz 6.39.

Hinweis: Wende die im Beweis von Satz 6.37 vorgestellte Technik an. Ein etwas allgemeineres Resultat findet man als Theorem 4.2 in [KST92].

(b) Zeige, dass Lemma 6.40 aus Satz 6.39 und der Selbst-Lowness von SPP (d.h., $SPP^{SPP} = SPP$) folgt.

Aufgabe 6.15 Beweise Lemma 6.42.

Aufgabe 6.16 Modifiziere den Beweis von Satz 6.47 so, dass Korollar 6.48 gezeigt werden kann.

Aufgabe 6.17 Beweise, dass die Probleme GI und auto, die in Definition 2.49 bzw. Definition 6.49 definiert wurden, \leq_T^p -äquivalent sind: $auto \in FP^{GI}$ und $GI \in P^{auto}$.

Hinweis: Siehe Köbler, Schöning und Torán [KST93].

Problem 6.1 (Schwellwert-Klassen: PP_{path} , RP_{path} und BPP_{path})

In Anmerkung 6.3 wurde festgestellt, dass probabilistische Komplexitätstypen wie PP und RP über Wahrscheinlichkeitsgewichte oder, alternativ dazu, über die Anzahlen akzeptierender Pfade in den Berechnungsbäumen normalisierter NPTMs de-

finiert werden können. Ist die Normalisierungsforderung notwendig, damit der Beweis gelingt? Um diese Frage etwas genauer stellen zu können, definieren wir die Funktion $\text{tot}_M : \Sigma^* \rightarrow \mathbb{N}$ für eine gegebene NPTM M durch

$$\text{tot}_M(x) = \|\{\alpha \mid \alpha \text{ ist ein Pfad von } M(x)\}\|$$

und betrachten die folgenden Klassen:

$$\begin{aligned} \text{PP}_{\text{path}} &= \left\{ A \left| \begin{array}{l} \text{es gibt eine NPTM } M, \text{ so dass f\"ur jede Eingabe } x \text{ gilt:} \\ x \in A \iff \text{acc}_M(x) \geq (1/2)\text{tot}_M(x) \end{array} \right. \right\}; \\ \text{RP}_{\text{path}} &= \left\{ A \left| \begin{array}{l} \text{es gibt eine NPTM } M, \text{ so dass f\"ur jede Eingabe } x \text{ gilt:} \\ x \in A \implies \text{acc}_M(x) \geq (1/2)\text{tot}_M(x); \\ x \notin A \implies \text{acc}_M(x) = 0 \end{array} \right. \right\}; \\ \text{BPP}_{\text{path}} &= \left\{ A \left| \begin{array}{l} \text{es gibt eine NPTM } M \text{ und eine Konstante } \varepsilon > 0, \\ \text{so dass f\"ur jede Eingabe } x \text{ gilt:} \\ x \in A \implies \text{acc}_M(x) \geq (1/2 + \varepsilon)\text{tot}_M(x); \\ x \notin A \implies \text{acc}_M(x) \leq (1/2 - \varepsilon)\text{tot}_M(x) \end{array} \right. \right\}, \end{aligned}$$

wobei die Maschinen nicht notwendig normalisiert sein m\"ussen.

PP_{path} , RP_{path} und BPP_{path} werden manchmal auch „Schwellwert-Klassen“ (englisch „threshold classes“) genannt. Beweise die folgenden Behauptungen:

- (a) $\text{PP}_{\text{path}} = \text{PP}$.
- (b) $\text{RP}_{\text{path}} = \text{NP}$.
- (c) $\text{NP} \subseteq \text{BPP}_{\text{path}} \subseteq \text{PP}$.
- (d) $\text{P}^{\text{NP}[\log]} \subseteq \text{BPP}_{\text{path}}$. **Hinweis:** Beweise, dass BPP_{path} unter $\leq_{\text{tt}}^{\text{P}}$ -Reduktionen abgeschlossen ist und verwende die Gleichheit $\text{P}^{\text{NP}[\log]} = \text{P}_{\text{tt}}^{\text{NP}}$ aus Korollar 5.55.
- (e) BPP ist low f\"ur BPP_{path} , d.h., $\text{BPP}_{\text{path}}^{\text{BPP}} = \text{BPP}_{\text{path}}$. **Hinweis:** Siehe [HHT97].
- (f) $\text{BPP} \subseteq \text{NP}^{\text{BPP}} \subseteq \text{MA} \subseteq \text{BPP}_{\text{path}} \subseteq \text{BPP}^{\text{NP}}$. **Hinweis:** Siehe [HHT97].
- (g) F\"uge die Klassen PP_{path} , BPP_{path} und RP_{path} an der jeweils richtigen Stelle in Abbildung 6.8 ein.

Problem 6.2 (Abschlusseigenschaften von PP)

- (a) Beweise, dass die Klasse PP unter Durchschnitt abgeschlossen ist.

Hinweis: Siehe Beigel, Reingold und Spielman [BRS91]. Die Schl\"usselidee f\"ur diesen Beweis besteht darin, eine rationale Funktion $R_n : \mathbb{Z} \rightarrow \mathbb{Z}$ zu finden, so dass f\"ur alle x und y mit $-2^n \leq x, y \leq 2^n$ der Wert $R_n(x, y)$ genau dann positiv ist, wenn sowohl x als auch y positiv sind. Hier repräsentieren x und y die Lücken zwischen der Anzahl der akzeptierenden und der Anzahl der ablehnenden Pfade von NPTMs (also die Werte gegebener GapP-Funktionen) f\"ur die gegebenen PP-Sprachen X und Y , und $R_n(x, y)$ repräsentiert den Wert der zu konstruierenden GapP-Funktion f\"ur die Menge $Z = X \cap Y$. Wie man solch eine Funktion R_n konstruiert und wie man die erforderlichen Eigenschaften von R_n beweist, kann man in [BRS91] finden.

(b) Beweise, dass die Klasse PP sogar unter $\leq_{\text{tt}}^{\text{p}}$ -Reduktionen abgeschlossen ist.

Hinweis: Siehe Fortnow und Reingold [FR96].

Problem 6.3 (Lowness für probabilistische und Zählklassen)

Beweise die folgenden Lowness-Resultate:

(a) BPP ist selbst-low, d.h., $\text{BPP}^{\text{BPP}} = \text{BPP}$. **Hinweis:** Verwende die Technik zur Wahrscheinlichkeitsverstärkung gemäß Satz 6.13. Siehe auch [Ko82, Zac82].

(b) $\oplus\text{P}$ ist selbst-low, d.h., $\oplus\text{P}^{\oplus\text{P}} = \oplus\text{P}$. **Hinweis:** Siehe [PZ83].

(c) $\text{AM} \cap \text{coAM}$ ist low für AM, d.h., $\text{AM}^{\text{AM} \cap \text{coAM}} = \text{AM}$.

Hinweis: Dieser Beweis ist dem von Satz 6.29 ähnlich, wo gezeigt wird, dass $\text{AM} \cap \text{coAM}$ low für Σ_2^P ist, d.h., $\Sigma_2^{P, \text{AM} \cap \text{coAM}} = \Sigma_2^P$. Siehe auch den zweiten Teil von Satz 5.76, welcher sagt, dass $\text{NP} \cap \text{coNP}$ low für NP ist: $\text{NP}^{\text{NP} \cap \text{coNP}} = \text{NP}$. Siehe auch Theorem 2.44 in [KST93].

6.7 Zusammenfassung und bibliographische Notizen

Die derzeit besten oberen Schranken für die deterministische und randomisierte Zeitkomplexität von k -SAT, für $k \geq 3$, und auch einige ältere Ergebnisse können Tabelle 6.1 entnommen werden. Der in Abschnitt 6.1.2 präsentierte Irrfahrtsalgorithmus für 3-SAT geht auf Schöning [Sch99] zurück (siehe auch [Sch02b]). Die Fehler- und Laufzeitanalyse für RANDOM-SAT, die in Abschnitt 6.1.2 nur grob skizziert wurde, kann etwas genauer als hier in Schönings Buch [Sch01] gefunden werden.

Für k -SAT mit $k \geq 4$ ist der Algorithmus von Paturi et al. [PPSZ98] allerdings etwas besser als Schönings Algorithmus. Iwama und Tamaki [IT03] schlugen einen randomisierten Algorithmus für k -SAT mit $3 \leq k \leq 4$ vor, der mit einer Laufzeit von $\tilde{\mathcal{O}}(1.324^n)$ für 3-SAT und von $\tilde{\mathcal{O}}(1.474^n)$ für 4-SAT sowohl den Algorithmus von Paturi et al. [PPSZ98] als auch den von Schöning [Sch99] verbessert, indem er beide in intelligenter Weise kombiniert. Für k -SAT mit $k \geq 5$ ist Iwama und Tamakis Algorithmus allerdings nicht besser als der von Paturi et al. [PPSZ98].

Brueggemann und Kern [BK04] verbesserten den deterministischen 3-SAT-Algorithmus von Dantsin et al. [DGH⁺02] und halten mit $\tilde{\mathcal{O}}(1.473^n)$ zurzeit den Weltrekord in dieser Disziplin. Der derzeit mit $\tilde{\mathcal{O}}(1.32216^n)$ beste randomisierte Algorithmus für 3-SAT ist Rolf [Rol05] zu verdanken. Zu erwähnen sind schließlich einige Übersichtsartikel zur (exakten) Exponentialzeit-Algorithmitik, z.B. die von Woeginger [Woe03], Schöning [Sch05] und Riege und Rothe [RR06c].

Der Begriff der probabilistischen Turingmaschine und die Klassen PP, RP, ZPP und BPP wurden von Gill [Gil77] eingeführt und studiert. Simon [Sim75] führte den verwandten Begriff der „Schwellwert-Maschine“ (englisch „threshold machine“) ein, welcher auf dem Anteil anstatt dem Wahrscheinlichkeitsgewicht akzeptierender Pfade beruht; siehe Anmerkung 6.3. Problem 6.1(a) stammt aus [Sim75]. Han, L. Hemaspaandra und Thierauf [HHT97] verfolgten diese Forschungsrichtung weiter. Insbesondere untersuchten sie die in Problem 6.1 definierte Klasse BPP_{path}

sorgfältig hinsichtlich ihrer Beziehung zu anderen Klassen und bezüglich „sicherem Orakelzugriff“. Die Probleme 6.1(b) bis 6.1(f) stammen aus [HHT97].

Der Abschluss von PP unter Durchschnitt wurde von Beigel, Reingold und Spielman [BRS91] bewiesen. Fortnow und Reingold [FR96] verschärfen dieses Resultat, indem sie bewiesen, dass PP sogar unter der \leq_{tt}^p -Reduzierbarkeit abgeschlossen ist, wie in Abschnitt 6.2.1 und in Problem 6.2 erwähnt.

Arthur-Merlin-Spiele wurden von Babai und Moran [BM88, Bab85] eingeführt. Unabhängig von ihnen entwickelten Goldwasser, Micali und Rackoff [GMR89] die Theorie der interaktiven Beweissysteme, die ein im Wesentlichen äquivalentes Konzept hervorgebracht hat. Eine der besten, tiefgründigsten und umfangreichsten Quellen für diese Theorie ist Kapitel 4 in Goldreichs Buch [Gol01]. Andere hübsche Einführungen in dieses Gebiet können z.B. in den Büchern von Balcazar, Díaz und Gabarró [BDG90], Beutelspacher [Beu02], Buchmann [Buc01a, Buc01b], Köbler, Schöning und Torán [KST93], Papadimitriou [Pap95], Salomaa [Sal96], Stinson [Sti05] und Wechsung [Wec00] gefunden werden sowie in den Übersichtsartikeln [Gol88, Gol89, Rot02]. Die Lemmata 6.20 und 6.21 und Satz 6.22 gehen auf Zachos und Heller [Zac88, ZH86] zurück. Korollar 6.23, welches die Zugehörigkeit von BPP zur zweiten Stufe der Polynomialzeit-Hierarchie zeigt, ist Gács (wie von Sipser [Sip83] zitiert) und, unabhängig, Lautemann [Lau83] zu verdanken. Teil 2 von Korollar 6.28 wurde von Boppana, Hästad und Zachos [BHZ87] bewiesen. Satz 6.29 und Korollar 6.30 wurden von Schöning [Sch88] gezeigt, der außerdem Resultate erzielte, die den Kollaps aus Teil 2 von Korollar 6.28 und die in Satz 6.29 angegebene Lowness für probabilistische Klassen verallgemeinern, siehe [Sch89]. Vereshchagin [Ver92] bewies $MA \subseteq PP$.

Wie oben erwähnt sind Arthur-Merlin-Spiele und interaktive Beweissysteme im Wesentlichen derselbe Begriff. Ein Unterschied ist, dass man Merlin im letzteren Modell als *Beweiser* und Arthur als *Verifizierer* bezeichnet (englisch „*prover*“ und „*verifier*“). Beweiser und Verifizierer spielen keine Spiele, sondern sie kommunizieren über ein Protokoll, das einem kryptographischen Protokoll nicht unähnlich ist, auch wenn es nicht für den Zweck gedacht ist, geheime Nachrichten auszutauschen.

Signifikanter als der oben genannte, lediglich die Notation betreffende Unterschied zwischen Arthur-Merlin-Spielen und interaktiven Beweissystemen wirkt, zumindest auf den ersten Blick, ein anderer Unterschied, nämlich dass im ersten Modell Arthurs Zufallsbits öffentlich – und insbesondere Merlin bekannt – sind, wohingegen im letzteren Modell die Zufallsbits des Verifizierers privat sind. Jedoch konnten Goldwasser und Sipser [GS89] beweisen, dass dieser Unterschied zwischen den beiden Modellen eigentlich irrelevant ist: Es ist egal, ob man private oder öffentliche Münzen für die Zufallswürfe verwendet.

Wenn die beiden Spieler (Arthur und Merlin, oder Verifizierer und Beweiser) nicht eine konstante Zahl von Runden spielen, sondern polynomiell viele, so erhält man eine als IP bezeichnete Komplexitätsklasse. Nach Definition enthält IP alle NP-Mengen und insbesondere GI. Es enthält auch Probleme aus coNP, von denen man nicht glaubt, dass sie zu NP gehören, wie zum Beispiel das Graph-Nichtisomorphie-Problem, das nach Satz 6.45 zu AM und somit zu IP gehört. Ein gefeiertes Resultat

von A. Shamir [Sha92] sagt, dass IP gleich PSPACE ist, womit IP durch eine traditionelle Komplexitätsklasse charakterisiert werden konnte.

Die Klasse #P wurde von Valiant [Val79a, Val79b] eingeführt. Fenner, Fortnow und Kurtz [FFK94] verallgemeinerten #P zur Klasse GapP und entwarfen eine Theorie der gap-definierbaren Zählklassen. Ähnliche Ideen wurden unabhängig von Köbler, Schöning, Toda und Torán [KSTT92, KST92], von Gupta [Gup91] und von Ogiura und Hemaspaandra [OH93] entwickelt. Probabilistische Komplexitätsklassen wie PP sind gap-definierbar und ebenso viele andere Zählklassen wie z.B. $\oplus P$, $C=P$ und SPP. Die Klasse $\oplus P$ wurde von Papadimitriou und Zachos [PZ83] und, unabhängig, von Goldschlager und Parberry [GP86] eingeführt. Die „exakte Zählklasse“ $C=P$ wurde von Simon [Sim75] eingeführt, siehe auch [Wag86]. Analog zur Polynomialzeit-Hierarchie und beruhend auf Operatoren, die den Klassen PP und $C=P$ entsprechen, definierte Wagner [Wag86] die Zähl-Hierarchie (englisch „*counting hierarchy*“), welche von ihm und u.a. auch von Torán [Tor91] intensiv untersucht wurde. Papadimitriou und Zachos bewiesen, dass $\oplus P$ selbst-low ist; siehe Problem 6.3(b). Ko [Ko82] und Zachos zeigten, dass BPP selbst-low ist (siehe Problem 6.3(a)), und Köbler et al. [KSTT92] bewiesen die PP-Lowness von BPP.

Die Klasse SPP wurde in [FFK94] eingeführt. Die erste SPP-artige Maschine wurde allerdings in [KSTT92] beschrieben, siehe auch [KST92]. Davon unabhängig wurde SPP von Ogiura und L. Hemaspaandra [OH93] unter der Bezeichnung XP und von Gupta [Gup91] unter der Bezeichnung ZUP eingeführt. Satz 6.37 und Korollar 6.38 gehen auf Fenner, Fortnow und Kurtz [FFK94] zurück. Satz 6.39 und Lemma 6.40 sind Köbler, Schöning und Torán [KST92] zu verdanken. Eine etwas allgemeinere Version von Satz 6.39 kann als Theorem 4.2 in [KST92] gefunden werden. SPP verallgemeinert Valiants Klasse UP (siehe [Val76]), die in Definition 3.81 von Abschnitt 3.6 definiert wurde. Die Promise-Klassen UP und SPP sind seither intensiv studiert worden; siehe zum Beispiel [Val76, HH88, Sel92, KST92, KSTT92, FFK94, RRW94, Rot95, HR97b, HRW97b, BHR00, RH02, AK02b, HT03b, Hom04].

Das Verhältnis der Polynomialzeit-Hierarchie zu den Zählklassen PP, $C=P$, $\oplus P$ und SPP ist offen. Nach Beigel, Hemaspaandra und Wechsungs Resultat ist Θ_2^P die höchste Stufe der Polynomialzeit-Hierarchie, die beweisbar in PP enthalten ist [BHW91]. Im Gegensatz dazu bewies Beigel [Bei91b], dass, relativ zu einem Orakel, PP die Klasse Δ_2^P nicht enthält. Weitere relativierte Separationsresultate bezüglich der Zählklassen PP, $C=P$ und $\oplus P$ einerseits und der Polynomialzeit-Hierarchie andererseits wurden von Balcázar und Russo [Bal85, BR88], Bruschi [Bru92], Green [Gre91], Ko [Ko90], Rothe [Rot99] und Torán [Tor91] erzielt. Einige dieser Resultate verwenden untere Schranken für die Schaltkreiskomplexität bestimmter boolescher Funktionen, die beispielsweise von Razborov [Raz87], Smolensky [Smo87] und Furst, Saxe und Sipser [FSS84] bewiesen wurden.

Toda [Tod91] bewies, dass sich jede Menge aus der Polynomialzeit-Hierarchie auf eine Menge in PP \leq_T^P -reduzieren lässt, d.h., $PH \subseteq P^{PP}$. Mit seiner Technik lässt sich sogar zeigen, dass die gesamte Polynomialzeit-Hierarchie low für P^{PP} ist, d.h., $P^{PP^{PH}} = P^{PP}$. Indem sie ein bahnbrechendes Resultat von Valiant und Vazirani [VV86] verallgemeinerten, konnten Toda und Ogiura [TO92, Tod91] und, unabhängig, Tauri [Tar93] beweisen, dass Zählklassen mindestens so hart wie die Polynomialzeit-

Hierarchie sind. Genauer gesagt zeigten sie, dass für jedes $\mathcal{C} \in \{\text{PP}, \oplus\text{P}, \text{C}=\text{P}\}$ eine jede Menge in \mathcal{C}^{PH} auf eine Menge in \mathcal{C} bezüglich polynomialzeit-beschränkter randomisierter Many-one-Reduktionen reduziert werden kann. Im Zusammenhang mit den von Even, Selman und Yacobi [EY80, ESY84, Sel88b] eingeführten so genannten *Promise-Problemen* (nicht zu verwechseln mit dem Begriff der Promise-Klasse) und unter Verwendung geeigneter *Reduktionen zwischen Promise-Problemen*, zeigte Rothe [Rot95], dass jede Menge aus der Polynomialzeit-Hierarchie randomisiert auf ein Promise-Problem in einer Klasse reduziert werden kann, die SPP zu einer Klasse von Promise-Problemen verallgemeinert. Die komplexitätstheoretischen Untersuchungen in [EY80, ESY84, Sel88b, GS88] sind von Fragen motiviert, die in der Public-Key-Kryptographie auftreten. Weitere Resultate über Zählklassen können in den Arbeiten [Her90, BG92, Gup93, OH93, HO93, Tar93] gefunden werden.

Köbler, Schöning und Torán [KST93] haben eine umfangreiche, tiefgründige Abhandlung über das Graphisomorphie-Problem geschrieben, in der insbesondere seine komplexitätstheoretischen Eigenschaften untersucht werden. Hoffman [Hof82] beschäftigt sich in erster Linie mit gruppentheoretischen Algorithmen für GI.

Lemma 6.44 wurde von Goldreich, Micali und Wigderson [GMW91] und, unabhängig, von Goldwasser und Sipser [GS89] bewiesen. Satz 6.45, nach welchem GI in Low₂ ist, geht auf Schöning [Sch88] zurück. Arvind und Köbler [AK02a] verschärfen Schönings Resultat, indem sie bewiesen, dass das Graphisomorphie-Problem nicht nur für die Klasse $\Sigma_2^P = \text{NP}^{\text{NP}}$, sondern sogar für die Klasse ZPP^{NP} low ist, die in Σ_2^P enthalten ist. Lemma 6.42, welches im Beweis von Satz 6.45 angewandt wird und das man auch Sipsers Coding Lemma nennt, erschien in [Sip83], siehe auch [GS89]. Es ist eine Anwendung des von Carter und Wegman [CW79] entwickelten universellen Hashing.

Köbler et al. [KST92, KSTT92] erzielten die ersten Lowness-Resultate für GI bezüglich probabilistischer und Zählklassen wie PP und C=P. Sie zeigten auch, dass das Graphautomorphie-Problem, GA, in SPP und somit low für SPP, $\oplus\text{P}$, PP und C=P ist. Arvind und Kurur [AK02b] zeigten, dass GI ebenso in SPP enthalten ist, weshalb auch dieses Problem low für SPP und andere Zählklassen ist, siehe Satz 6.50.

Abbildung 6.8 gibt einen Überblick über die bekannten Inklusionsbeziehungen zwischen probabilistischen Klassen, Arthur-Merlin-Klassen, Zählklassen und den Stufen der Polynomialzeit-Hierarchie. Diese Inklusionsstruktur wird als Hasse-Diagramm dargestellt, das heißt, die Inklusion einer Klasse \mathcal{C} in einer Klasse \mathcal{D} wird durch eine Linie angezeigt, die von \mathcal{C} aus aufwärts zu \mathcal{D} verläuft. Von keiner der dargestellten Inklusionen weiß man, ob sie echt ist.

Schließlich sollen ein paar interessante Probleme erwähnt werden, die mit GI verwandt und durch die *Graphrekonstruktionsvermutung* inspiriert sind, welche auf Kelly und Ulam zurückgeht, siehe [Har69, Har74]. Obwohl sie bereits 1942 formuliert wurde, ist diese Vermutung noch immer offen und gilt als sehr schwer zu lösen. Zunächst werden einige Definitionen benötigt. Sei G ein Graph mit n Knoten, $V(G) = \{1, 2, \dots, n\}$. Eine Folge $\mathcal{G} = (G_1, G_2, \dots, G_n)$ von Graphen heißt *Deck von G*, falls es eine Permutation $\pi \in \mathfrak{S}_n$ gibt, so dass für jedes i mit $1 \leq i \leq n$ der Graph $G_{\pi(i)}$ isomorph zu dem Graphen ist, den man aus G durch Löschen des Knoten i und aller mit diesem verbundenen Kanten erhält. Ist \mathcal{G} ein Deck von G , dann nennt

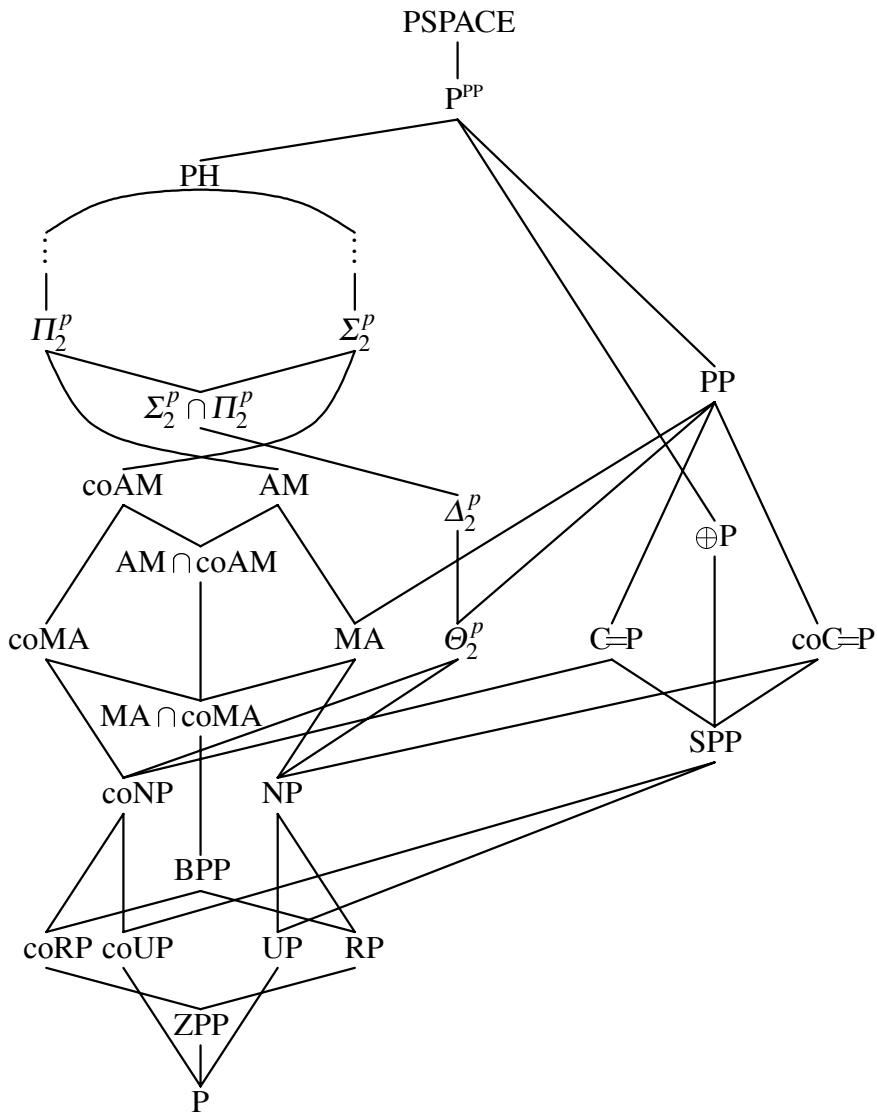


Abb. 6.8. Probabilistische, Arthur-Merlin- und Zählklassen und die PH

man G ein *Urbild von \mathcal{G}* . Eine Folge $\mathcal{G} = (G_1, G_2, \dots, G_n)$ heißt *legitimes Deck*, falls es ein Urbild hat. Die *Graphrekonstruktionsvermutung* sagt, dass für jedes legitime Deck ein (bis auf Isomorphie) eindeutig bestimmtes Urbild existiert. Definiere die folgenden Probleme:

$$\text{Deck-Checking} = \{\langle G, \mathcal{G} \rangle \mid G \text{ ist ein Urbild von } \mathcal{G} = \langle G_1, G_2, \dots, G_n \rangle\};$$

$$\text{Legitimate-Deck} = \{\mathcal{G} \mid \mathcal{G} = \langle G_1, G_2, \dots, G_n \rangle \text{ ist ein legitimes Deck}\}.$$

Kratsch und L. Hemaspaandra [KH94] bewiesen:

$$\text{Deck-Checking} \leq_m^p \text{GI} \leq_m^p \text{Legitimate-Deck}.$$

Die Frage, ob Legitimate-Deck auf GI \leq_m^p -reduzierbar ist, ist offen. Köbler, Schöning und Torán [KST92] zeigten, dass Legitimate-Deck low für bestimmte probabilistische und Zählklassen inklusive PP und C=P ist, falls die Graphrekonstruktionsvermutung gilt. Weitere Resultate zur Graphenrekonstruktion bewiesen E. Hemaspaandra, L. Hemaspaandra, Radziszowski und Tripathi [HHRT04].

RSA-Kryptosystem, Primzahltests und das Faktorisierungsproblem

The problem of distinguishing prime numbers from composites, and of resolving composite numbers into their prime factors, is one of the most important and useful in all of arithmetic. . . . The dignity of science seems to demand that every aid to the solution of such an elegant and celebrated problem be zealously cultivated.

(Aus „*Disquisitiones Arithmeticae*“ von Carl Friedrich Gauß,
die Übersetzung aus dem Lateinischen ist aus [Knu98])

In den letzten beiden Kapiteln, die sich wieder der Kryptographie zuwenden, werden einige grundlegende kryptographische Protokolle behandelt. Die Sicherheit solcher Protokolle beruht gewöhnlich auf der Annahme, dass bestimmte Probleme aus der Zahlentheorie und Algebra „widerspenstig“, also schwer zu lösen sind. Um also diese Kryptosysteme und Protokolle beschreiben und ihre Sicherheit diskutieren zu können, benötigen wir einige zahlentheoretische, algebraische und komplexitäts-theoretische Begriffe, Methoden und Resultate.

Abschnitt 7.1 stellt das berühmte RSA-Kryptosystem vor, das 1978 von Rivest, A. Shamir und Adleman erfunden wurde. Ihr Resultat ist ein Meilenstein in der Geschichte der Kryptographie, denn RSA war das allererste Public-Key-Kryptosystem in der offenen Literatur (siehe die bibliographischen Anmerkungen in Abschnitt 7.6). RSA ist noch immer sehr populär und spielt in verschiedenen kryptographischen Anwendungen eine Rolle. Beispielsweise kann das RSA-System so modifiziert werden, dass man ein digitales Signatur-Schema erhält.

Das RSA-System benutzt große Primzahlen. Das seit Jahrhunderten untersuchte Primzahl-Problem fragt, ob eine gegebene natürliche Zahl prim ist oder nicht. Es ist schon lange bekannt, dass dieses Problem in randomisierter Polynomialzeit gelöst werden kann. Abschnitt 7.2 stellt einige der gebräuchlichsten randomisierten Primärtätstests vor, wie zum Beispiel den Miller–Rabin-Test und den Solovay–Strassen-Test. Die zahlentheoretischen Ergebnisse, auf denen diese Tests beruhen, werden in der zum Verständnis erforderlichen Tiefe präsentiert. Ebenfalls erwähnt wird das gefeierte Resultat von Agrawal, Kayal und N. Saxena, welches sagt, dass das Primzahl-Problem sogar in deterministischer Polynomialzeit gelöst werden kann.

Das RSA-System ist nur dann sicher, wenn große Zahlen nicht effizient faktoriert werden können. Abschnitt 7.3 studiert das Faktorisierungsproblem, das bisher allen Versuchen, effiziente Algorithmen für seine Lösung zu entwerfen, widerstan-

den hat. Es gibt Tonnen an Literatur über Faktorisierungsalgorithmen und ihre Anwendungen in der Praxis, und einige dieser Algorithmen werden in Abschnitt 7.3 diskutiert. Abschnitt 7.4 untersucht die Sicherheit von RSA, stellt bestimmte mögliche Angriffe auf dieses Kryptosystem vor und erörtert geeignete Gegenmaßnahmen.

7.1 RSA

Wir haben in Kapitel 4 gesehen, dass viele symmetrische Kryptosysteme durch bestimmte Angriffsarten gebrochen werden können. Insbesondere sind affin-lineare Blockchiffren unsicher gegen Known-Plaintext-Angriffe. Neben solchen Sicherheitsproblemen (und trotz ihrer Vorteile hinsichtlich der Effizienz) liegt ein weiterer Nachteil der symmetrischen Kryptosysteme im Problem der Schlüsselverwaltung und Schlüsselverteilung. Bei der Nutzung eines großen Computernetzwerks durch viele Nutzer etwa oder bei der Kommunikation über das Internet kann die Frage, wie die Schlüssel für ein symmetrisches Kryptosystem verteilt und dann gemeinsam genutzt werden, recht mühselig sein.

Eine elegante Lösung dieses Problems, das Diffie–Hellman–Protokoll, wird in Abschnitt 8.1 präsentiert. Interessanterweise öffnete ihre Lösung des Schlüsseltausch-Problems in der symmetrischen Kryptographie die Tür zu einer grundlegend anderen, neuen Richtung der Kryptographie, bei der es keinen Bedarf mehr gibt, private Schlüssel zu (ver)teilen. In der Public-Key–Kryptographie wählt Bob sowohl einen privaten Schlüssel, den er geheimhält, als auch einen öffentlichen Schlüssel, der allen anderen Nutzern in einem öffentlichen Verzeichnis zugänglich gemacht wird. Möchte Alice ihm eine Nachricht zukommen lassen, so schlägt sie seinen öffentlichen Schlüssel nach und verwendet ihn, um ihre Nachricht zu verschlüsseln. Die Idee ist, dass Bob ihren Schlüsseltext leicht entschlüsseln kann, indem er seinen privaten Schlüssel benutzt, wohingegen es einer nicht autorisierten Partei, wie etwa dem Lau-scher Erich, misslingt, Alice' Schlüsseltext zu entschlüsseln, denn ihm fehlt Bobs privater Schlüssel.

7.1.1 Das RSA Public-Key–Kryptosystem

Viele der Protokolle in den Kapiteln 7 und 8 verwenden Grundbegriffe aus der Zah-lentheorie, siehe Abschnitt 2.4. Um insbesondere das RSA–Kryptosystem zu be-schreiben, benötigen wir die multiplikative Gruppe

$$\mathbb{Z}_k^* = \{i \mid 1 \leq i \leq k-1 \text{ und } \text{ggT}(i, k) = 1\},$$

die in Beispiel 2.35 eingeführt wurde. Ebenfalls wichtig sind die Euler-Funktion φ aus Definition 2.36, welche die Ordnung von \mathbb{Z}_k^* angibt, sowie die Arithmetik im Ring $\mathbb{Z}_k = \{0, 1, \dots, k-1\}$, siehe Problem 2.1. Der Begriff des Public-Key–Kryptosystems wurde formal in Definition 4.1 eingeführt.

Abbildung 7.1 fasst die einzelnen Schritte des RSA–Protokolls zusammen. Nun werden diese Schritte etwas detaillierter beschrieben.

Schritt	Alice	Erich	Bob
1			wählt zufällig zwei große Primzahlen, p und q , und berechnet $n = pq$ und $\varphi(n) = (p-1)(q-1)$, seinen öffentlichen Schlüssel (n, e) und seinen privaten Schlüssel d gemäß (7.1) und (7.2)
2		$\Leftarrow (n, e)$	
3	verschlüsselt m als $c = m^e \bmod n$		
4		$c \Rightarrow$	
5			entschlüsselt c durch $m = c^d \bmod n$

Abb. 7.1. RSA-Protokoll

Schritt 1: Schlüsselerzeugung. Bob wählt zwei verschiedene große Primzahlen, p und q mit $p \neq q$, und berechnet ihr Produkt $n = pq$. Dann wählt er einen Exponenten $e \in \mathbb{N}$, für den

$$1 < e < \varphi(n) = (p-1)(q-1) \quad \text{und} \quad \text{ggT}(e, \varphi(n)) = 1 \quad (7.1)$$

gilt. Mit dem erweiterten Euklidischen Algorithmus aus Abbildung 2.2 bestimmt er nun das inverse Element von $e \bmod \varphi(n)$, d.h., die eindeutige Zahl d mit

$$1 < d < \varphi(n) \quad \text{und} \quad e \cdot d \equiv 1 \bmod \varphi(n). \quad (7.2)$$

Das Paar (n, e) ist Bobs öffentlicher Schlüssel und d ist Bobs privater Schlüssel.

Schritt 2: Kommunikation. Bob macht seinen Schlüssel (n, e) öffentlich zugänglich.

Schritt 3: Verschlüsselung. Wie in Kapitel 4 sind Nachrichten Wörter über einem Alphabet Σ , welche als natürliche Zahlen in $\|\Sigma\|$ -adischer Darstellung aufgefasst werden können. Jede Nachricht soll blockweise mit einer festen Blocklänge verschlüsselt werden. Sei $m < n$ die Zahl, die einen Block der Nachricht codiert, die Alice an Bob schicken möchte. Alice kennt Bobs öffentlichen Schlüssel (n, e) und verschlüsselt m als die Zahl $c = E_{(n,e)}(m)$, wobei die Verschlüsselungsfunktion definiert ist durch

$$E_{(n,e)}(m) = m^e \bmod n. \quad (7.3)$$

Schritt 4: Kommunikation. Alice schickt ihren Schlüsseltext c an Bob.

Schritt 5: Entschlüsselung. Sei c mit $0 \leq c < n$ die Zahl, die einen Block des Schlüsseltextes codiert, den Bob erhält. Der Lauscher Erich kennt c möglicherweise ebenfalls, doch er kennt nicht Bobs privaten Schlüssel d . Bob entschlüsselt c mittels d und der Entschlüsselungsfunktion

$$D_d(c) = c^d \bmod n. \quad (7.4)$$

Satz 7.1 sagt, dass das oben beschriebene RSA-Schema tatsächlich ein Kryptosystem gemäß Definition 4.1 ist. Das heißt, Ver- und Entschlüsselung sind zueinander invers.

Satz 7.1. Seien (n, e) der öffentliche und d der private Schlüssel im RSA-Protokoll. Dann gilt für jede Nachricht m mit $0 \leq m < n$:

$$m = (m^e)^d \bmod n.$$

Beweis. Nach (7.2) gilt $e \cdot d \equiv 1 \pmod{\varphi(n)}$. Somit existiert eine ganze Zahl k , so dass

$$e \cdot d = 1 + k(p-1)(q-1)$$

gilt, wobei $n = pq$. Daraus folgt:

$$\begin{aligned} (m^e)^d &= m^{e \cdot d} = m^{1+k(p-1)(q-1)} \\ &= m \left(m^{k(p-1)(q-1)} \right) \\ &= m \left(m^{p-1} \right)^{k(q-1)}. \end{aligned}$$

Weiter ergibt sich

$$(m^e)^d \equiv m \pmod{p}, \quad (7.5)$$

denn wenn p ein Teiler von m ist, dann sind beide Seiten von (7.5) kongruent zu $0 \pmod{p}$, und falls p kein Teiler von m ist (d.h., falls $\text{ggT}(p, m) = 1$ ist), so gilt

$$m^{p-1} \equiv 1 \pmod{p} \quad (7.6)$$

nach dem Kleinen Fermat. Ein symmetrisches Argument zeigt:

$$(m^e)^d \equiv m \pmod{q}. \quad (7.7)$$

Weil p und q verschiedene Primzahlen sind, implizieren (7.6) und (7.7) mit Satz 2.46:

$$(m^e)^d \equiv m \pmod{n}.$$

Da $m < n$ gilt, ist der Beweis abgeschlossen. □

Das RSA-Protokoll kann effizient implementiert werden. Alice berechnet $c = m^e \pmod{n}$ und Bob berechnet $m = c^d \pmod{n}$. Naiv ausgeführt würden diese Berechnungen, in Abhängigkeit von der Größe des Exponenten, zu viele Multiplikationen erfordern. Glücklicherweise jedoch kann die modulare Exponentiationsfunktion effizient berechnet werden, indem man den „Square-and-Multiply“-Algorithmus aus Abbildung 7.2 verwendet.

Die Berechnung von $b^a \pmod{m}$ in Abbildung 7.2 ist korrekt, denn in der Arithmetik modulo m gilt:

$$b^a = b^{\sum_{i=0}^k a_i 2^i} = \prod_{\substack{i=0 \\ a_i=1}}^k \left(b^{2^i} \right)^{a_i} = \prod_{\substack{i=0 \\ a_i=1}}^k b^{2^i}.$$

```

SQUARE-AND-MULTIPLY( $a, b, m$ ) {
    //  $a$  ist der Exponent,  $b < m$  die Basis und  $m$  der Modul
    Bestimme die Binärentwicklung des Exponenten  $a = \sum_{i=0}^k a_i 2^i$ , wobei  $a_i \in \{0, 1\}$ ;
    Berechne sukzessive  $b^{2^0}, b^{2^1}, \dots, b^{2^k}$  durch Anwendung der Kongruenz
         $b^{2^{i+1}} \equiv (b^{2^i})^2 \pmod{m};$ 
        // die Zwischenwerte  $b^{2^i}$  müssen dabei nicht gespeichert werden
    Berechne  $b^a = \prod_{\substack{i=0 \\ a_i=1}}^k b^{2^i}$  in der Arithmetik modulo  $m$ ;
    return  $b^a$ ;
}

```

Abb. 7.2. Der *Square-and-Multiply*-Algorithmus

Wie sollte man die Primzahlen p und q im RSA-Protokoll aus Abbildung 7.1 wählen? Zunächst müssen sie groß genug sein, da Erich andernfalls die Zahl n in Bobs öffentlichem Schlüssel (n, e) faktorisieren könnte. Und sind ihm die Primfaktoren von n bekannt, so auch $\varphi(n) = (p - 1)(q - 1)$, und er kann mit dem erweiterten Euklidischen Algorithmus aus Abbildung 2.2 leicht Bobs privaten Schlüssel d bestimmen, der das eindeutige Inverse von $e \pmod{\varphi(n)}$ ist. Deshalb müssen die Primzahlen p und q geheimgehalten und hinreichend groß gewählt werden, denn nur so kann dieser direkte Angriff verhindert werden. In der Praxis sollten sie so groß gewählt werden, dass sie mindestens 80 Dezimalziffern haben. Üblicherweise erzeugt man Zahlen dieser Größe zufällig und überprüft dann mit einem der bekannten Primalitätstests, ob die gewählten Zahlen prim sind oder nicht. Nach dem Primzahlssatz gibt es ungefähr $N / \ln N$ Primzahlen unterhalb von N , siehe Satz 7.3. Also stehen die Chancen gut, dass man nach nicht allzu vielen Versuchen auf eine Primzahl stößt. Solche Primalitätstests werden in Abschnitt 7.2 beschrieben. Beispiel 7.2 zeigt eine konkrete Anwendung des RSA-Systems. Natürlich ist dies nur ein Spielzeugbeispiel mit viel zu kleinen Zahlen und daher weit davon entfernt, sicher zu sein.

Beispiel 7.2 (RSA). Bob wählt die Primzahlen $p = 67$ und $q = 11$ und berechnet $n = 67 \cdot 11 = 737$ und $\varphi(n) = (p - 1)(q - 1) = 66 \cdot 10 = 660$. Wählt Bob nun den kleinstmöglichen Exponenten für $\varphi(n) = 660$, nämlich $e = 7$, dann ist das Paar $(n, e) = (737, 7)$ sein öffentlicher Schlüssel. Mit dem erweiterten Euklidischen Algorithmus aus Abbildung 2.2 bestimmt Bob seinen privaten Schlüssel $d = 283$, und es ergibt sich (siehe Übung 7.1):

$$e \cdot d = 7 \cdot 283 = 1981 \equiv 1 \pmod{660}.$$

Wie in Abschnitt 4.1 wird das Alphabet $\Sigma = \{A, B, \dots, Z\}$ mit der Menge $\mathbb{Z}_{26} = \{0, 1, \dots, 25\}$ identifiziert. Nachrichten sind wieder Wörter über Σ und werden, mit einer festen Blocklänge, blockweise als natürliche Zahlen in 26-adischer Darstellung codiert. Die Blocklänge in unserem Beispiel ist

$$\ell = \lfloor \log_{26} n \rfloor = \lfloor \log_{26} 737 \rfloor = 2.$$

Jeder Block $b = b_1 b_2 \cdots b_\ell$ der Länge ℓ mit $b_i \in \mathbb{Z}_{26}$ wird durch die Zahl $m_b = \sum_{i=1}^{\ell} b_i \cdot 26^{\ell-i}$ repräsentiert. Aus der Definition der Blocklänge $\ell = \lfloor \log_{26} n \rfloor$ folgt

$$0 \leq m_b \leq 25 \cdot \sum_{i=1}^{\ell} 26^{\ell-i} = 26^\ell - 1 < n.$$

Mit der RSA-Verschlüsselungsfunktion (7.3), wird die dem Block b entsprechende Zahl m_b durch $c_b = (m_b)^e \bmod n$ verschlüsselt, wobei $c_b = c_0 c_1 \cdots c_\ell$ mit $c_i \in \mathbb{Z}_{26}$ der Schlüsseltext für Block b ist. RSA bildet also Blöcke der Länge ℓ injektiv auf Blöcke der Länge $\ell + 1$ ab. Tabelle 7.1 zeigt eine Nachricht der Länge 34, die in 17 Blöcke der Länge 2 unterteilt ist, und wie die einzelnen durch Zahlen codierten Blöcke dann verschlüsselt werden. Alice codiert diese Nachricht Block für Block durch Zahlen, verschlüsselt diese gemäß der RSA-Funktion, decodiert die den einzelnen Blöcken entsprechenden Zahlen wieder als Wörter über Σ und schickt Bob den resultierenden Schlüsseltext. Beispielsweise wird der erste Block, „RS“, folgendermaßen in eine Zahl umgewandelt: Das „R“ entspricht der 17 und das „S“ der 18, also ergibt sich:

$$17 \cdot 26^1 + 18 \cdot 26^0 = 442 + 18 = 460.$$

Tabelle 7.1. Beispiel einer RSA-Verschlüsselung

	R	S	A	I	S	T	H	E	K	Y	T	O	P	U	B	L	I	C	K	E	Y	C	R	Y	P	T	O	G	R	A	P	H	Y
m_b	460	8	487	186	264	643	379	521	294	62	128	69	639	508	173	15	206																
c_b	697	387	229	340	165	223	586	5	189	600	325	262	100	689	354	665	673																

Die resultierende Zahl c_b wird wieder in 26-adischer Darstellung geschrieben und kann die Länge $\ell + 1$ haben:

$$c_b = \sum_{i=0}^{\ell} c_i \cdot 26^{\ell-i},$$

wobei $c_i \in \mathbb{Z}_{26}$, siehe Übung 7.1. Insbesondere wird der erste Block, $697 = 676 + 21 = 1 \cdot 26^2 + 0 \cdot 26^1 + 21 \cdot 26^0$, in den Schlüsseltext „BAV“ umgewandelt.

Entschlüsselt wird ebenfalls blockweise. Um zum Beispiel den ersten Block mit dem privaten Schlüssel $d = 283$ zu entschlüsseln, berechnet Bob $697^{283} \bmod 737$, wobei wieder das schnelle Potenzieren aus Abbildung 7.2 benutzt wird. Es ist nützlich, nach jeder Multiplikation modulo $n = 737$ zu reduzieren, um zu verhindern, dass die Zahlen zu groß werden. Die Binärentwicklung des Exponenten ist $283 = 2^0 + 2^1 + 2^3 + 2^4 + 2^8$, und es ergibt sich

$$697^{283} \equiv 697^{2^0} \cdot 697^{2^1} \cdot 697^{2^3} \cdot 697^{2^4} \cdot 697^{2^8} \equiv 697 \cdot 126 \cdot 9 \cdot 81 \cdot 15 \equiv 460 \bmod 737,$$

wie gewünscht.

7.1.2 Digitale Signaturen mit RSA

Das RSA Public-Key-Kryptosystem aus Abbildung 7.1 kann so modifiziert werden, dass man das in Abbildung 7.3 dargestellte Protokoll für digitale Signaturen erhält. Es ist leicht zu sehen, dass das Protokoll funktioniert, siehe Übung 7.2. Dieses Protokoll ist zum Beispiel gegen Chosen-Plaintext-Angriffe anfällig, d.h., der Angreifer kann Klartexte wählen und erfährt die entsprechenden Schlüsseltexte, und aus dieser Information kann er Schlüssel bestimmen. Abschnitt 7.4 beschreibt diesen Angriff und mögliche Gegenmaßnahmen, um ihn zu verhindern.

Schritt	Alice	Erich	Bob
1	wählt $n = pq$, ihren öffentlichen Schlüssel (n, e) und ihren privaten Schlüssel d so, wie Bob dies im RSA-Protokoll von Abbildung 7.1 tut		
2		$(n, e) \Rightarrow$	
3	signiert die Nachricht m mit $\text{sig}_A(m) = m^d \bmod n$		
4		$\langle m, \text{sig}_A(m) \rangle \Rightarrow$	
5			verifiziert Alice' Signatur durch $m \equiv (\text{sig}_A(m))^e \bmod n$

Abb. 7.3. Digitale Signaturen mit RSA

7.2 Primzahltests

Wie in Abschnitt 7.1.1 erwähnt verwenden viele Kryptosysteme große Primzahlen. In der Praxis erhält man diese dadurch, dass man zufällig Zahlen der erforderlichen Größe wählt und dann überprüft, ob diese tatsächlich Primzahlen sind oder nicht. Diese Überprüfung wird üblicherweise mit einem der randomisierten Primzahltests durchgeführt, die in diesem Abschnitt beschrieben werden. Im Zusammenhang damit soll hier auch die Komplexität des Primzahlproblems untersucht werden.

Sowohl zur Erzeugung großer Zahlen als auch für diese Primzahltests benötigt man eine geeignete Zufallsquelle. Im Idealfall würde man einfach eine faire Münze wieder und wieder werfen, um einen Strom zufälliger Bits zu erzeugen, oder mit einem fairen n -seitigen Würfel würfeln, um unter Gleichverteilung eine Zufallszahl aus dem Intervall $\{1, 2, \dots, n\}$ zu ziehen. Doch die Welt ist nicht ideal. Die Erzeugung echter Zufallsfolgen ist in der Regel einfach zu teuer. In der Praxis verwendet

man deshalb Pseudozufallsgeneratoren, die Folgen von Zahlen (oder Bits) produzieren, die „zufällig aussehen“ in dem Sinn, dass sie statistisch oder effizient nicht von wirklichen Zufallsfolgen zu unterscheiden sind.

Eine natürliche Zahl $n \geq 2$ heißt *Primzahl*, falls sie keine Teiler außer den trivialen hat, 1 und n . Ist eine Zahl n nicht prim, so heißt sie *zusammengesetzt*, d.h., dann ist $n = x \cdot y$ das Produkt natürlicher Zahlen x und y mit $1 < x, y < n$.

Die zu Beginn dieses Abschnitts beschriebene Methode zum Finden von Primzahlen hat nur deshalb Aussicht auf Erfolg, weil die Primzahlen in Intervallen der Form $\{1, 2, \dots, N\}$ hinreichend dicht liegen. Dieser Fakt ist als das Primzahl-Theorem bekannt, ein tiefliegendes Resultat aus der Zahlentheorie, auf dessen Beweis wir hier verzichten.

Satz 7.3 (Primzahl-Theorem). *Bezeichnet $\pi(N)$ die Anzahl der Primzahlen p mit $p \leq N$ und bezeichnet $\ln N$ den natürlichen Logarithmus, so gilt:*

$$\lim_{N \rightarrow \infty} \frac{\pi(N) \ln N}{N} = 1.$$

Oft genügt es, die folgende schwächere Version des Primzahl-Theorems zu betrachten, die (bei Vernachlässigung von Konstanten) Abschätzungen der oberen und unteren Schranken von $\pi(N)$ angibt:

$$\mathcal{O}\left(\frac{N}{\log N}\right) \leq \pi(N) \leq \mathcal{O}\left(\frac{N}{\log N}\right).$$

Das Primzahl-Problem fragt für eine binär gegebene Zahl $n \geq 2$, ob n prim ist oder nicht. Dieses ganz natürliche, faszinierende Problem wurde bereits im Altertum studiert, lange bevor seine praktische Bedeutung (etwa für kryptographische Anwendungen) zum Vorschein kam, und seither hat es Mathematiker und Informatiker stetig begeistert.

Definition 7.4 (Primzahl-Problem). *Definiere das Primzahl-Problem durch*

$$\text{Primes} = \{\text{bin}(n) \mid n \text{ ist eine Primzahl}\}.$$

Natürlich wäre dieses Problem trivial, wenn **Primes** eine endliche Menge wäre. Euklid bewies jedoch, dass es unendlich viele Primzahlen gibt, siehe Übung 7.3. Eine wohlbekannte Methode, eine endliche Liste von Primzahlen zu erzeugen, ist das *Sieb des Eratosthenes*.

Beispiel 7.5 (Sieb des Eratosthenes). Informal gesagt arbeitet dieser antike Algorithmus folgendermaßen. Angenommen, man möchte wissen, welche Zahlen in $\{2, 3, \dots, n\}$ Primzahlen sind. Zu Beginn ist keines der Elemente in diesem Intervall markiert. Für $i = 2, 3, \dots, \lfloor \sqrt{n} \rfloor$ werden in dieser Reihenfolge nun die folgenden Schritte ausgeführt: Ist i unmarkiert, so markiere alle Vielfachen von i , d.h., alle Zahlen $j = k \cdot i$ mit $i \leq k \leq n/i$. Keine Primzahl kann durch diese Prozedur je markiert werden, denn nur Zahlen mit nichttrivialen Teilern werden markiert. Alle zusammengesetzten Zahlen in $\{2, 3, \dots, n\}$ werden jedoch markiert. Tabelle 7.2 zeigt als ein konkretes Beispiel, welche Zahlen aus dem Intervall $\{2, 3, \dots, 39\}$ bei der Anwendung des Siebs von Eratosthenes markiert werden.

Tabelle 7.2. Sieb des Eratosthenes

$i = 2$	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
$i = 3$	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
$i = 4$	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
$i = 5$	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
$i = 6$	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39

Abbildung 7.4 zeigt einen naiven Zugang zur Lösung des Primzahl-Problems, die Probedivision. Beginnend mit $i = 2$ überprüft dieser Algorithmus in jedem Durchlauf, ob der aktuelle Wert von i die Eingabe n teilt. Falls ja, ist n zusammengesetzt. Andernfalls wird i um eins erhöht, um den nächsten Kandidaten zu überprüfen, und so weiter, bis i den Wert $\lfloor \sqrt{n} \rfloor$ erreicht hat. War keiner der Tests erfolgreich, so antwortet der Algorithmus, dass n prim ist. Ist $\ell = \lfloor \log n \rfloor + 1$ die Länge der binär codierten Eingabe n , so läuft dieser Algorithmus im schlechtesten Fall in der Zeit $\mathcal{O}(2^{\ell/2})$ und ist demnach weit davon entfernt, effizient zu sein.

```
TRIAL-DIVISION( $n$ ) {
    //  $n \in \mathbb{N}$  mit  $n \geq 2$ 
    for ( $i = 2, 3, \dots, \lfloor \sqrt{n} \rfloor$ ) {
        if ( $i$  teilt  $n$ ) return „ $n$  ist zusammengesetzt“ und halte;
    }
    return „ $n$  ist prim“ und halte;
}
```

Abb. 7.4. Probedivision zur Lösung des Primzahl-Problems

Man könnte versuchen, den Algorithmus TRIAL-DIVISION zu verbessern. Überprüft man beispielsweise in einer Vorbereitungsphase, ob n teilbar ist durch 2, 3, 5 oder 7, und überspringt dann alle Vielfachen von 2, 3, 5 und 7 in der `for`-Schleife, so beschleunigt man diese Berechnung. Solche simplen Tricks reichen jedoch nicht, um eine nennenswerte Laufzeitverbesserung zu erzielen. Im Gegensatz dazu sind randomisierte Algorithmen, die bestimmte zahlentheoretische Resultate ausnutzen, viel geeigneter, um die Primheit von Zahlen effizient zu testen. In den folgenden Abschnitten werden randomisierte, effiziente Algorithmen für das Primzahl-Problem diskutiert.

7.2.1 Fermat-Test

Der Kleine Fermat (siehe Korollar 2.39) sagt, dass $a^{p-1} \equiv 1 \pmod{p}$ für jede Primzahl p mit $1 \leq a < p$ gilt. Dieses Resultat stellt ein Kriterium bereit, mit dem man die Primheit von Zahlen überprüfen kann und das in einem randomisierten Primätstest verwendet wird, der nun vorgestellt wird.

Beispiel 7.6. Sei $a = 3$. Tabelle 7.3 enthält die Paare $(n, 3^{n-1} \pmod{n})$ für jedes n mit $4 \leq n \leq 23$. Alle zusammengesetzten Zahlen n in dieser Tabelle haben einen von 1 verschiedenen Wert $3^{n-1} \pmod{n}$, wohingegen $3^{n-1} \pmod{n} = 1$ für alle Primzahlen n gilt. Anscheinend stellt also der Kleine Fermat ein gutes Kriterium für die Primheit von Zahlen dar.

Tabelle 7.3. Primzahltest unter Verwendung des Kleinen Fermat

n	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
$3^{n-1} \pmod{n}$	3	1	3	1	3	0	3	1	3	1	3	9	11	1	9	1	7	9	3	1

Nun werden die Begriffe Fermat-Zeuge und Fermat-Lügner eingeführt. Intuitiv zertifiziert ein Fermat-Zeuge die Zusammengesettheit einer Zahl: Nach der Kontraposition des Kleinen Fermat muss n zusammengesetzt sein, wenn $a^{n-1} \not\equiv 1 \pmod{n}$ für n und a gilt, $1 \leq a < n$. Ein solches a heißt Fermat-Zeuge für n . Eine Zahl a , die $a^{n-1} \equiv 1 \pmod{n}$ erfüllt und n somit wie eine Primzahl erscheinen lässt, obwohl n eigentlich zusammengesetzt ist, heißt hingegen Fermat-Lügner für n .

Definition 7.7 (Fermat-Zeuge und Fermat-Lügner).

- Sei $n \geq 2$. Eine Zahl a , $1 \leq a < n$, heißt Fermat-Zeuge für n , falls $a^{n-1} \not\equiv 1 \pmod{n}$.
- Für jede zusammengesetzte Zahl $n \geq 3$ heißt eine Zahl a , $1 \leq a < n$, Fermat-Lügner für n , falls $a^{n-1} \equiv 1 \pmod{n}$.

Ein Fermat-Zeuge a für eine zusammengesetzte Zahl n liefert keine Information über die Primfaktoren von n . Deshalb ist der Fermat-Test in Abbildung 7.5 unten kein Faktorisierungsalgorithmus. In Übung 7.4 ist zu zeigen, dass 2 für jedes zusammengesetzte $n \leq 340$ ein Fermat-Zeuge ist, aber ein Fermat-Lügner für $341 = 11 \cdot 31$. Die Zahl 3 jedoch ist ein Fermat-Zeuge für 341, denn $3^{340} \equiv 56 \pmod{341}$.

Die Existenz von Fermat-Lügnern zeigt, dass die Umkehrung des Kleinen Fermat nicht gilt, d.h., die Bedingung $a^{n-1} \equiv 1 \pmod{n}$ impliziert nicht, dass n eine Primzahl ist. Wie oben bemerkt ist beispielsweise $2^{340} \equiv 1 \pmod{341}$, aber 341 ist zusammengesetzt. Man kann allerdings eine etwas schwächere Aussage als die Umkehrung des Kleinen Fermat zeigen.

Lemma 7.8.

Sei $n \geq 2$ eine beliebige natürliche Zahl.

1. Gibt es für eine Zahl a , $1 \leq a < n$, ein $k \geq 1$ mit $a^k \equiv 1 \pmod{n}$, so ist $a \in \mathbb{Z}_n^*$.
2. Ist n eine ungerade zusammengesetzte Zahl, so existiert ein Fermat-Zeuge für n .

Beweis. 1. Angenommen, es gibt ein $k \geq 1$, so dass $a^k \equiv 1 \pmod{n}$ für ein a mit $1 \leq a < n$ gilt. Dann ist $a \cdot a^{k-1} \equiv 1 \pmod{n}$. Man kann leicht zeigen, dass $a \in \mathbb{Z}_n^*$ genau dann gilt, wenn es ein $b \in \mathbb{Z}_n$ mit $a \cdot b \equiv 1 \pmod{n}$ gibt, siehe Übung 7.5. Setzt man nun $b = a^{k-1} \in \mathbb{Z}_n$, so folgt die Behauptung.

2. Wir zeigen die Kontraposition. Angenommen, n hat keinen Fermat-Zeugen. Für jedes a mit $1 \leq a < n$ folgt dann $a^{n-1} \not\equiv 1 \pmod{n}$. Nach der ersten Aussage dieses Lemmas gilt $\mathbb{Z}_n^* = \{1, 2, \dots, n-1\}$. Aber das bedeutet, dass n keine nichttrivialen Teiler hat und somit eine Primzahl ist. \square

Man kann sogar eine Behauptung beweisen, die noch stärker ist als der zweite Teil von Lemma 7.8: Für jede ungerade zusammengesetzte Zahl n enthält die Menge

$$\{1, 2, \dots, n-1\} - \mathbb{Z}_n^* = \{a \mid 1 \leq a < n \text{ und } \text{ggT}(a, n) \neq 1\}$$

keinen Fermat-Lügner. Leider ist diese Menge für viele zusammengesetzte Zahlen sehr dünn. Im RSA-Kryptosystem etwa wählt man zwei Primzahlen, p und q mit $p \neq q$, und betrachtet ihr Produkt $n = pq$. Eine Zahl a erfüllt $\text{ggT}(a, n) \neq 1$ genau dann, wenn a ein Vielfaches von p oder q ist. Es gibt genau $p+q-2$ solche Zahlen in der Menge $\{1, 2, \dots, n-1\}$, sehr wenige also im Vergleich mit n , falls p und q etwa dieselbe Größe haben.

Beispiel 7.9 (Fermat-Zeuge und Fermat-Lügner). Sei $n = 143 = 11 \cdot 13$. Tabelle 7.4 zeigt alle Fermat-Zeugen und alle Fermat-Lügner für n . Die Fermat-Zeugen sind in drei Teilmengen unterteilt: Vielfache von p , Vielfache von q und Fermat-Zeugen in \mathbb{Z}_{143}^* . Die letztere Gruppe von Fermat-Zeugen ist weitaus größer als die Menge der Vielfachen von p oder q .

Tabelle 7.4. Fermat-Zeugen und Fermat-Lügner für $n = 143$

Vielfache von 11	11	22	33	44	55	66	77	88	99	110	121	132
Vielfache von 13	13	26	39	52	65	78	91	104	117	130		
Fermat-Zeugen in \mathbb{Z}_{143}^*	2	3	4	5	6	7	8	9	10	14	15	16
	17	18	19	20	21	23	24	25	27	28	29	30
	31	32	34	35	36	37	38	40	41	42	43	45
	46	47	48	49	50	51	53	54	56	57	58	59
	60	61	62	63	64	67	68	69	70	71	72	73
	74	75	76	79	80	81	82	83	84	85	86	87
	89	90	92	93	94	95	96	97	98	100	101	102
	103	105	106	107	108	109	111	112	113	114	115	116
	118	119	120	122	123	124	125	126	127	128	129	133
	134	135	136	137	138	139	140	141				
Fermat-Lügner	1	12	131	142								

Abbildung 7.5 zeigt den Fermat-Test, unseren ersten randomisierten Primzahltest. Gemäß Tabelle 7.4 aus Beispiel 7.9 ist die Wahrscheinlichkeit für die Antwort

```

FERMAT( $n$ ) {
    //  $n \geq 3$  ist eine ungerade Zahl
    Wähle zufällig eine Zahl  $a \in \{2, 3, \dots, n-2\}$  unter Gleichverteilung;
    if ( $a^{n-1} \not\equiv 1 \pmod{n}$ ) return „ $n$  ist zusammengesetzt“ und halte;
    else return „ $n$  ist prim“ und halte;
}

```

Abb. 7.5. Fermat-Test

„ n ist zusammengesetzt“ bei Eingabe $n = 143$ gleich $138/140 \approx 0.9857$, denn es gibt nur zwei Fermat-Lügner außer den trivialen, 1 und 142. Anders gesagt übersteigt die Anzahl der Fermat-Zeugen die der Fermat-Lügner in diesem Beispiel ganz deutlich. Wäre dies für alle ungeraden zusammengesetzten Zahlen der Fall, dann wäre der Fermat-Test ein so genannter „*no-biased*“ Monte-Carlo-Algorithmus für das Primzahl-Problem oder, äquivalent dazu, ein „*yes-biased*“ Monte-Carlo-Algorithmus für das komplementäre Problem, die Zusammengesetztheit einer gegebenen Zahl zu entscheiden.

Zur Erinnerung: In Abschnitt 6.2.1 wurde gesagt, dass wenn ein *yes-biased* Monte-Carlo-Algorithmus für ein Entscheidungsproblem L die Antwort „ja“ gibt, dann ist die Eingabe in L ; ist umgekehrt die Eingabe in L , dann tritt die (korrekte) Antwort „ja“ mit einer Wahrscheinlichkeit von mindestens $1/2$ auf, aber eine inkorrekte Antwort „nein“ kann mit einer Wahrscheinlichkeit kleiner als $1/2$ gegeben werden. Demnach geben *yes-biased* Monte-Carlo-Algorithmen stets eine korrekte „Ja“-Antwort, aber möglicherweise eine inkorrekte „Nein“-Antwort.

Wenn der Fermat-Test aus Abbildung 7.5 die Antwort „ n ist zusammengesetzt“ gibt, dann war zufällig ein Fermat-Zeuge gezogen worden, woraus folgt, dass n tatsächlich zusammengesetzt ist. Wenn jedoch n zusammengesetzt ist, dann gibt der Fermat-Test diese Antwort nicht für jedes gegebene n mit Wahrscheinlichkeit mindestens $1/2$ und ist daher kein Monte-Carlo-Algorithmus. Im Extremfall gibt es nämlich zusammengesetzte Zahlen n , für die sämtliche Elemente von \mathbb{Z}_n^* Fermat-Lügner sind. Solche störrischen Zahlen heißen *Carmichael-Zahlen*; das kleinste Beispiel einer solchen Zahl ist $561 = 3 \cdot 11 \cdot 17$. Carmichael-Zahlen haben in $\mathbb{Z}_n - \mathbb{Z}_n^*$ auch Fermat-Zeugen, d.h., die Fermat-Zeugen einer Carmichael-Zahl n sind mit n nicht teilerfremd. Es ist bekannt, dass es unendlich viele Carmichael-Zahlen gibt.

Satz 7.12 unten sagt, dass der Fermat-Test für „gute“ Eingaben eine hinreichend hohe Erfolgswahrscheinlichkeit hat. Zunächst werden dafür einige grundlegende gruppentheoretische Fakten benötigt. Lemma 7.10 gibt eine hinreichende Bedingung dafür an, wann eine Teilmenge einer endlichen Gruppe sogar eine Untergruppe ist. Dass die Gruppe dabei endlich ist, stellt eine wesentliche Voraussetzung dar. Beispielsweise gilt zwar $\mathbb{N} \subseteq \mathbb{Z}$ und $0 \in \mathbb{N}$ und ist \mathbb{N} unter Addition abgeschlossen, aber $(\mathbb{N}, +)$ ist dennoch keine Untergruppe von $(\mathbb{Z}, +)$, weil \mathbb{Z} nämlich nicht endlich ist.

Nach Definition 2.34 in Abschnitt 2.4.1 ist $\mathfrak{H} = (H, \circ)$ eine Untergruppe einer Gruppe $\mathfrak{G} = (G, \circ)$, falls $H \subseteq G$ gilt und \mathfrak{H} die Gruppenaxiome erfüllt: \mathfrak{H} ist unter der Gruppenoperation \circ abgeschlossen, welche auf H assoziativ ist, \mathfrak{H} erbt das neutrale Element e von \mathfrak{G} , und alle Elemente von \mathfrak{H} besitzen Inverse.

Lemma 7.10. Ist $\mathfrak{G} = (G, \circ)$ eine endliche Gruppe mit neutralem Element e und ist H eine Teilmenge von G , so dass $e \in H$ und H unter der Gruppenoperation \circ abgeschlossen ist, dann ist $\mathfrak{H} = (H, \circ)$ eine Untergruppe von \mathfrak{G} .

Beweis. Es genügt zu zeigen, dass alle Elemente von \mathfrak{H} Inverse haben. Definiere für jedes Element x von H die Funktion $\sigma_x : H \rightarrow H$ durch

$$\sigma_x(y) = x \circ y.$$

Da H unter der Gruppenoperation \circ abgeschlossen ist, ist jede Funktion σ_x wohldefiniert. Da $\mathfrak{G} = (G, \circ)$ eine Gruppe ist, ist jede Funktion σ_x injektiv. Weil \mathfrak{G} endlich ist, muss jedes σ_x eine Bijektion auf H sein. Aus diesem Grund und weil e , das neutrale Element in \mathfrak{G} , zu H gehört, existiert ein Element $z \in H$, so dass $\sigma_x(z) = x \circ z = e$ gilt. Also ist $z = x^{-1}$ das inverse Element von x in \mathfrak{H} . \square

\mathbb{Z}_n^* ist bekanntlich eine endliche multiplikative Gruppe der Ordnung $\varphi(n)$. Das folgende Lemma sagt, dass die Ordnung einer beliebigen endlichen Gruppe \mathfrak{G} von der Ordnung einer jeden Untergruppe von \mathfrak{G} geteilt wird. Der Beweis von Lemma 7.11, auch bekannt als der Satz von Lagrange, wird dem Leser als Übung 7.6 überlassen.

Lemma 7.11 (Lagrange). Ist $\mathfrak{G} = (G, \circ)$ eine endliche Gruppe und ist \mathfrak{H} eine Untergruppe von \mathfrak{G} , dann teilt die Ordnung von \mathfrak{H} die Ordnung von \mathfrak{G} .

Nun kann das folgende Resultat bewiesen werden, das für viele zusammengesetzte Zahlen (nämlich für alle Zahlen, die keine Carmichael-Zahlen sind) eine gute Abschätzung für die Erfolgswahrscheinlichkeit des Fermat-Tests liefert.

Satz 7.12. Ist $n \geq 3$ eine ungerade zusammengesetzte Zahl, die mindestens einen Fermat-Zeugen in \mathbb{Z}_n^* hat, so gibt der Fermat-Test bei Eingabe n die korrekte Antwort „ n ist zusammengesetzt“ mit Wahrscheinlichkeit mindestens $1/2$.

Beweis. Nach der ersten Aussage von Lemma 7.8 ist die Menge der Fermat-Lügner für eine Zahl n eine Teilmenge von \mathbb{Z}_n^* . Definiere diese Menge für jedes $n \geq 3$ durch

$$\text{F-Liars}_n = \{a \mid 1 \leq a < n \text{ und } a^{n-1} \equiv 1 \pmod{n}\}.$$

Es wird nun gezeigt, dass F-Liars_n sogar eine Untergruppe von \mathbb{Z}_n^* ist. Weil \mathbb{Z}_n^* eine endliche multiplikative Gruppe ist, kann Lemma 7.10 angewandt werden. Demnach genügt es zu zeigen, dass F-Liars_n das neutrale Element von \mathbb{Z}_n^* besitzt und unter Multiplikation modulo n abgeschlossen ist. Das neutrale Element von \mathbb{Z}_n^* ist 1, und wie oben erwähnt ist 1 trivialerweise in F-Liars_n , siehe Übung 7.4(c). Um zu zeigen, dass die Menge F-Liars_n unter Multiplikation modulo n abgeschlossen ist, seien a und b zwei beliebige Elemente von F-Liars_n , d.h., es gilt $a^{n-1} \equiv 1 \pmod{n}$ und $b^{n-1} \equiv 1 \pmod{n}$. Dann folgt

$$(a \cdot b)^{n-1} \equiv a^{n-1} \cdot b^{n-1} \equiv 1 \cdot 1 \equiv 1 \pmod{n},$$

also ist auch $a \cdot b$ in F-Liars_n . Nach Lemma 7.10 ist F-Liars_n eine Untergruppe von \mathbb{Z}_n^* .

Laut Annahme gibt es mindestens einen Fermat-Zeugen für n in \mathbb{Z}_n^* . Somit ist F-Liars_n sogar eine *echte* Untergruppe von \mathbb{Z}_n^* . Dieser Fakt kann ausgenutzt werden, um eine noch viel bessere Schranke als $\varphi(n) - 1$ (die Größe von \mathbb{Z}_n^* minus den einen Fermat-Zeugen in \mathbb{Z}_n^*) für die Größe von F-Liars_n zu zeigen. Da nämlich F-Liars_n eine echte Untergruppe von \mathbb{Z}_n^* ist, folgt aus Lemma 7.11, dass die Größe von F-Liars_n ein nichtrivialer Teiler von $\varphi(n)$ ist. Weil aber $\varphi(n) < n - 1$ gilt, ist die Größe von F-Liars_n höchstens $(n - 2)/2$.

Betrachte das Ereignis, dass eine vom Fermat-Test zufällig aus $\{2, 3, \dots, n - 2\}$ gezogene Zahl a ein Fermat-Lügner ist (abgesehen von den trivialen, 1 und $n - 1$). Es folgt, dass dieses Ereignis mit einer Wahrscheinlichkeit von höchstens

$$\frac{(n - 2)/2 - 2}{n - 3} = \frac{n - 6}{2(n - 3)} < \frac{1}{2}$$

eintritt, wie gewünscht. \square

7.2.2 Miller–Rabin-Test

Wie schon erwähnt ist der Grund, weshalb der Fermat-Test kein Monte-Carlo-Algorithmus für Primes ist, dass \mathbb{Z}_n^* für unendlich viele zusammengesetzte Zahlen n zu viele Fermat-Lügner enthält, nämlich für alle Carmichael-Zahlen n . In diesem Fall besteht ganz \mathbb{Z}_n^* aus nichts anderem als Fermat-Lügnern. Ist also eine Carmichael-Zahl n die Eingabe, so gibt der Fermat-Test die falsche Antwort „ n ist prim“ mit der Wahrscheinlichkeit

$$\frac{\varphi(n) - 2}{n - 3} > \frac{\varphi(n) - 2}{n} = \frac{\varphi(n)}{n} - \frac{2}{n}.$$

Da man zeigen kann, dass $\varphi(n)/n$ gleich $\prod(1 - 1/p)$ ist, wobei das Produkt über alle Primfaktoren p von n genommen wird, ist die Fehlerwahrscheinlichkeit des Fermat-Tests bei Eingabe einer Carmichael-Zahl n unangenehm nahe an eins. Diesen Nachteil vermeidet der Miller–Rabin-Test, einer der populärsten randomisierten Primzahltests, der in Abbildung 7.6 zu sehen ist.

Satz 7.19 unten sagt, dass der Miller–Rabin-Test ein „no-biased“ Monte-Carlo-Algorithmus für Primes ist, das heißt, seine „Nein“-Antworten sind stets verlässlich, wohingegen seine „Ja“-Antworten fehlerhaft sein können. Anders gesagt, Primes ist in coRP. Die Klasse RP, die in Definition 6.2 eingeführt wurde, enthält genau die Probleme A , für die es eine NPTM M gibt, so dass für jede Eingabe x gilt: Ist $x \in A$, so akzeptiert $M(x)$ mit einer Wahrscheinlichkeit von mindestens $1/2$, und ist $x \notin A$, so lehnt $M(x)$ mit Sicherheit ab. Und $\text{coRP} = \{\overline{A} \mid A \in \text{RP}\}$ ist die Klasse der Komplemente von RP-Mengen.

Der Beweis von Satz 7.19 erfordert einige grundlegende zahlentheoretische Fakten, von denen manche hier ohne Beweis angegeben werden. Der Beweis von Lemma 7.13 wird dem Leser als Übung 7.7(a) überlassen.

```

MILLER-RABIN( $n$ ) {
    //  $n \geq 3$  ist eine ungerade Zahl
    Bestimme die Darstellung  $n - 1 = 2^k m$ , wobei  $m$  ungerade ist;
    Wähle zufällig eine Zahl  $a \in \{1, 2, \dots, n - 1\}$  unter Gleichverteilung;
     $x := a^m \bmod n$ ;
    if ( $x \equiv 1 \pmod{n}$ ) return „ $n$  ist prim“ und halte;
    for ( $j = 0, 1, \dots, k - 1$ ) {
        if ( $x \equiv -1 \pmod{n}$ ) return „ $n$  ist prim“ und halte;
        else  $x := x^2 \bmod n$ ;
    }
    return „ $n$  ist zusammengesetzt“ und halte;
}

```

Abb. 7.6. Miller–Rabin-Test

Lemma 7.13. *Jede Carmichael-Zahl ist das Produkt von mindestens drei verschiedenen Primzahlen.*

Definition 2.43 stellte den Begriff der quadratischen Reste modulo n vor. Ein Element $x \in \mathbb{Z}_n^*$ ist ein quadratischer Rest modulo n , falls es ein a gibt, $1 \leq a < n$, so dass $x \equiv a^2 \pmod{n}$ gilt. Ist $x = 1$, dann heißt ein solches a (*quadratische*) Wurzel von 1 modulo n . Trivialerweise sind 1 und $n - 1$ stets Wurzeln von 1 modulo n , denn

$$1^2 \equiv 1 \pmod{n} \quad \text{und} \quad (n-1)^2 \equiv (-1)^2 \equiv 1 \pmod{n}.$$

Wenn n eine Primzahl ist, dann hat sie keine Wurzeln von 1 modulo n außer den trivialen. Der Beweis von Lemma 7.14 wird dem Leser als Übung 7.7(b) überlassen.

Lemma 7.14. *Jede ungerade Primzahl n hat nur die beiden trivialen Wurzeln von 1 modulo n , nämlich $\pm 1 \pmod{n}$.*

Folglich muss eine Zahl n , die eine nichttriviale Wurzel von 1 modulo n hat, zusammengesetzt sein. Ist umgekehrt $n = p_1 p_2 \cdots p_k$ zusammengesetzt, wobei die p_i ungerade Primzahlen sind, dann kann der Chinesische Restesatz (siehe Satz 2.46) angewandt werden, um zu zeigen, dass n genau 2^k Wurzeln von 1 modulo n hat. Genauer gesagt sind die Wurzeln von 1 modulo n alle Zahlen a , $1 \leq a < n$, für die $a \pmod{p_i} \in \{1, p_i - 1\}$ für $1 \leq i \leq k$ gilt. Der Versuch, nichttriviale Wurzeln von 1 modulo n dadurch zu finden, dass man zufällig eine Zahl a zieht, ist deshalb hoffnungslos, außer wenn n ungewöhnlich viele Primfaktoren hat.

Beispiel 7.15 (Nichttriviale Wurzeln von 1 modulo n). In Fortsetzung von Beispiel 7.9 betrachten wir die zusammengesetzte Zahl $n = 143 = 11 \cdot 13$. Da 143 zwei Primfaktoren hat, gibt es vier Wurzeln von 1 modulo 143, nämlich 1, 12, 131 und 142. Die nichttrivialen Wurzeln von 1 modulo 143 sind 12 und 131. In diesem Beispiel stimmen die Wurzeln von 1 modulo 143 zufällig gerade mit den Fermat-Lügnern für 143 überein, siehe Tabelle 7.4. Im Allgemeinen ist dies jedoch nicht der Fall.

Der Miller–Rabin-Test beruht auf einem Kriterium, das den Kleinen Fermat verschärft. Um dieses Kriterium zu beschreiben, wird nun der Begriff des Miller–Rabin-Zeugen und der des Miller–Rabin-Lügners eingeführt (kurz: MR-Zeuge und MR-Lügner). Intuitiv bezeugt ein MR-Zeuge a , dass eine Zahl n zusammengesetzt ist, und ein MR-Lügner a lässt n wie eine Primzahl aussehen, obwohl n in Wirklichkeit zusammengesetzt ist.

Definition 7.16 (MR-Zeuge und MR-Lügner). Sei $n \geq 3$ eine beliebige ungerade Zahl, und sei a eine beliebige Zahl in \mathbb{Z}_n^* . Definiere $m = (n - 1)/2^k$, wobei $k = \max\{j \in \mathbb{N} \mid 2^j \text{ teilt } n - 1\}$ gilt.

- Wir sagen, dass a ein MR-Zeuge für n ist, falls weder (7.8) noch (7.9) gilt:

$$a^m \equiv 1 \pmod{n}; \quad (7.8)$$

$$(\exists j \in \{0, 1, \dots, k-1\}) [a^{2^j m} \equiv -1 \pmod{n}]. \quad (7.9)$$

- Wir sagen, dass a ein MR-Lügner für n ist, falls n eine zusammengesetzte Zahl und a kein MR-Zeuge für n ist.

Beispiel 7.17 (MR-Zeuge und MR-Lügner). Betrachte $n = 561 = 3 \cdot 11 \cdot 17$, die bereits erwähnte kleinste Carmichael-Zahl. Tabelle 7.5 zeigt 22 ausgewählte MR-Zeugen für 561 und vier ausgewählte MR-Lügner für 561. Wie man sieht, sind 13 dieser 22 MR-Zeugen für 561 in \mathbb{Z}_{561}^* , während neun von ihnen Vielfache der Primfaktoren von 561 sind. Im Gegensatz dazu ist keines der Elemente \mathbb{Z}_{561}^* ein Fermat-Zeuge für 561, weil 561 eine Carmichael-Zahl ist. Somit ist der Fermat-Test nicht in der Lage, die Zusammengesetztheit von 561 zu entdecken.

Gemäß Definition 7.16 ergibt sich $k = 4$ und $m = 35$, weil $560 = 35 \cdot 2^4$ gilt und da keine größere Zweierpotenz 560 teilt. In der Spalte ganz links in Tabelle 7.5 ist jede der Zahlen a außer denen in $\{1, 50, 101, 460\}$ ein MR-Zeuge, denn weder Bedingung (7.8) noch Bedingung (7.9) ist erfüllt. Jedoch ist (7.9) für $a = 50$ wahr:

$$50^{2^0 \cdot 35} = 50^{35} \equiv -1 \pmod{561},$$

und ebenso für $a = 101$. Somit sind sowohl 50 als auch 101 MR-Lügner. Und 1 und 460 sind auch MR-Lügner, denn sie erfüllen die Bedingung (7.8):

$$1^{35} \equiv 460^{35} \equiv 1 \pmod{561}.$$

Nach der Definition von k und m ergibt sich $a^{n-1} = a^{2^k m}$ für jedes a ; die entsprechenden Werte von $a^{560} \equiv a^{2^4 \cdot 35} \pmod{561}$ stehen in der Spalte ganz rechts in Tabelle 7.5. Wie oben erwähnt sind sämtliche $a \in \mathbb{Z}_{561}^*$ in dieser Spalte Fermat-Lügner für 561, denn 561 ist eine Carmichael-Zahl.

Das folgende Lemma ist leicht zu beweisen, siehe Übung 7.7(c) und auch den Beweis von Satz 7.19.

Lemma 7.18. Wenn es einen MR-Zeugen für n gibt, dann ist n zusammengesetzt.

Tabelle 7.5. Einige MR-Zeugen und MR-Lügner für die Carmichael-Zahl 561

a	$a^{35} \bmod 561$	$a^{70} \bmod 561$	$a^{140} \bmod 561$	$a^{280} \bmod 561$	$a^{560} \bmod 561$
1	1	1	1	1	1
2	263	166	67	1	1
3	78	474	276	441	375
4	166	67	1	1	1
5	23	529	463	67	1
6	318	144	540	441	375
7	241	298	166	67	1
8	461	463	67	1	1
9	474	276	441	375	375
10	439	298	166	67	1
11	209	484	319	220	154
12	45	342	276	441	375
13	208	67	1	1	1
14	551	100	463	67	1
15	111	540	441	375	375
16	67	1	1	1	1
17	527	34	34	34	34
18	120	375	375	375	375
19	76	166	67	1	1
20	452	100	463	67	1
30	21	441	375	375	375
40	505	331	166	67	1
50	560	1	1	1	1
101	560	1	1	1	1
452	320	298	166	67	1
460	1	1	1	1	1

Satz 7.19. Primes ist in coRP.

Beweis. Zu zeigen ist, dass der Miller–Rabin-Test Primes mit einer einseitigen Fehlerwahrscheinlichkeit in Polynomialzeit akzeptiert. Zunächst wird gezeigt, dass wenn die Eingabe n tatsächlich eine Primzahl ist, dann muss der Algorithmus aus Abbildung 7.6 auch antworten, dass n prim ist. Für einen Widerspruch sei angenommen, dass n zwar prim ist, doch der Miller–Rabin-Test hält mit der Ausgabe: „ n ist zusammengesetzt“. Wir zeigen, dass er dann einen MR-Zeugen a für n gefunden haben muss, was nach Lemma 7.18 heißt, dass n im Widerspruch zur Annahme zusammengesetzt ist.

Da der Miller–Rabin-Test „ n ist zusammengesetzt“ ausgibt, gilt $a^m \not\equiv 1 \pmod n$, wobei $a \in \{1, 2, \dots, n-1\}$ die zufällig vom Miller–Rabin-Test gewählte Zahl ist. Weil x in jedem Durchlauf der for-Schleife quadriert wird, testet der Algorithmus nacheinander die Werte $a^m, a^{2m}, \dots, a^{2^{k-1}m}$. Für keinen dieser Werte antwortet der Algorithmus, dass n eine Primzahl sei. Es folgt, dass für jedes j mit $0 \leq j \leq k-1$ gilt:

$$a^{2^j m} \not\equiv -1 \pmod{n}.$$

Da $n - 1 = 2^k m$ ist, folgt aus dem Kleinen Fermat (siehe Korollar 2.39), dass $a^{2^k m} \equiv 1 \pmod{n}$ gilt. Also ist $a^{2^{k-1} m}$ eine Wurzel von 1 modulo n . Weil n nach unserer Voraussetzung prim ist, impliziert Lemma 7.14, dass es nur die zwei trivialen Wurzeln von 1 modulo n gibt, nämlich $\pm 1 \pmod{n}$, siehe auch Übung 7.7(b).

Da $a^{2^{k-1} m} \not\equiv -1 \pmod{n}$ gilt und n prim ist, folgt $a^{2^{k-1} m} \equiv 1 \pmod{n}$. Somit ist $a^{2^{k-2} m}$ ebenfalls eine Wurzel von 1 modulo n . Mit demselben Argument ergibt sich wieder $a^{2^{k-2} m} \equiv 1 \pmod{n}$. Durch wiederholte Anwendung dieses Arguments erhalten wir schließlich $a^m \equiv 1 \pmod{n}$, ein Widerspruch. Also gibt der Miller–Rabin–Test für jede Primzahl n korrekt „ n ist prim“ aus.

Nehmen wir umgekehrt an, dass n zusammengesetzt ist, so ist zu zeigen, dass der Miller–Rabin–Test die falsche Ausgabe „ n ist prim“, mit einer Wahrscheinlichkeit kleiner als $1/2$ macht. Jedoch können wir hier nicht dasselbe Argument wie im Beweis von Satz 7.12 anwenden: Wir können die Anzahl der MR-Lügner nämlich nicht beschränken, indem wir zeigen, dass die Menge

$$\text{MR-Liars}_n = \{a \mid a \text{ ist ein MR-Lügner für } n\}$$

eine echte Untergruppe von \mathbb{Z}_n^* bildet, denn dies ist im Allgemeinen nicht der Fall; einige Gegenbeispiele findet man in Übung 7.8(c). Deshalb wenden wir (in Fall 2 unten) ein anderes Argument an: Wir spezifizieren eine Obermenge von MR-Liars_n , die eine echte Untergruppe von \mathbb{Z}_n^* bildet. Unterscheide die folgenden zwei Fälle.

Fall 1: n ist keine Carmichael-Zahl. Es gilt $\text{MR-Liars}_n \subseteq \text{F-Liars}_n \neq \mathbb{Z}_n^*$ und mit dem Argument aus Satz 7.12 kann gezeigt werden, dass die Wahrscheinlichkeit dafür, einen MR-Lügner zu erwischen, wenn man zufällig eine Zahl aus $\{1, 2, \dots, n-1\}$ zieht, kleiner als $1/2$ ist.

Fall 2: n ist eine Carmichael-Zahl. In diesem Fall gilt $\text{MR-Liars}_n \subseteq \text{F-Liars}_n = \mathbb{Z}_n^*$, also ist die Menge der Fermat-Lügner keine *echte* Untergruppe von \mathbb{Z}_n^* . Um eine echte Untergruppe von \mathbb{Z}_n^* zu finden, die gleichzeitig sämtliche MR-Lügner enthält, seien m und k wie in Definition 7.16 gewählt: $m = (n-1)/2^k$ ist ungerade und $k = \max\{j \in \mathbb{N} \mid 2^j \text{ teilt } n-1\}$. Weil m ungerade ist, gilt

$$(n-1)^m \equiv (-1)^m \equiv -1 \pmod{n}.$$

Folglich gibt es (mindestens) einen MR-Lügner a , der (7.9) erfüllt. Sei j_{\max} das größte $j \in \{0, 1, \dots, k-1\}$, so dass $a^{2^j m} \equiv -1 \pmod{n}$ gilt. Da n eine Carmichael-Zahl ist, folgt

$$a^{2^k m} = a^{n-1} \equiv 1 \pmod{n},$$

also ist $j_{\max} < k$. Definiere die Menge

$$\text{MR-LIARS}_n = \{a \mid 0 \leq a < n \text{ und } a^{m \cdot 2^{j_{\max}}} \equiv \pm 1 \pmod{n}\}.$$

Die folgende Behauptung sagt, dass MR-LIARS_n die gewünschten Eigenschaften hat.

Behauptung 7.20. 1. $\text{MR-Liars}_n \subseteq \text{MR-LIARS}_n$.
 2. MR-LIARS_n ist eine echte Untergruppe von \mathbb{Z}_n^* .

Beweis von Behauptung 7.20. 1. Sei a ein beliebiger MR-Lügner für n . Also gilt für a entweder die Bedingung (7.8) oder die Bedingung (7.9). Falls a die Bedingung (7.8) erfüllt (falls also $a^m \equiv 1 \pmod{n}$ gilt), dann ist $a^{m \cdot 2^{j_{\max}}} \equiv 1 \pmod{n}$, woraus folgt, dass a in MR-LIARS_n liegt.

Falls a jedoch die Bedingung (7.9) erfüllt (falls also $a^{2^j} \equiv -1 \pmod{n}$ für ein j mit $0 \leq j < k$ gilt), dann ist $j \leq j_{\max}$ gemäß der Definition von j_{\max} . Ist dabei $j = j_{\max}$, so ist a unmittelbar in MR-LIARS_n . Ist jedoch $j < j_{\max}$, so gilt:

$$a^{m \cdot 2^{j_{\max}}} \equiv a^{m \cdot 2^j \cdot 2^{j_{\max}-j}} \equiv (a^{m \cdot 2^j})^{2^{j_{\max}-j}} \equiv 1 \pmod{n},$$

woraus ebenfalls folgt, dass a in MR-LIARS_n liegt.

2. Da \mathbb{Z}_n^* eine endliche Gruppe ist, folgt aus Lemma 7.10, dass MR-LIARS_n eine Untergruppe von \mathbb{Z}_n^* ist. Das neutrale Element von \mathbb{Z}_n^* ist selbstverständlich in MR-LIARS_n , denn

$$1^{m \cdot 2^{j_{\max}}} \equiv 1 \pmod{n}.$$

Um zu beweisen, dass MR-LIARS_n unter Multiplikation abgeschlossen ist, der Gruppenoperation in \mathbb{Z}_n^* , seien zwei beliebige Elemente, a und b , von MR-LIARS_n gewählt. Dann sind sowohl $a^{m \cdot 2^{j_{\max}}}$ als auch $b^{m \cdot 2^{j_{\max}}}$ kongruent zu $\pm 1 \pmod{n}$.

Es gilt $1 \cdot 1 = 1$ und $1(n-1) \equiv -1 \pmod{n}$ und $(n-1)(n-1) \equiv 1 \pmod{n}$. Folglich ist

$$(a \cdot b)^{m \cdot 2^{j_{\max}}} \equiv a^{m \cdot 2^{j_{\max}}} \cdot b^{m \cdot 2^{j_{\max}}} \equiv \pm 1 \pmod{n}.$$

Somit ist das Produkt $a \cdot b$ in MR-LIARS_n . Nach Lemma 7.10 ist MR-LIARS_n eine Untergruppe von \mathbb{Z}_n^* .

Um nun zu zeigen, dass MR-LIARS_n eine *echte* Untergruppe von \mathbb{Z}_n^* ist, wenden wir Lemma 7.13 an, welches sagt, dass jede Carmichael-Zahl das Produkt von mindestens drei verschiedenen Primfaktoren ist. Somit kann n als $n = n_1 \cdot n_2$ für ungerade Zahlen n_1 und n_2 mit $\text{ggT}(n_1, n_2) = 1$ geschrieben werden.

Sei \hat{a} ein fest gewählter MR-Lügner mit $\hat{a}^{m \cdot 2^{j_{\max}}} \equiv -1 \pmod{n}$; die Existenz von \hat{a} folgt aus der Definition von j_{\max} . Sei $a_1 = \hat{a} \pmod{n_1}$. Da $\text{ggT}(n_1, n_2) = 1$ gilt, folgt aus dem Chinesischen Restesatz (siehe Satz 2.46), dass das System von zwei Kongruenzen:

$$a \equiv a_1 \pmod{n_1}$$

$$a \equiv 1 \pmod{n_2}$$

die eindeutige Lösung $a = a_1 \cdot n_2 \cdot n_2^{-1} + 1 \cdot n_1 \cdot n_1^{-1} \pmod{n}$ hat, wobei n_1^{-1} das inverse Element von n_1 in $\mathbb{Z}_{n_2}^*$ und n_2^{-1} das inverse Element von n_2 in $\mathbb{Z}_{n_1}^*$ ist. Wir zeigen, dass diese Lösung a in der Differenz $\mathbb{Z}_n^* - \text{MR-LIARS}_n$ liegt.

Um zu sehen, dass a nicht zu MR-LIARS_n gehört, stellen wir fest, dass gilt:

$$a^{m \cdot 2^{j_{\max}}} \equiv \hat{a}^{m \cdot 2^{j_{\max}}} \equiv -1 \pmod{n_1}; \quad (7.10)$$

$$a^{m \cdot 2^{j_{\max}}} \equiv 1^{m \cdot 2^{j_{\max}}} \equiv 1 \pmod{n_2}, \quad (7.11)$$

wobei (7.10) aus $a \equiv a_1 \equiv \hat{a} \pmod{n_1}$ folgt. Die Kongruenz (7.10) impliziert $a^{m \cdot 2^{j_{\max}}} \not\equiv 1 \pmod{n}$, und die Kongruenz (7.11) impliziert $a^{m \cdot 2^{j_{\max}}} \not\equiv -1 \pmod{n}$. Also ist $a^{m \cdot 2^{j_{\max}}} \not\equiv \pm 1 \pmod{n}$. Es folgt, dass a nicht in MR-LIARS_n liegt. Um zu sehen, dass a zu \mathbb{Z}_n^* gehört, stellen wir fest, dass aus (7.10) und (7.11) folgt:

$$\begin{aligned} a^{m \cdot 2^{j_{\max}} + 1} &\equiv 1 \pmod{n_1}; \\ a^{m \cdot 2^{j_{\max}} + 1} &\equiv 1 \pmod{n_2}. \end{aligned}$$

Nach dem Chinesischen Restesatz ist $a^{m \cdot 2^{j_{\max}} + 1} \equiv 1 \pmod{n}$. Aus der ersten Aussage von Lemma 7.8 folgt nun, dass a in \mathbb{Z}_n^* liegt. \square Behauptung 7.20

Mit Behauptung 7.20 kann ein Argument ähnlich dem im Beweis von Satz 7.12 angewandt werden, um zu zeigen, dass auch im Fall 2 die Fehlerwahrscheinlichkeit des Miller–Rabin-Tests kleiner als $1/2$ ist.

Um den Beweis von Satz 7.19 abzuschließen, bleibt nur noch festzustellen, dass der Miller–Rabin-Test in Polynomialzeit läuft, siehe Übung 7.7(d). \square Satz 7.19

Anmerkung 7.21. 1. Mit einer raffinierteren Analyse kann man zeigen, dass die Fehlerwahrscheinlichkeit des Miller–Rabin-Tests den Schwellwert $1/4$ nicht übersteigt, siehe Problem 7.1.

2. Lässt man den Miller–Rabin-Test bei derselben Eingabe in hinreichend (aber immer noch nur polynomiell in $\log n$) vielen unabhängigen Versuchen laufen (d.h., die Wahl der Zufallszahl a ist in den einzelnen Versuchen unabhängig), dann kann die Fehlerwahrscheinlichkeit beliebig nahe an null herangebracht werden; die Details findet man in Satz 6.6 und seinem Beweis. Der formale Beweis dieser Behauptung wird dem Leser als Übung 7.7(e) überlassen.

7.2.3 Solovay–Strassen-Test

Solovay und Strassen entwickelten einen Primalitätstest, der auf anderen zahlentheoretischen Resultaten beruht als der Miller–Rabin-Test. Ihr *no-biased* Monte-Carlo-Algorithmus für das Primzahl-Problem kann im Beweis von Satz 7.19 alternativ zum Miller–Rabin-Test benutzt werden. Jedoch ist der Solovay–Strassen-Test weniger populär als der Miller–Rabin-Test, weil er in der Praxis etwas weniger effizient und weniger akkurat ist. Um den Solovay–Strassen-Test zu erklären, sind die folgenden Begriffe und Fakten aus Abschnitt 2.4.1 nötig, siehe die Definitionen 2.43 und 2.45:

- Ein *quadratischer Rest modulo n* ist ein Element a in \mathbb{Z}_n^* , so dass $a \equiv w^2 \pmod{n}$ für ein $w \in \mathbb{Z}_n$ gilt. Andernfalls heißt a *quadratischer Nichtrest modulo n*.

- Das *Euler-Kriterium* (siehe Satz 2.44) sagt, dass für jede ungerade Primzahl p eine Zahl a genau dann ein quadratischer Rest modulo p ist, wenn $a^{(p-1)/2} \equiv 1 \pmod{p}$ gilt.
- Das *Legendre-Symbol* $\left(\frac{a}{p}\right)$ für $a \in \mathbb{Z}_p$ und eine Primzahl p drückt die Eigenschaft von a , ein quadratischer Rest oder Nichtrest modulo p oder aber ein Vielfaches von p zu sein, wie folgt aus:

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{falls } a \equiv 0 \pmod{p} \\ 1 & \text{falls } a \text{ ein quadratischer Rest modulo } p \text{ ist} \\ -1 & \text{falls } a \text{ ein quadratischer Nichtrest modulo } p \text{ ist} \end{cases}$$

Nach dem Euler-Kriterium gilt $a^{(p-1)/2} \equiv 1 \pmod{p}$ genau dann, wenn $\left(\frac{a}{p}\right) = 1$ gilt. Ist andererseits a ein Vielfaches von p , so ist $a^{(p-1)/2} \equiv 0 \pmod{p}$, und ist a ein quadratischer Nichtrest modulo p , so gilt $a^{(p-1)/2} \equiv -1 \pmod{p}$, denn

$$\left(a^{(p-1)/2}\right)^2 \equiv a^{p-1} \equiv 1 \pmod{p} \quad \text{und} \quad a^{(p-1)/2} \not\equiv 1 \pmod{p}.$$

Zusammengefasst gilt $\left(\frac{a}{p}\right) \equiv a^{(p-1)/2} \pmod{p}$ für jede ungerade Primzahl p . Also kann das Legendre-Symbol effizient berechnet werden.

- Das *Jacobi-Symbol* $\left(\frac{a}{n}\right)$, welches das Legendre-Symbol auf zusammengesetzte „Nenner“ n erweitert, wobei $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ für Primzahlpotenzen $p_i^{e_i}$ gilt, wurde in Definition 2.45 wie folgt eingeführt:

$$\left(\frac{a}{n}\right) = \prod_{i=1}^k \left(\frac{a}{p_i}\right)^{e_i}. \quad (7.12)$$

Sei $n \geq 3$ eine ungerade Zahl. Aus den obigen Definitionen und Beobachtungen folgt, dass wenn n eine Primzahl ist, dann gilt für jedes a :

$$\left(\frac{a}{n}\right) \equiv a^{(n-1)/2} \pmod{n}. \quad (7.13)$$

Ist n jedoch zusammengesetzt, dann ist $\left(\frac{a}{n}\right) \equiv a^{(n-1)/2} \pmod{n}$ vielleicht wahr, vielleicht auch nicht. Die Kongruenz (7.13) stellt somit ein verlässliches Kriterium zur Überprüfung der Zusammengesetztheit von Zahlen bereit: Gilt $\left(\frac{a}{n}\right) \not\equiv a^{(n-1)/2} \pmod{n}$ für ein a , so muss n zusammengesetzt sein. Ist andererseits (7.13) für ein zufällig gewähltes a wahr, so ist n vielleicht prim, vielleicht aber auch nicht. Dieses Kriterium nützt der in Abbildung 7.7 dargestellte Solovay–Strassen-Test aus.

Zwei Fragen sind zu stellen: Ist, erstens, der Solovay–Strassen-Test ein effizienter Algorithmus? Zweitens, was ist seine Fehlerwahrscheinlichkeit? Wie schon erwähnt, wenn n eine Primzahl ist, dann sagt SOLOVAY–STRASSEN bei Eingabe n dies gemäß (7.13) mit Sicherheit. Was also ist die Wahrscheinlichkeit dafür, dass der Solovay–Strassen-Test für eine gegebene zusammengesetzte Zahl n die inkorrekte Antwort gibt, n wäre prim?

```

SOLOVAY-STRASSEN( $n$ ) { //  $n \geq 3$  ist eine ungerade Zahl
    Wähle zufällig eine Zahl  $a \in \{1, 2, \dots, n-1\}$  unter Gleichverteilung;
     $x := (\frac{a}{n})$ ;
    if ( $x = 0$ ) return „ $n$  ist zusammengesetzt“ und halte;
     $y := a^{(n-1)/2} \bmod n$ ;
    if ( $x \equiv y \pmod{n}$ ) return „ $n$  ist prim“ und halte;
    else return „ $n$  ist zusammengesetzt“ und halte;
}

```

Abb. 7.7. Solovay–Strassen-Test

Doch wenden wir uns zunächst der ersten Frage zu. Wir haben bereits gesehen, dass $a^{(n-1)/2} \bmod n$ mit dem Algorithmus für schnelles Potenzieren aus Abbildung 7.2 effizient berechnet werden kann. Und wenn n prim ist, dann genügt es nach Eulers Kriterium, $a^{(n-1)/2} \bmod n$ zu berechnen, um das Legendre-Symbol $(\frac{a}{n})$ auszuwerten. Ist n jedoch zusammengesetzt, so ist es noch nicht klar, ob und, falls ja, wie man das Jacobi-Symbol $(\frac{a}{n})$ effizient auswerten kann.

Da das Jacobi-Symbol durch die Zerlegung von n in Primzahlpotenzen definiert ist, könnte man vermuten, es sei notwendig, die Primfaktoren von n zu finden, um das Jacobi-Symbol $(\frac{a}{n})$ zu berechnen. Hätte man n jedoch bereits faktorisiert, so wäre ein Primzahltest für n natürlich völlig überflüssig. Auch ist das Faktorisieren selbst eine schwere Aufgabe, siehe Abschnitt 7.3. Zum Glück kann das Jacobi-Symbol $(\frac{a}{n})$ jedoch effizient berechnet werden, ohne dass man zuvor die Primfaktoren von n bestimmen muss. Dieser effiziente Algorithmus benutzt ganz wesentlich die folgenden Eigenschaften des Jacobi-Symbols, die hier ohne Beweis angegeben werden.

Behauptung 7.22 (Eigenschaften des Jacobi-Symbols).

1. Gesetz der quadratischen Reziprozität: *Für ungerade natürliche Zahlen m und n gilt*

$$\left(\frac{m}{n}\right) = \begin{cases} -\left(\frac{n}{m}\right) & \text{falls } m \equiv n \equiv 3 \pmod{4} \\ \left(\frac{n}{m}\right) & \text{sonst.} \end{cases}$$

2. Ist n eine ungerade natürliche Zahl und gilt $a \equiv b \pmod{n}$, so ist $\left(\frac{a}{n}\right) = \left(\frac{b}{n}\right)$.

3. Multiplikativität: Ist n eine ungerade natürliche Zahl und sind a und b ganze Zahlen, so gilt

$$\left(\frac{a \cdot b}{n}\right) = \left(\frac{a}{n}\right) \left(\frac{b}{n}\right).$$

Ist insbesondere $m = a \cdot 2^k$ und a ungerade, so ist $\left(\frac{m}{n}\right) = \left(\frac{a}{n}\right) \left(\frac{2}{n}\right)^k$.

4. Ist n eine ungerade natürliche Zahl, so gilt

$$\left(\frac{2}{n}\right) = \begin{cases} 1 & \text{falls } n \equiv \pm 1 \pmod{8} \\ -1 & \text{falls } n \equiv \pm 3 \pmod{8}. \end{cases}$$

5. Ist n eine ungerade natürliche Zahl, so gilt $\left(\frac{1}{n}\right) = 1$ und $\left(\frac{0}{n}\right) = 0$.

Ausgehend von den in Behauptung 7.22 aufgeführten zahlentheoretischen Eigenschaften soll in Übung 7.9(a) ein effizienter Algorithmus zur Berechnung des Jacobi-Symbols entworfen werden. Das folgende Beispiel illustriert die Anwendung dieser Eigenschaften.

Beispiel 7.23 (Berechnung des Jacobi-Symbols). Angenommen, wir möchten den Wert des Jacobi-Symbols $(\frac{5775}{6399})$ bestimmen. Sowohl $5775 = 3 \cdot 5^2 \cdot 7 \cdot 11$ als auch $6399 = 3^4 \cdot 79$ ist zusammengesetzt. Weil insbesondere 6399 keine Primzahl ist, ist $(\frac{5775}{6399})$ kein Legendre-Symbol.

Wende die Eigenschaften aus Behauptung 7.22 folgendermaßen an:

$$\begin{aligned}
 (\frac{5775}{6399}) &= -(\frac{6399}{5775}) && \text{nach Eigenschaft 1, denn } 5775 \equiv 6399 \equiv 3 \pmod{4} \\
 &= -(\frac{624}{5775}) && \text{nach Eigenschaft 2, denn } 6399 \equiv 624 \pmod{5775} \\
 &= -\left(\frac{39}{5775}\right) \left(\frac{2}{5775}\right)^4 && \text{nach Eigenschaft 3, denn } 624 = 39 \cdot 2^4 \\
 &= -\left(\frac{39}{5775}\right) && \text{nach Eigenschaft 4, denn } 5775 \equiv -1 \pmod{8} \\
 &= \left(\frac{5775}{39}\right) && \text{nach Eigenschaft 1, denn } 39 \equiv 5775 \equiv 3 \pmod{4} \\
 &= \left(\frac{3}{39}\right) && \text{nach Eigenschaft 2, denn } 5775 \equiv 3 \pmod{39} \\
 &= -\left(\frac{39}{3}\right) && \text{nach Eigenschaft 1, denn } 3 \equiv 39 \equiv 3 \pmod{4} \\
 &= -\left(\frac{0}{3}\right) && \text{nach Eigenschaft 2, denn } 39 \equiv 0 \pmod{3} \\
 &= 0 && \text{nach Eigenschaft 5.}
 \end{aligned}$$

Wählt SOLOVAY–STRASSEN(6399) also die Zufallszahl $a = 5775$, so erhält man die korrekte Ausgabe: „6399 ist zusammengesetzt“, denn es gilt $(\frac{5775}{6399}) = 0$.

Betrachten wir nun den Fall, dass SOLOVAY–STRASSEN(6399) die Zufallszahl $a = 1111$ gewählt hat. Die Berechnung des Jacobi-Symbol ergibt sich nun so:

$$\begin{aligned}
 (\frac{1111}{6399}) &\stackrel{(1)}{=} -(\frac{6399}{1111}) \stackrel{(2)}{=} -(\frac{844}{1111}) \stackrel{(3)}{=} -\left(\frac{211}{1111}\right) \left(\frac{2}{1111}\right)^2 \stackrel{(4)}{=} -\left(\frac{211}{1111}\right) \\
 &\stackrel{(1)}{=} \left(\frac{1111}{211}\right) \stackrel{(2)}{=} \left(\frac{56}{211}\right) \stackrel{(3)}{=} \left(\frac{7}{211}\right) \left(\frac{2}{211}\right)^3 \stackrel{(4)}{=} -\left(\frac{7}{211}\right) \\
 &\stackrel{(1)}{=} \left(\frac{211}{7}\right) \stackrel{(2)}{=} \left(\frac{1}{7}\right) \stackrel{(5)}{=} 1,
 \end{aligned}$$

wobei $\stackrel{(i)}{=}$ die Anwendung der Eigenschaft i aus Behauptung 7.22 bezeichnet. Andererseits muss $a^{(n-1)/2} \pmod{n}$ für $a = 1111$ und $n = 6399$ bestimmt werden, also $1111^{3199} \pmod{6399}$. Für diese Berechnung verwenden wir wieder die schnelle Potenzierung aus Abbildung 7.2. Die Binärentwicklung des Exponenten ist

$$3199 = 2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6 + 2^{10} + 2^{11},$$

und Tabelle 7.6 zeigt die Werte $a^{2^i} \pmod{n}$, die mit dem *Square-and-Multiply*-Algorithmus nacheinander berechnet werden. Multipliziert man die Werte in den grauen Feldern von Tabelle 7.6 und reduziert modulo 6399, so erhält man

Tabelle 7.6. Berechnung von $a^{(n-1)/2} \bmod n$ beim Solovay–Strassen-Test

a^{2^0}	a^{2^1}	a^{2^2}	a^{2^3}	a^{2^4}	a^{2^5}	a^{2^6}	a^{2^7}	a^{2^8}	a^{2^9}	$a^{2^{10}}$	$a^{2^{11}}$
1111	5713	3469	3841	3586	3805	3487	1069	3739	4705	2884	5155

$$1111^{3199} \equiv 6088 \bmod 6399.$$

Somit ist $\left(\frac{1111}{6399}\right) = 1 \neq 6088 \equiv 1111^{3199} \bmod 6399$, und SOLOVAY–STRASSEN(6399) gibt für die Zufallszahl $a = 1111$ die korrekte Antwort „6399 ist zusammengesetzt“.

Wählt SOLOVAY–STRASSEN(6399) jedoch eine Zufallszahl $a \in \{1, 6398\}$, so erhält man die inkorrekte Ausgabe „6399 ist prim“, denn

$$\left(\frac{1}{6399}\right) \stackrel{(5)}{=} 1 = 1^{3199} \bmod 6399, \quad \text{und} \quad (7.14)$$

$$\begin{aligned} \left(\frac{6398}{6399}\right) &\stackrel{(3)}{=} \left(\frac{3199}{6399}\right) \left(\frac{2}{6399}\right) \stackrel{(4)}{=} \left(\frac{3199}{6399}\right) \stackrel{(1)}{=} -\left(\frac{6399}{3199}\right) \stackrel{(2)}{=} -\left(\frac{1}{3199}\right) \\ &\stackrel{(5)}{=} -1 = (-1)^{3199} \bmod 6399. \end{aligned} \quad (7.15)$$

Das in (7.14) und (7.15) gezeigte Verhalten ist kein Zufall. Für eine gegebene zusammengesetzte Zahl n gibt der Solovay–Strassen-Algorithmus aus Abbildung 7.7 stets die falsche Antwort „ n ist prim“, falls er eine Zufallszahl $a \in \{1, n-1\}$ gewählt hat, siehe Übung 7.9(b). Deshalb kann die Zufallszahl a im Solovay–Strassen-Test ebensogut nur aus dem Bereich $\{2, 3, \dots, n-2\}$ gewählt werden.

Nun wenden wir uns der zweiten der oben genannten Fragen zu: Wie groß ist die Wahrscheinlichkeit, dass der Solovay–Strassen-Test für eine gegebene zusammengesetzte Zahl n inkorrekt antwortet, dass n prim sei? In Beispiel 7.23 haben wir gesehen, dass der Solovay–Strassen-Test bei Eingabe einer zusammengesetzten Zahl n vielleicht korrekt antwortet, vielleicht aber auch nicht, je nachdem, welche Zufallszahl a er wählt. Es wird nun der Begriff des Solovay–Strassen-Zeugen und der des Solovay–Strassen-Lügners (kurz: SS-Zeuge und SS-Lügner) eingeführt, basierend auf dem Kriterium (7.13). Intuitiv bezeugt ein SS-Zeuge a , dass eine Zahl n zusammengesetzt ist, und ein SS-Lügner a lässt n wie eine Primzahl aussehen, obwohl n in Wirklichkeit zusammengesetzt ist.

Definition 7.24 (SS-Zeuge und SS-Lügner). Sei $n \geq 3$ eine ungerade Zahl.

- Wir sagen, a ist ein SS-Zeuge für n , falls a die Eigenschaft (7.13) nicht erfüllt oder $\text{ggT}(a, n) > 1$ gilt:

$$\left(\frac{a}{n}\right) \left(a^{(n-1)/2} \bmod n\right) \neq 1.$$

- Wir sagen, a ist ein SS-Lügner für n , falls n eine zusammengesetzte Zahl und a kein SS-Zeuge ist, d.h., a erfüllt (7.13):

$$\left(\frac{a}{n}\right) \equiv a^{(n-1)/2} \bmod n.$$

Definiere die Menge $\text{SS-Liars}_n = \{a \mid a \text{ ist ein SS-Lügner für } n\}$.

Der folgende Satz sagt, dass die Fehlerwahrscheinlichkeit des Solovay–Strassen-Tests höchstens $1/2$ ist. Anders gesagt sind höchstens die Hälfte der Elemente in \mathbb{Z}_n^* SS-Lügner für n . Satz 7.25 folgt, mit dem im Beweis von Satz 7.12 gegebenen Argument, aus Lemma 7.26.

Satz 7.25. *Der Solovay–Strassen-Test ist ein effizienter no-biased Monte-Carlo-Algorithmus für Primes mit einer Fehlerwahrscheinlichkeit von höchstens $1/2$.*

Lemma 7.26. *Für jede ungerade zusammengesetzte Zahl $n \geq 3$ ist SS-Liars_n eine echte Untergruppe von \mathbb{Z}_n^* .*

Beweis. Es ist leicht zu sehen, dass $\text{SS-Liars}_n \subseteq \text{F-Liars}_n$ gilt, siehe Übung 7.9(d). Folglich gilt auch $\text{SS-Liars}_n \subseteq \mathbb{Z}_n^*$. Wieder soll Lemma 7.10 angewandt werden, um zu zeigen, dass SS-Liars_n eine Untergruppe von \mathbb{Z}_n^* ist.

Wie in Beispiel 7.23 festgestellt wurde, ist das neutrale Element 1 von \mathbb{Z}_n^* auch in SS-Liars_n , siehe dazu Übung 7.9(b). Um nun zu zeigen, dass SS-Liars_n unter Multiplikation abgeschlossen ist, seien a und b beliebige Elemente von SS-Liars_n , also gilt $(\frac{a}{n}) \equiv a^{(n-1)/2} \pmod{n}$ und $(\frac{b}{n}) \equiv b^{(n-1)/2} \pmod{n}$. Wegen der Multiplikativität des Jacobi-Symbols (siehe Eigenschaft 3 von Behauptung 7.22), folgt

$$\left(\frac{a \cdot b}{n}\right) = \left(\frac{a}{n}\right) \left(\frac{b}{n}\right) \equiv \left(a^{(n-1)/2}\right) \left(b^{(n-1)/2}\right) \equiv (a \cdot b)^{(n-1)/2} \pmod{n}.$$

Somit ist $a \cdot b$ in SS-Liars_n . Nach Lemma 7.10 ist SS-Liars_n eine Untergruppe von \mathbb{Z}_n^* .

Um zu zeigen, dass SS-Liars_n eine *echte* Untergruppe von \mathbb{Z}_n^* ist, beweisen wir, dass \mathbb{Z}_n^* mindestens einen SS-Zeugen enthält. Unterscheide die folgenden beiden Fälle.

Fall 1: *Es gibt eine ungerade Primzahl p , so dass p^2 ein Teiler von n ist.* In diesem Fall konstruieren wir einen Fermat-Zeugen $a \in \mathbb{Z}_n^*$ für n . Wegen $\text{SS-Liars}_n \subseteq \text{F-Liars}_n$ (siehe Übung 7.9(d)) folgt, dass a ebenso ein SS-Zeuge für n ist. Somit ist $\text{SS-Liars}_n \neq \mathbb{Z}_n^*$.

Weil p^2 ein Teiler von n ist, gilt $n = k \cdot p^i$ für ein $i \geq 2$ und eine ungerade Zahl k mit $\text{ggT}(p, k) = 1$. Also ist $\text{ggT}(p^2, k) = 1$. Definiere

$$a = \begin{cases} p+1 & \text{falls } k=1 \\ (p+1) \cdot k \cdot k^{-1} + 1 \cdot p^2 \cdot (p^2)^{-1} \pmod{k \cdot p^2} & \text{falls } k \geq 3, \end{cases}$$

wobei $(p^2)^{-1}$ das inverse Element von p^2 in \mathbb{Z}_k^* und k^{-1} das inverse Element von k in $\mathbb{Z}_{p^2}^*$ ist. Für $k \geq 3$ folgt aus dem Chinesischen Restesatz (siehe Satz 2.46), dass a die eindeutige Lösung des Systems

$$\begin{aligned} a &\equiv p+1 \pmod{p^2} \\ a &\equiv 1 \pmod{k} \end{aligned}$$

ist. Da $a - (p+1) \equiv 0 \pmod{p^2}$ gilt, folgt $\text{ggT}(p, a) = 1$. Da außerdem $a - 1 \equiv 0 \pmod{k}$ gilt, folgt weiter $\text{ggT}(k, a) = 1$. Somit ist $\text{ggT}(n, a) = 1$, also $a \in \mathbb{Z}_n^*$.

Um zu zeigen, dass a ein Fermat-Zeuge für n ist, nehmen wir für einen Widerspruch an, es würde $a^{n-1} \equiv 1 \pmod{n}$ gelten. Weil aber p^2 ein Teiler von n ist, folgt aus dieser Annahme $a^{n-1} \equiv 1 \pmod{p^2}$. Nach dem binomischen Satz gilt:

$$\begin{aligned} a^{n-1} &\equiv (p+1)^{n-1} \equiv 1 + (n-1)p + \sum_{2 \leq i \leq n-1} \binom{n-1}{i} p^i \\ &\equiv 1 + (n-1)p \pmod{p^2}. \end{aligned}$$

Weil $a^{n-1} \equiv 1 \pmod{p^2}$ gilt, teilt p^2 die Zahl $(n-1)p$. Doch dies ist ein Widerspruch, denn p teilt nicht $n-1 = k \cdot p^i - 1$.

Fall 2: Alle Primfaktoren von n sind verschieden. In diesem Fall ist $n = k \cdot p$ für ungerade Zahlen k und p , so dass p prim ist und $\text{ggT}(p, k) = 1$ gilt.

Sei $x \in \mathbb{Z}_p^*$ ein quadratischer Nichtrest modulo p . Somit gilt $\left(\frac{x}{p}\right) = -1$ nach Definition des Legendre-Symbols. Nach dem Chinesischen Restesatz hat das Kongruenzensystem

$$\begin{aligned} a &\equiv x \pmod{p} \\ a &\equiv 1 \pmod{k} \end{aligned}$$

die eindeutige Lösung a . Weil p die Zahl a nicht teilt und weil $\text{ggT}(a, k) = 1$ gilt, ist a in \mathbb{Z}_n^* .

Nun soll gezeigt werden, dass diese Lösung a ein SS-Zeuge für n ist. Nach Behauptung 7.22 gilt:

$$\left(\frac{a}{n}\right) = \left(\frac{a}{k}\right) \left(\frac{a}{p}\right) = \left(\frac{1}{k}\right) \left(\frac{x}{p}\right) = 1 \cdot (-1) = -1. \quad (7.16)$$

Wäre a in SS-Liars_n , so würde aus (7.16)

$$a^{(n-1)/2} \equiv -1 \pmod{n}$$

folgen. Da k ein Teiler von n ist, folgt $a^{(n-1)/2} \equiv -1 \pmod{k}$, im Widerspruch zu $a \equiv 1 \pmod{k}$. Somit ist a ein SS-Zeuge für n .

Der Beweis von Lemma 7.26 ist damit abgeschlossen. \square

7.2.4 Das Primzahl-Problem ist in P

Alle bisher vorgestellten effizienten Primzahltests sind randomisierte Algorithmen. Kann das Primzahl-Problem sogar in *deterministischer* Polynomialzeit gelöst werden? Diese faszinierende Frage ist in der Komplexitätstheorie, trotz intensiver Bemühungen seit vielen Jahrzehnten offen geblieben. Schließlich gelang Agrawal, Kayal und N. Saxena ein entscheidender Durchbruch, und sie lösten dieses Problem in ihrer Arbeit „Primes is in P“ [AKS04, AKS02].

Dieses gefeierte Resultat, dass die Primheit von Zahlen in deterministischer Polynomialzeit getestet werden kann, rief eine Sensation hervor, nicht nur innerhalb der Komplexitätstheorie, sondern auch in der Zahlentheorie, Kryptologie und sogar in der Tagespresse, wie etwa in *The New York Times*. Unmittelbar nachdem die Autoren dieses Resultat auf ihrer Website am 6. August 2002 veröffentlicht hatten, verbreiteten sich die Neuigkeiten schneller als ein Waldbrand.

Warum ist dieses Resultat so wichtig und spektakulär? Genau wie das Graphisomorphie-Problem wurde das Primzahl-Problem lange als einer der seltenen *natürlichen* Kandidaten für solche Probleme gehandelt, die weder in P noch NP-vollständig sind, siehe die Abschnitte 3.6.1 und 5.7 und insbesondere den Abschnitt 6.5. Primes hat seinen Status, ein guter solcher Kandidat zu sein, nun verloren, aber hat im Austausch dafür den Status gewonnen, ein effizient lösbares Problem (im Sinne eines in deterministischer Polynomialzeit lösbarer Problems) zu sein.

Hier wird dieses Resultat ohne Beweis angegeben. Der interessierte Leser sei auf die Originalarbeit verwiesen [AKS04], siehe auch Dietzfelbingers exzellente Präsentation [Die04]. Überraschenderweise gelingt Agrawal, Kayal und Saxenas eleganter Beweis von Satz 7.27 durch Anwendung von zahlentheoretischen Resultaten, die relativ „klassisch“ und grundlegend sind.

Satz 7.27 (Agrawal, Kayal und Saxena). Primes ist in P.

Trotz der Bedeutung von Satz 7.27 scheint es unwahrscheinlich zu sein, dass dieser deterministische Algorithmus für das Primzahl-Problem in absehbarer Zeit die randomisierten Primzahltests wie etwa den Miller–Rabin-Test verdrängen wird. Die Originalarbeit [AKS02] beweist eine Laufzeit von $\mathcal{O}(n^{12})$, wobei n die Länge der (binär codierten) Eingabe ist. Mit einer sorgfältigeren Analyse ist diese Schranke inzwischen bis auf $\mathcal{O}(n^6)$ gedrückt worden. Dennoch sind die gebräuchlichen randomisierten Primzahltests noch deutlich effizienter. Und für praktische Zwecke kann die Fehlerwahrscheinlichkeit dieser randomisierten Algorithmen so klein gemacht werden, dass sie vernachlässigbar ist.

7.3 Das Faktorisierungsproblem

Wie in Abschnitt 7.1 erwähnt ist das RSA-System nur dann sicher, wenn große Zahlen nicht effizient faktorisiert werden können. Traditionell ist das Faktorisierungsproblem ein funktionales Problem: Gegeben eine natürliche Zahl $n \geq 2$, berechne die Primfaktoren von n (wobei diese Faktoren z.B. in aufsteigender Reihenfolge aufgelistet werden).

Definition 7.28 (Faktorisierungsproblem). Definiere das Faktorisierungsproblem als ein funktionales Problem durch

$$\text{factoring}(n) = \langle p_1, p_2, \dots, p_k \rangle,$$

wobei $n = \prod_{i=1}^k p_i$ die Primfaktorisierung von n ist und $p_i \leq p_j$ für $i < j$ gilt. Wie üblich werden alle Zahlen binär über dem Alphabet $\Sigma = \{0, 1\}$ dargestellt.

Beispielsweise bedeutet $\text{factoring}(1453452) = \langle 2, 2, 3, 7, 11, 11, 11, 13 \rangle$, dass $1453452 = 2 \cdot 2 \cdot 3 \cdot 7 \cdot 11 \cdot 11 \cdot 11 \cdot 13$ gilt. Äquivalent dazu kann das Faktorisierungsproblem als ein Entscheidungsproblem formuliert werden, siehe die Übungen 7.10(a) und (b).

Trotz beträchtlicher Anstrengungen in der Vergangenheit konnte bisher kein effizienter Algorithmus für das Faktorisierungsproblem gefunden werden. Es ist daher nicht bekannt, ob factoring in FP liegt. Für das geeignet als Entscheidungsproblem formalisierte Faktorisierungsproblem ist also ebenso wenig bekannt, ob es in P ist. Andererseits sieht man leicht, dass dieses in NP liegt, siehe Übung 7.10(c). Es ist sogar bekannt, dass es in $UP \cap coUP$ ist, siehe Fellows und Koblitz [FK92]. Dieses Resultat kann als ein Indiz dafür gesehen werden, dass das Faktorisierungsproblem wahrscheinlich nicht NP-vollständig ist, genau wie das Graphisomorphie-Problem vermutlich nicht NP-vollständig ist. Faktorisierung ist somit noch ein weiterer Kandidat für ein Problem, das vermutlich weder in P noch NP-vollständig ist. Anders als bei GI jedoch sind für das Faktorisierungsproblem keine Lowness- oder ähnlichen Eigenschaften bekannt; vergleiche die Abschnitte 5.7, 6.5 und 7.6.

Die Annahme, dass Faktorisieren ein schweres Problem ist – worauf die Sicherheit zum Beispiel von RSA zu einem großen Teil ruht – wird lediglich durch unsere fortbestehende Unfähigkeit gestützt, einen effizienten Algorithmus für dieses Problem zu finden. Das ist natürlich ein ziemlich schwaches Indiz für die Härte des Faktorisierens. Wie in Abschnitt 7.2.4 bemerkt wurde das Primzahl-Problem lange Zeit ebenso für ein hartes Problem gehalten, lediglich wegen der fortbestehenden Unfähigkeit, einen effizienten Algorithmus dafür zu finden. Schließlich jedoch stellte sich heraus, dass die Primheit von Zahlen mit einem raffinierten deterministischen Algorithmus in Polynomialzeit getestet werden kann.

Und selbst wenn man einen strengen Beweis für die obige Annahme, dass Faktorisieren schwer ist, fände, würde dieser Beweis nicht die Sicherheit des RSA-Kryptosystems garantieren. Das Brechen von RSA ist höchstens so schwer wie das Faktorisieren von Zahlen, doch ist es nicht bekannt, ob diese beiden Probleme gleich schwer sind. Es könnte gut sein, dass man RSA brechen kann, ohne den RSA-Modul n im öffentlichen Schlüssel (n, e) zu faktorisieren. Potenzielle Angriffe auf das RSA-System und geeignete Gegenmaßnahmen gegen diese Angriffe werden in Abschnitt 7.4 diskutiert. Im folgenden Abschnitt werden einige der bekannten Algorithmen für das Faktorisierungsproblem vorgestellt.

7.3.1 Probdivision

Die erste Beobachtung ist, dass mit dem Probdivisionsalgorithmus aus Abbildung 7.4 ebenso gut faktorisiert werden kann. Genauer gesagt werden, um die Primfaktoren einer gegebenen Zahl $n \geq 2$ zu finden, die folgenden Schritte ausgeführt.

Schritt 1: Berechne alle Primzahlen, die kleiner als oder gleich einer vorbestimmten Schranke b sind, etwa mit dem Sieb des Eratosthenes aus Beispiel 7.5.

Schritt 2: Bestimme für jede Primzahl p in dieser Liste die größte Potenz von p , die n teilt, d.h., den größten Exponenten e_p , so dass p^{e_p} ein Teiler von n ist, und gib die entsprechenden Primfaktoren von n in nicht-fallender Reihenfolge aus.

Eine typische Schranke wäre etwa $b = 1\,000\,000$, und ein typischer RSA-Modul n hätte mindestens 768 Bits. In unserem Spielzeugbeispiel unten begnügen wir uns jedoch mit einer viel kleineren Schranke und einer viel kleineren Zahl, die zu faktorisieren ist.

Beispiel 7.29 (Faktorisieren mit der Probdivision). Angenommen, wir wollen die Zahl $n = 1404$ faktorisieren. In Schritt 1 wählen wir die Schranke $b = 39$ und bestimmen mit dem Sieb des Eratosthenes alle Primzahlen, die 39 nicht überschreiten. Tabelle 7.2 zeigt die entstehende Liste. Schritt 2 liefert nun die folgende Zerlegung in Primzahlpotenzen: $1404 = 2^2 \cdot 3^3 \cdot 13$. Also wird $\langle 2, 2, 3, 3, 3, 13 \rangle$ ausgegeben.

7.3.2 Pollards Algorithmus

Einige Faktorisierungsalgorithmen arbeiten für Zahlen n mit bestimmten Eigenschaften besonders gut. Deshalb müssen solche Zahlen vermieden werden, wenn man den Modul n im öffentlichen Schlüssel beim RSA-Verfahren wählt. Beispielsweise funktioniert die $(p - 1)$ -Methode von Pollard besonders gut für zusammengesetzte Zahlen n , die einen Primfaktor p haben, so dass die Primfaktoren von $p - 1$ klein sind. Abbildung 7.8 präsentiert den Algorithmus, welcher als Eingabe eine zu faktorisierende Zahl n und eine vorbestimmte Schranke B erhält.

```
POLLARD( $n, B$ ) { //  $n \geq 3$  ist eine ungerade Zahl und  $B$  ist eine vorbestimmte Schranke
     $x := 2$ ;
    for ( $i = 2, 3, \dots, B$ ) {  $x := x^i \bmod n$ ; }
     $d := \text{ggT}(n, x - 1)$ ;
    if ( $1 < d < n$ ) {
        return  $d$ ;
        Rufe rekursiv POLLARD( $d, B$ ) und POLLARD( $n/d, B$ ) auf;
    } else return „Erfolgloser Abbruch“ und Neustart mit einer neuen Schranke  $\tilde{B} > B$ ;
}
```

Abb. 7.8. Pollards $(p - 1)$ -Faktorisierungsalgorithmus

Sei p ein Primfaktor von n , und sei B eine obere Schranke aller Primzahlpotenzen, die $p - 1$ teilen. Das heißt, es gilt $q^k \leq B$ für jede Primzahl q und alle $k \geq 1$, so dass q^k ein Teiler von $p - 1$ ist. Dann ist $B!$ ein Vielfaches von $p - 1$. Nach der for-Schleife in Abbildung 7.8 hat die Variable x einen Wert mit $x \equiv 2^{B!} \bmod n$. Da p ein Primfaktor von n ist, gilt ebenso $x \equiv 2^{B!} \bmod p$. Nach dem Kleinen Fermat (siehe Korollar 2.39) ergibt sich

$$2^{p-1} \equiv 1 \bmod p.$$

Und weil $B!$ ein Vielfaches von $p - 1$ ist, folgt

$$x \equiv 2^{B!} \equiv 2^{p-1} \equiv 1 \pmod{p}.$$

Also ist p ein Teiler von $x - 1$. Weil p auch n teilt, ist p ein Teiler von $d = \text{ggT}(n, x - 1)$. Somit ist, außer wenn $x = 1$ gilt, d ein nichttrivialer Faktor von n , und diese Prozedur kann nun rekursiv angewandt werden, um d und n/d zu faktorisieren. Wird kein nichttrivialer Teiler von n gefunden, muss der Algorithmus mit einer neuen Schranke $\tilde{B} > B$ wieder gestartet werden. Vorausgesetzt, die Schranke ist groß genug gewählt worden, findet diese Methode schließlich alle Primfaktoren von n .

Beispiel 7.30 (Pollards $(p - 1)$ -Methode). Sei $n = 56291$ und die Schranke $B = 10$ gewählt. Beginnend mit $x = 2$ zeigt Tabelle 7.7 die Werte von $x = x^i \pmod{n}$ für $2 \leq i \leq 10$, die während der `for`-Schleife in Abbildung 7.8 erhalten werden.

Tabelle 7.7. Berechnung von $x^i \pmod{n}$ in Pollards $(p - 1)$ -Algorithmus

i	2	3	4	5	6	7	8	9	10
x	4	64	2498	29092	49595	2535	5793	25522	1268

Am Ende der `for`-Schleife ergibt sich $x = 1268 \equiv 2^{10!} \pmod{n}$. Dann kann $\text{ggT}(56291, 1267) = 181$ mit dem Euklidischen Algorithmus aus Abbildung 2.1 bestimmt werden. Und tatsächlich ist $p = 181$ ein Primfaktor von $n = 56291$, und der andere Primfaktor von n ist $q = 311$. Da alle Primzahlpotenzen, die $180 = 2^2 \cdot 3^2 \cdot 5$ teilen, kleiner als $B = 10$ sind, teilt $p - 1 = 180$ die Zahl

$$10! = 3628800 = 180 \cdot 20160,$$

wie gewünscht.

Wenn N die Länge der zu faktorisierenden Zahl n und wenn B die gewählte Schranke ist, dann sieht man leicht, dass die Laufzeit einer Ausführung der $(p - 1)$ -Methode von Pollard aus Abbildung 7.8, ohne Berücksichtigung der Rekursion oder des möglichen Neustarts mit einer größeren Schranke, in $\mathcal{O}(B \log BN^2 + N^3)$ ist. Der Beweis dieser Behauptung wird dem Leser als Übung 7.11(c) überlassen. Es folgt, dass wenn B polynomiell in N ist, wenn also $B \in \mathcal{O}(N^k)$ für eine feste Konstante k gilt, dann läuft Pollards $(p - 1)$ -Algorithmus in Polynomialzeit. Leider ist die Wahrscheinlichkeit, dass alle Primzahlpotenzen, die $p - 1$ teilen, unterhalb von B liegen, für solch eine kleine Schranke B ziemlich klein. Deshalb ist es unwahrscheinlich, dass der Algorithmus in diesem Fall erfolgreich ist. Wählen wir jedoch eine große Schranke B , etwa $B \in \mathcal{O}(\sqrt{n})$, dann funktioniert die Methode garantiert, läuft aber nicht schneller als die Probdivision.

7.3.3 Das quadratische Sieb

Einige der derzeit besten Faktorisierungsmethoden beruhen auf der folgenden ganz simplen Idee. Angenommen, man möchte eine ungerade Zahl $n \geq 3$ faktorisieren.

Mit einer geeigneten *Siebmethode* – eine spezielle wird unten noch im Detail beschrieben – bestimmt man Zahlen a und b , für die gilt:

$$a^2 \equiv b^2 \pmod{n} \quad \text{und} \quad a \not\equiv \pm b \pmod{n}. \quad (7.17)$$

Folglich teilt n die Zahl $a^2 - b^2 = (a - b)(a + b)$, aber weder $a - b$ noch $a + b$. Deshalb sind sowohl $\text{ggT}(n, a - b)$ als auch $\text{ggT}(n, a + b)$ nichttriviale Faktoren von n .

Beispiel 7.31 (Idee des quadratischen Siebs). Seien $n = 561$, $a = 322$ und $b = 256$ gegeben. Dann ist $a^2 = 103684 \equiv 460 \pmod{561}$ und $b^2 = 65536 \equiv 460 \pmod{561}$, also ist $a^2 \equiv b^2 \pmod{n}$. Andererseits gilt $a \equiv -239 \pmod{561}$ und $b \equiv -305 \pmod{561}$, also ist $a \not\equiv \pm b \pmod{n}$, und (7.17) ist erfüllt. Folglich teilt 561 die Zahl

$$103684 - 65536 = (322 - 256)(322 + 256) = 66 \cdot 578 = 38148 = 68 \cdot 561,$$

aber 561 teilt weder 66 noch 578. Somit sind sowohl $\text{ggT}(561, 66) = 33$ als auch $\text{ggT}(561, 578) = 17$ nichttriviale Faktoren von 561.

Um noch ein weiteres Beispiel zu geben, betrachte $n = 1269$, $a = 213$ und $b = 210$. Dann ergibt sich $a^2 = 45369 \equiv 954 \pmod{1269}$ und $b^2 = 44100 \equiv 954 \pmod{1269}$, also ist $a^2 \equiv b^2 \pmod{n}$. Da $a \equiv -1056 \pmod{1269}$ und $b \equiv -1059 \pmod{1269}$ gilt, folgt weiter $a \not\equiv \pm b \pmod{n}$, und (7.17) ist erfüllt. Somit teilt 1269 die Zahl

$$45369 - 44100 = (213 - 210)(213 + 210) = 3 \cdot 423 = 1269,$$

aber 1269 teilt weder 3 noch 423. Mit $\text{ggT}(1269, 3) = 3$ und $\text{ggT}(1269, 423) = 423$ sind also zwei nichttriviale Faktoren von 1269 gefunden.

Es gibt verschiedene solche Siebverfahren, die sich jeweils in der spezifischen Art unterscheiden, wie die Zahlen a und b bestimmt werden, für die (7.17) gilt. Ein Beispiel einer sehr erfolgreichen Siebmethode ist das „Zahlkörpersieb“. In diesem Abschnitt jedoch wird das *quadratische Sieb* vorgestellt, welches älter ist, aber in der Praxis immer noch weit verbreitet und auch etwas leichter zu verstehen ist.

Wie wurden die Zahlen a und b , die (7.17) in Beispiel 7.31 erfüllen, ermittelt? Nun, um ehrlich zu sein, die Zahlen $a = 322$ und $b = 256$ im ersten Beispiel (für $n = 561$) wurden einfach durch Probieren bestimmt, was ja bei so kleinen Zahlen durchaus ein probates Mittel ist. Das zweite Beispiel (mit $n = 1269$) wurde etwas sorgfältiger eingefädelt, wie nun gezeigt werden soll.

Beispiel 7.32 (Quadratisches Sieb – Fortsetzung von Beispiel 7.31). Wie in Beispiel 7.31 sei $n = 1269$ die Zahl, die faktorisiert werden soll. Sei $s = \lfloor \sqrt{n} \rfloor = 35$. Definiere eine Funktion $\sigma : \mathbb{Z} \rightarrow \mathbb{Z}$ durch

$$\sigma(x) = (x + s)^2 - n.$$

In unserem Beispiel ist also $\sigma(x) = (x + 35)^2 - 1269$. Tabelle 7.8 zeigt die Werte von $\sigma(x)$ für $-5 \leq x \leq 5$. Insbesondere ergibt sich für $x \in \{3, 4\}$:

$$\sigma(3) = 38^2 - 1269 = 175 = 5^2 \cdot 7 \quad (7.18)$$

$$\sigma(4) = 39^2 - 1269 = 252 = 2^2 \cdot 3^2 \cdot 7. \quad (7.19)$$

Tabelle 7.8. Berechnung von $\sigma(x)$ beim quadratischen Sieb

x	-5	-4	-3	-2	-1	0	1	2	3	4	5
$\sigma(x)$	-369	-308	-245	-180	-113	-44	27	100	175	252	331

Die Gleichungen (7.18) und (7.19) implizieren:

$$38^2 \equiv 5^2 \cdot 7 \pmod{1269} \quad (7.20)$$

$$39^2 \equiv 2^2 \cdot 3^2 \cdot 7 \pmod{1269}. \quad (7.21)$$

Multipliziert man nun (7.20) und (7.21), so erhält man

$$(38 \cdot 39)^2 \equiv (2 \cdot 3 \cdot 5 \cdot 7)^2 \pmod{1269}. \quad (7.22)$$

Auf beiden Seiten von (7.22) ergibt sich somit eine Quadratzahl, wobei das Quadrat auf der rechten Seite aus kleinen Primzahlen besteht. Setzt man

$$a = 38 \cdot 39 \pmod{1269} = 213 \quad \text{und} \quad b = 2 \cdot 3 \cdot 5 \cdot 7 \pmod{1269} = 210,$$

so erhält man nun die Werte von a und b aus Beispiel 7.31.

In (7.18) und (7.19) und in Tabelle 7.8 aus Beispiel 7.32 wurde $\sigma(x)$ für bestimmte Argumente x berechnet, so dass $\sigma(x)$ nur kleine Primfaktoren hat und

$$(x + s)^2 \equiv \sigma(x) \pmod{n} \quad (7.23)$$

gilt. Dann wurden geeignete Kongruenzen der Form (7.23) ausgewählt, so dass ihr Produkt auf beiden Seiten eine Quadratzahl ergibt. Die linke Seite von (7.23) ist selbst ein Quadrat, so dass das Produkt von linken Seiten stets wieder ein Quadrat ist. Das Problem besteht darin, Kongruenzen der Form (7.23) zu finden, so dass die rechten Seiten bei der Multiplikation ebenfalls ein Quadrat ergeben. Offenbar ist dies der Fall, wenn die Summe der Exponenten von -1 und den Primfaktoren von $\sigma(x)$ gerade ist. Um also geeignete Kongruenzen der Form (7.23) auszuwählen, muss einfach ein lineares System von Gleichungen über dem Körper \mathbb{Z}_2 gelöst werden. Dies wird im folgenden Beispiel gezeigt, das Beispiel 7.32 fortsetzt.

Beispiel 7.33 (Quadratisches Sieb – Fortsetzung von Beispiel 7.32). Wir zeigen, wie man geeignete Kongruenzen der Form (7.23) so auswählt, dass ihr Produkt auf beiden Seiten ein Quadrat ist.

Zur besseren Illustration nehmen wir an, dass uns nicht nur die Gleichungen (7.18) und (7.19) für $\sigma(x)$ mit $x \in \{3, 4\}$ gegeben sind, welche die Kongruenzen (7.20) und (7.21) implizieren, sondern es sind noch zwei weitere Gleichungen für $\sigma(x)$ gegeben, nämlich diejenigen mit $x \in \{-4, -2\}$. Also stehen wir dem Problem gegenüber, die geeigneten unter den folgenden vier Kongruenzen der Form (7.23) auszuwählen:

$$\begin{aligned} 31^2 &\equiv -1 \cdot 2^2 \cdot 7 \cdot 11 \pmod{1269} \\ 33^2 &\equiv -1 \cdot 2^2 \cdot 3^2 \cdot 5 \pmod{1269} \\ 38^2 &\equiv 5^2 \cdot 7 \pmod{1269} \\ 39^2 &\equiv 2^2 \cdot 3^2 \cdot 7 \pmod{1269}. \end{aligned}$$

Um diejenigen Kongruenzen auszuwählen, deren Produkt auf der rechten Seite ein Quadrat ergibt, muss man Koeffizienten $\alpha_i \in \{0, 1\}$ finden, $1 \leq i \leq 4$, so dass gilt:

$$\begin{aligned} &(-1 \cdot 2^2 \cdot 7 \cdot 11)^{\alpha_1} (-1 \cdot 2^2 \cdot 3^2 \cdot 5)^{\alpha_2} (5^2 \cdot 7)^{\alpha_3} (2^2 \cdot 3^2 \cdot 7)^{\alpha_4} \\ &= (-1)^{\alpha_1 + \alpha_2} \cdot 2^{2\alpha_1 + 2\alpha_2 + 2\alpha_4} \cdot 3^{2\alpha_2 + 2\alpha_4} \cdot 5^{\alpha_2 + 2\alpha_3} \cdot 7^{\alpha_1 + \alpha_3 + \alpha_4} \cdot 11^{\alpha_1}. \end{aligned} \quad (7.24)$$

Der Koeffizient α_i entspricht dabei dem Exponenten der i -ten Kongruenz. Die Zahl in (7.24) ist genau dann ein Quadrat, wenn die Exponenten von -1 und aller Primfaktoren jeweils gerade sind. Das heißt also, man muss das folgende System von Kongruenzen lösen:

$$\begin{aligned} \alpha_1 + \alpha_2 &\equiv 0 \pmod{2} \\ 2\alpha_1 + 2\alpha_2 + 2\alpha_4 &\equiv 0 \pmod{2} \\ 2\alpha_2 + 2\alpha_4 &\equiv 0 \pmod{2} \\ \alpha_2 + 2\alpha_3 &\equiv 0 \pmod{2} \\ \alpha_1 + \alpha_3 + \alpha_4 &\equiv 0 \pmod{2} \\ \alpha_1 &\equiv 0 \pmod{2}. \end{aligned}$$

Reduziert man modulo 2, so können zwei dieser sechs Kongruenzen eliminiert werden, und man erhält das folgende vereinfachte System von vier Kongruenzen:

$$\begin{aligned} \alpha_1 + \alpha_2 &\equiv 0 \pmod{2} \\ \alpha_2 &\equiv 0 \pmod{2} \\ \alpha_1 + \alpha_3 + \alpha_4 &\equiv 0 \pmod{2} \\ \alpha_1 &\equiv 0 \pmod{2}, \end{aligned}$$

welches die nichttriviale (d.h., vom Nullvektor verschiedene) Lösung $\alpha_1 = \alpha_2 = 0$ und $\alpha_3 = \alpha_4 = 1$ hat. Demgemäß werden genau die Gleichungen (7.18) und (7.19) für $\sigma(x)$ mit $x \in \{3, 4\}$ ausgewählt, gerade wie im Beispiel 7.32.

Wie bestimmt nun das quadratische Sieb Kongruenzen der Form (7.23)? In praktischen Anwendungen ist die zu faktorisierende Zahl n natürlich viel größer als das $n = 1269$ in unserem Beispiel oben; sie wird nicht vier Ziffern, sondern wahrscheinlich mehr als einhundert Ziffern haben. Hier soll nur grob skizziert werden, wie das oben beschriebene Prinzip allgemein funktioniert, wobei zahlreiche Tricks und mathematische Verfeinerungen weggelassen werden, mit denen die Berechnung in der Praxis beschleunigt werden kann.

Schritt 1: Sei n die zu faktorisierende Zahl. Zunächst wird eine *Faktorbasis* \mathfrak{B} gewählt, die aus -1 sowie aus Primzahlen besteht, die klein genug sind, um

als Faktoren in $\sigma(x)$ zugelassen werden zu können. Genauer gesagt wird dafür eine „kleine“ Schranke B festgelegt und dann die Faktorbasis definiert als

$$\mathfrak{B} = \{-1\} \cup \{p \mid p \text{ ist eine Primzahl, die kleiner als oder gleich } B \text{ ist}\}.$$

Schritt 2: Sei $s = \lfloor \sqrt{n} \rfloor$. Ein Funktionswert $\sigma(x) = (x+s)^2 - n$ heißt \mathfrak{B} -glatt, falls alle seine Primfaktoren in \mathfrak{B} enthalten sind.

Schritt 3: Bestimme $\|\mathfrak{B}\|$ Zahlen x , so dass $\sigma(x)$ \mathfrak{B} -glatt ist.¹

Schritt 4: Löse das entsprechende System von $\|\mathfrak{B}\|$ Kongruenzen, um geeignete Kongruenzen der Form (7.23) so auszuwählen, dass ihr Produkt ein Quadrat auf beiden Seiten ergibt, wie in Beispiel 7.33 gezeigt. Dies kann entweder mit Standardmethoden, wie etwa dem Eliminierungsverfahren von Gauß, oder mit speziellen, effizienteren Methoden erledigt werden.

Schritt 5: Bestimme die Werte von a und b , für die (7.17) gilt, durch Multiplizieren der ausgewählten Kongruenzen, wie in Beispiel 7.32 gezeigt.

Schritt 6: Bestimme, wie in Beispiel 7.31 gezeigt, die nichttrivialen Faktoren $d_1 = \text{ggT}(n, a-b)$ und $d_2 = \text{ggT}(n, a+b)$ von n . (Es gilt $n = d_1 \cdot d_2$.)

Schritt 7: Wende diese Prozedur rekursiv auf d_1 und d_2 und so weiter an, bis schließlich die Faktorisierung von n gefunden ist.

Es müssen nun noch die Details von Schritt 3 genauer spezifiziert werden: Wie werden die ganzen Zahlen $x_1, x_2, \dots, x_{\|\mathfrak{B}\|}$ so ermittelt, dass jedes $\sigma(x_i)$ \mathfrak{B} -glatt ist? Dieser Schritt ist die eigentliche Siebprozedur und wird zur Illustration wieder an einem Beispiel erklärt.

Beispiel 7.34 (Quadratisches Sieb – Fortsetzung von Beispiel 7.33). Wir setzen hier die Beispiele 7.31, 7.32 und 7.33 fort. Zur Erinnerung: $n = 1269$ ist die zu faktorisierende Zahl, $s = \lfloor \sqrt{n} \rfloor = 35$ und $\sigma(x) = (x+35)^2 - 1269$. Wähle die Faktorbasis $\mathfrak{B} = \{-1, 2, 3, 5, 7, 11\}$.

Für diese kleinen Zahlen könnte man den naiven „Brute-Force“-Zugang vorziehen, bei dem $\sigma(x)$ für jedes $x \in \{0, \pm 1, \pm 2, \dots\}$ ausgerechnet und dann durch Probbedivision überprüft wird, ob $\sigma(x)$ \mathfrak{B} -glatt ist. Dieser Zugang würde in praktischen Anwendungen jedoch zu lange dauern. Hat n etwa 120 Ziffern, so besitzt die Faktorbasis ungefähr 245000 Elemente, und alle diese Elemente müssten getestet werden, um auch nur einen Wert $\sigma(x)$ zu überprüfen. Viel schlauer ist es, stattdessen ein Sieb zu verwenden.

Fixiere ein Siebintervall, $S = \{-D, -D+1, \dots, -1, 0, 1, \dots, D-1, D\}$, wobei D eine vorbestimmte Konstante ist. Berechne $\sigma(x)$ für jedes $x \in S$. Teile nun, für jede Primzahl p in \mathfrak{B} und für jedes x in S , den Wert $\sigma(x)$ durch das größtmögliche Vielfache von p . Der Funktionswert $\sigma(x)$ ist genau dann \mathfrak{B} -glatt, wenn diese Prozedur schließlich den Quotienten ± 1 lässt.

¹ In Beispiel 7.33 sind die vier Gleichungen, die den ganzen Zahlen $x \in \{-4, -2, 3, 4\}$ entsprechen, eigentlich zu wenige für eine vernünftige Faktorbasis wie z.B. $\mathfrak{B} = \{-1, 2, 3, 5, 7, 11\}$. Es sollte aus diesem Beispiel aber klar sein, wie die Methode im Prinzip funktioniert.

Für eine gegebene Primzahl p in \mathfrak{B} kann man folgendermaßen diejenigen x in S bestimmen, für die p ein Teiler von $\sigma(x)$ ist. Zunächst werden alle $x \in \mathbb{Z}_p$ ermittelt, für die p ein Teiler von $\sigma(x)$ ist, d.h., für die $\sigma(x) \equiv 0 \pmod{p}$ gilt. Da $\sigma(x)$ ein Polynom vom Grad 2 ist, gibt es höchstens zwei solche Zahlen x in \mathbb{Z}_p . Läuft man nun ausgehend von diesen x in Schritten der Länge p nach links und nach rechts durch das Siebintervall, so entdeckt man alle $x \in S$, für die p ein Teiler von $\sigma(x)$ ist. Diese Prozedur nennt man ein *Sieb mit p* . Der Vorteil ist, dass ein Sieb mit p nur solche Werte $\sigma(x)$ betrachtet, die Vielfache von p sind, wodurch erfolglose Probefaktorisierungen vermieden werden.

Tabelle 7.9. Bestimmung der \mathfrak{B} -glatten Werte $\sigma(x)$ durch Siebe mit p

x	-5	-4	-3	-2	-1	0	1	2	3	4	5
$\sigma(x)$	-369	-308	-245	-180	-113	-44	27	100	175	252	331
Sieb mit 2		-77		-45		-11		25		63	
Sieb mit 3	-41			-5			1			7	
Sieb mit 5			-49	-1				1	7		
Sieb mit 7		-11	-1						1	1	
Sieb mit 11		-1				-1					

Sei speziell $S = \{-5, -4, \dots, -1, 0, 1, \dots, 4, 5\}$ unser Siebintervall. Tabelle 7.9 ergänzt Tabelle 7.8 aus Beispiel 7.32 und zeigt das Sieb mit p für jede Primzahl p in \mathfrak{B} . Somit ist $\sigma(x)$ \mathfrak{B} -glatt für jedes $x \in \{-4, -3, -2, 0, 1, 2, 3, 4\}$, nicht aber für $x \in \{-5, -1, 5\}$.

7.3.4 Andere Faktorisierungsmethoden

In diesem Abschnitt werden einige weitere Faktorisierungsmethoden und ihre Laufzeiten erwähnt, ohne dabei allzu sehr ins Detail zu gehen. In vielen Fällen hängen die Laufzeiten von Faktorisierungsalgorithmen von bestimmten Annahmen und Parametern ab, wie etwa von der Wahl der Größe der Faktorbasis beim quadratischen Sieb, und erlauben daher nur eine heuristische Analyse.

Offensichtlich gibt es einen *Trade-off* zwischen der Erfolgswahrscheinlichkeit einer Faktorisierungsmethode und ihrer Laufzeit. Wählt man zum Beispiel eine große Faktorbasis \mathfrak{B} beim quadratischen Sieb, so erhöht sich die Wahrscheinlichkeit dafür, dass man \mathfrak{B} -glatte Werte $\sigma(x)$ findet, und somit erhöht sich auch die Wahrscheinlichkeit dafür, n erfolgreich zu faktorisieren. Andererseits, je größer die Faktorbasis ist, um so mehr Kongruenzen der Form (7.23) erhält man, was dazu führt, dass die Lösung des entsprechenden Kongruenzensystems zeitaufwändiger wird.

Mit bestimmten zahlentheoretischen Resultaten und unter Ausnutzung gewisser nützlicher Annahmen kann man zeigen, dass das quadratische Sieb bei Eingabe n in der Zeit

$$\mathcal{O}\left(e^{(1+o(1))\sqrt{\ln n \ln \ln n}}\right)$$

läuft, wobei $\ln n$ den natürlichen Logarithmus von n bezeichnet, also den Logarithmus zur Basis $e = 2.71828\cdots$. Insbesondere benutzt man dafür das Primzahl-Theorem (siehe Satz 7.3), um die Größe der Faktorbasis durch $\|\mathfrak{B}\| = 1 + \pi(B) \approx B/\ln B$ abzuschätzen, wobei B die vorbestimmte Schranke und $\pi(B)$ die Anzahl der Primzahlen $p \leq B$ bezeichnet.

Tabelle 7.10 fasst die besten derzeit bekannten Laufzeiten ausgewählter Faktorisierungsalgorithmen zusammen, wobei n die zu faktorisierende Eingabezahl ist. Hier bezeichnet p den kleinsten Primfaktor von n , und die anderen Bezeichnungen wurden im vorangegangenen Absatz erklärt.

Tabelle 7.10. Laufzeiten ausgewählter Faktorisierungsalgorithmen

Algorithmus	Laufzeit
Pollards $(p - 1)$ -Algorithmus	$\mathcal{O}(B \log B (\log n)^2 + (\log n)^3)$
Quadratisches Sieb	$\mathcal{O}(e^{(1+o(1))\sqrt{\ln n \ln \ln n}})$
Zahlkörpersieb	$\mathcal{O}(e^{(1.92+o(1))\sqrt[3]{\ln n} \sqrt[3]{(\ln \ln n)^2}})$
Elliptische-Kurven-Methode	$\mathcal{O}(e^{(1+o(1))\sqrt{2 \ln p \ln \ln p}})$

Pollards $(p - 1)$ -Algorithmus wurde in Abschnitt 7.3.2 präsentiert und analysiert, und Abschnitt 7.3.3 erklärte, wie das quadratische Sieb funktioniert. Dixons *Methode der zufälligen Quadrate* arbeitet nach demselben Prinzip wie das quadratische Sieb, siehe z.B. Stinson [Sti05]. Ein Unterschied ist, dass man eine Variable x zufällig unter Gleichverteilung aus \mathbb{Z}_n wählt und hofft, dass die Quadrate $x^2 \bmod n$ vollständig über der Faktorbasis faktorisieren. Man kann die Erfolgswahrscheinlichkeit auch durch geeignete Wahl von x erhöhen. Beliebte Werte sind $x = \lfloor \sqrt{kn} \rfloor$ oder auch $x = \lceil \sqrt{kn} \rceil + \ell$, wobei $k \in \mathbb{N} - \{0\}$ und $\ell \in \mathbb{N}$, da sich dann in der Arithmetik modulo n Quadrate nahe n ergeben, so dass $-x^2 \bmod n$ klein ist. Dabei sind wieder -1 sowie alle Primzahlen bis zu einer vorgegebenen Schranke in der Faktorbasis \mathfrak{B} , und mit dem Produkt ausgewählter Quadrate versucht man eine Kongruenz der Form (7.17) zu finden.

Auch das *Zahlkörpersieb* verallgemeinert das quadratische Sieb und faktorisiert n mittels Kongruenzen der Form (7.17). Ein Unterschied hier ist, dass alle Berechnungen in Ringen über algebraischen Zahlen ausgeführt werden, siehe A. Lenstra und H. Lenstra, Jr. [LL93]. Asymptotisch hat das Zahlkörpersieb die beste Laufzeit unter allen derzeit bekannten Faktorisierungsmethoden, siehe Tabelle 7.10.

Die *Elliptische-Kurven-Methode*, ein von H. Lenstra in den 1980ern entwickelter randomisierter Algorithmus verallgemeinert den $(p - 1)$ -Algorithmus von Pollard. Hier finden die Berechnungen nicht im Ring \mathbb{Z}_p statt, sondern in Gruppen, die auf elliptischen Kurven modulo p definiert sind. Die Laufzeit dieser Methode hängt von der Größe des kleinsten Primfaktors von n ab. Im schlimmsten Fall haben alle Primfaktoren von n ungefähr dieselbe Größe. Hat n insbesondere zwei Primfaktoren wie beim RSA-Kryptosystem, so ist ihre Größe etwa \sqrt{n} , also sind die Laufzeiten des quadratischen Siebs und der Elliptische-Kurven-Methode asymptotisch im Wesent-

lichen gleich. Haben die Primfaktoren von n jedoch sehr unterschiedliche Größen, so ist die Elliptische-Kurven-Methode effizienter. Beispielsweise ist $2^{2^{11}} - 1$ eine sehr große Zahl, die von Brent mit der Elliptischen-Kurven-Methode faktorisiert werden konnte.

Zahlen der Form $F_m = 2^{2^m} + 1$ heißen *Fermat-Zahlen*. Fermat glaubte, dass alle F_m Primzahlen seien, was für $F_0 = 3, F_1 = 5, F_2 = 17, F_3 = 257$ und $F_4 = 65537$ auch tatsächlich stimmt. Jedoch ist $F_5 = 641 \cdot 6700417$ eine zusammengesetzte Zahl, wie Euler 1732 bewies. F_6 konnte 1880 von Landry und Le Lasseur faktorisiert werden. Brillhart und Morrison faktorisierten F_7 im Jahr 1970, beinahe ein Jahrhundert später. 1980 fanden Brent und Pollard die Faktorisierung von F_8 ; A. Lenstra, H. Lenstra, Manasse und Pollard faktorisierten F_9 im Jahr 1990; und 1995 faktorierte Brent

$$F_{10} = 45592577 \cdot 6487031809 \cdot 4659775785220018543264560743076778192897 \cdot P_{52},$$

wobei P_{52} die 252-te Primzahl bezeichnet; siehe auch Übung 7.12(c).

Bei der Analyse solcher Algorithmen ist zu beachten, dass es hier darauf ankommt, wie die Eingabe repräsentiert wird. Ist die Eingabe $F_m = 2^{2^m} + 1$ binär gegeben, so hat sie die Form $\text{bin}(F_m) = 10^{2^m-1}1$. Schon vor dem Algorithmus von Agrawal, Kayal und N. Saxena [AKS02] war es bekannt, dass das Problem, ob F_m prim ist oder nicht, in einer Zeit gelöst werden kann, die polynomiell in der Eingabellänge $2^m + 1$ ist. Daher ist es üblich, die Zeit für Algorithmen zur Entscheidung der Primheit von F_m in der Länge der Binärdarstellung von m zu messen.

Tabelle 7.11. Faktorisierung einiger Zahlen RSA- d

Herausforderung	Faktorisierungsmethode	Jahr
RSA-129	Quadratisches Sieb	1994
RSA-130	Zahlkörpersieb	1996
RSA-140	Zahlkörpersieb	1999
RSA-155	Zahlkörpersieb	1999

Eine den Fermat-Zahlen ähnliche Herausforderung an Faktorisierungsalgorithmen stellen die Zahlen RSA- d dar. Jede solche Zahl ist ein RSA-Modul n mit d Ziffern, der zwei Primfaktoren ungefähr gleicher Größe hat. Die Zahlen RSA-100, RSA-110, ..., RSA-500 wurden im Internet veröffentlicht, und Faktorisierungsspezialisten überall auf der Welt versuchten, die jeweiligen Primfaktoren zu ermitteln. Oft werden die nötigen Berechnungen auf Hunderte von Rechnern verteilt, um diese Herausforderung gemeinsam schneller zu meistern. Diese Methode, die manchmal als „Faktorisierung per E-Mail“ bezeichnet wird, hat sich oft als erfolgreich erwiesen. Einige dieser Meilensteine in der Geschichte des Faktorisierens sind in Tabelle 7.11 aufgelistet.

7.4 Sicherheit von RSA: Angriffe und Gegenmaßnahmen

Das RSA-Protokoll und das digitale RSA-Signatur-Schema wurden in Abschnitt 7.1 vorgestellt, siehe die Abbildungen 7.1 und 7.3. Wie dort erwähnt hängt die Sicherheit des RSA-Kryptosystems stark von der Annahme ab, dass das Faktorisieren großer Zahlen eine praktisch nicht machbare Aufgabe ist. Abschnitt 7.3 zeigte, dass die besten bekannten Faktorisierungsalgorithmen in superpolynomialer Zeit laufen und somit nicht effizient sind, auch wenn einige subexponentielle Algorithmen sind. Allerdings ist nicht bekannt, ob das Faktorisierungsproblem und das Problem, das RSA-System zu brechen, gleich schwer sind. Mehr Details dazu findet man in Abschnitt 7.6.

In diesem Abschnitt werden einige mögliche Angriffe auf das RSA-System aufgeführt. Unter diesen sind die Faktorisierungsangriffe am offensichtlichsten. Um solche direkten Angriffe zu verhindern, ist eine gewisse Sorgfalt bei der Wahl der Primzahlen p und q , des Moduls n , des Exponenten e und des privaten Schlüssels d aus Abbildung 7.1 nötig.

Literatur zur Sicherheit von RSA gibt es in Hülle und Fülle, und die Liste der hier vorgestellten Angriffe ist weit davon entfernt, vollständig zu sein. Für jeden in der Literatur bisher vorgeschlagenen Angriff auf RSA sind bestimmte praktische Gegenmaßnahmen bekannt, Daumenregeln, die den Erfolg solcher Angriffe unmöglich oder jedenfalls die Wahrscheinlichkeit eines Erfolgs vernachlässigbar klein machen.

Faktorisierungsangriffe

Der Angreifer Erich möchte aus seiner Kenntnis des öffentlichen Schlüssels (n, e) den privaten Schlüssel d durch Faktorisierung von n ermitteln, indem er also die Primfaktoren p und q von $n = pq$ bestimmt. Kennt er p und q , so kann er $\varphi(n) = (p - 1)(q - 1)$ berechnen, und genau wie Bob kann er dann das Inverse d von e modulo $\varphi(n)$ mit dem erweiterten Euklidischen Algorithmus bestimmen; siehe Abbildung 2.1.

Es gibt verschiedene Möglichkeiten, wie ein Faktorisierungsangriff auf RSA ausgeführt werden kann. Wir unterscheiden die folgenden Typen von Faktorisierungsangriffen.

Brute-Force-Angriff: Mit einer erschöpfenden Suche versucht Erich, n einfach durch Probefaktorisierung zu faktorisieren, siehe Abschnitt 7.3.1. Durch die Wahl eines hinreichend großen n kann dieser Angriff verhindert werden. Derzeit wird die Verwendung eines Moduls n mit mindestens 1024 Bits empfohlen; es gibt Schätzungen, nach denen diese Bitlänge bis etwa zum Jahr 2018 als sicher gelten kann. Insbesondere kann die Größe von 512 Bits, die früher verwendet wurde, heute keinen angemessenen Schutz mehr bieten. Beispielsweise hat die Zahl RSA-155, die 1999 erfolgreich faktorisiert wurde (siehe Tabelle 7.11), ungefähr dieselbe Größe wie eine Zahl mit 512 Bits. Andererseits ist bei einer Empfehlung wie der Bitlänge 1024 natürlich Vorsicht angebracht, da der Fortschritt in der Algorithmik und bei der Hardware-Entwicklung schwer vorherzusagen ist. Wenn sich irgendwann herausstellen sollte,

dass das Faktorisierungsproblem effizient lösbar ist, dann sind alle Kryptosysteme, deren Sicherheit auf der Härte des Faktorisierens beruht, einschließlich RSA, nicht mehr sicher.

Natürlich wächst die Zeitkomplexität des modularen Potenzierens rasant mit der Größe des Moduls, und deshalb gibt es einen *Trade-off* zwischen dem Erhöhen der Sicherheit und dem Verringern der Effizienz von RSA. Auch ist die Tatsache allgemein anerkannt, dass die Zahlen n , deren Primfaktoren p und q etwa die gleiche Größe haben, am schwersten zu faktorisieren sind.

Spezial-Faktorisierungsmethoden: Ein Beispiel für diese Angriffsart ist Pollards $(p - 1)$ -Methode. Wie in Abschnitt 7.3.2 gezeigt wurde, versucht diese Faktorisierungsmethode eine Schwäche der Primfaktoren von n auszunutzen: Ist $n = pq$ und hat $p - 1$ nur kleine Primfaktoren, dann kann Pollards $(p - 1)$ -Algorithmus den RSA-Modul n effizient faktorisieren. Um dieser potenziellen Bedrohung effektiv zu begegnen, wurden so genannte „starke“ Primzahlen eingeführt, welche bestimmte Bedingungen erfüllen müssen. Beispielsweise sollte, damit p als ein *starker* Primfaktor von n gilt, die Zahl $p - 1$ einen großen Faktor r haben, um zu verhindern, dass die $(p - 1)$ -Methode effizient funktioniert. Ähnlich sollte auch $r - 1$ wieder einen großen Faktor haben und so weiter.

Ein weiteres Beispiel einer Spezial-Faktorisierungsmethode ist Lenstras *Elliptische-Kurven-Methode*. Wie schon in Abschnitt 7.3.4 bemerkt wurde, verallgemeinert diese Methode Pollards $(p - 1)$ -Methode. Sie funktioniert um so besser beim Brechen von RSA, je kleiner der kleinste Primfaktor von n ist.

Spezial-Faktorisierungsmethoden, die bestimmte Eigenschaften der Primfaktoren p und q von n ausnutzen, können wirkungsvoller und erfolgreicher als die unten beschriebenen Universal-Faktorisierungsmethoden sein. Aber weil sie von speziellen Eigenschaften der gewählten Parameter abhängen, können diese Angriffstypen leichter vermieden werden.

Universal-Faktorisierungsmethoden: Beispiele sind das *quadratische Sieb* und das *Zahlkörpersieb*, die in den Abschnitten 7.3.3 und 7.3.4 diskutiert wurden. Egal, welche Form die Primfaktoren von n haben, diese Faktorisierungsalgorithmen besitzen eine gewisse Erfolgswahrscheinlichkeit. Daher ist der Gebrauch von sehr großen Primzahlen die wirkungsvollste Gegenmaßnahme gegen solche Angriffe. Diese Gegenmaßnahme gewährt gleichzeitig Schutz vor allen Arten von Spezial-Faktorisierungsmethoden. Kurz gesagt, auf die Größe kommt es eben doch an, und große Primzahlen sind wichtiger als starke Primzahlen.

Faktorisieren mit der Euler-Funktion

Angenommen, Erich kann $\varphi(n)$ bestimmen, wobei φ die Euler-Funktion ist. Da er nun sowohl n als auch $\varphi(n)$ kennt, könnte er die (ihm unbekannten) Primfaktoren von $n = pq$ ermitteln, indem er das folgende Gleichungssystem nach den Unbekannten p und q löst:

$$\begin{aligned} n &= p \cdot q \\ \varphi(n) &= (p - 1)(q - 1). \end{aligned}$$

Setzt man $q = n/p$ in die zweite Gleichung ein, so erhält man eine quadratische Gleichung in p :

$$p^2 - (n - \varphi(n) + 1)p + n = 0. \quad (7.25)$$

Nach dem Vietaschen Wurzelsatz sind p und q genau dann die Lösungen einer quadratischen Gleichung der Form $p^2 + ap + b = 0$, wenn $p + q = -a$ und $pq = b$ gilt. Da die Primfaktoren p und q von n sowohl $pq = n$ als auch

$$p + q = pq - pq + p + q - 1 + 1 = pq - (p - 1)(q - 1) + 1 = n - \varphi(n) + 1$$

erfüllen, hat (7.25) die Wurzeln p und q . Es folgt, dass ein Kryptoanalytiker, der $\varphi(n)$ kennt, RSA leicht brechen kann. Anders gesagt ist das Berechnen von $\varphi(n)$ mindestens so schwer wie n zu faktorisieren, und umgekehrt. Somit sind $\varphi(n)$ zu berechnen und n zu faktorisieren gleich schwere Aufgaben.

Beispiel 7.35. Sei $n = 60477719$. Angenommen, Erich ist es gelungen, den Wert $\varphi(n) = 60462000$ zu ermitteln. Nach (7.25) kann er dann die Primfaktoren von n einfach dadurch bestimmen, dass er die quadratische Gleichung

$$p^2 - 15720p + 60477719 = 0$$

folgendermaßen löst:

$$\begin{aligned} p &= \frac{15720}{2} + \sqrt{\left(\frac{15720}{2}\right)^2 - 60477719} = 9001 \quad \text{und} \\ q &= \frac{15720}{2} - \sqrt{\left(\frac{15720}{2}\right)^2 - 60477719} = 6719. \end{aligned}$$

Superverschlüsselung

Simmons und Norris schlugen bereits 1977 einen Angriff auf RSA vor, kurz nach der Erfindung von RSA. Ihr Angriff, der Superverschlüsselung genannt wird, beruht auf der Beobachtung, dass eine hinreichende Anzahl von Verschlüsselungen, die ihre Bahn durch \mathbb{Z}_n zieht, womöglich am Ende wieder bei der ursprünglichen Nachricht m landet. Dieser Angriff stellt eine Bedrohung für RSA dar, sofern die Anzahl der für die Wiederentdeckung von m benötigten Verschlüsselungen klein ist. Glücklicherweise ist die Superverschlüsselung jedoch kein praktibler Angriff, wenn die Primzahlen p und q groß sind und zufällig gewählt werden.

Beispiel 7.36 (Superverschlüsselung). Sei $n = 5 \cdot 7 = 35$, also $\varphi(n) = 4 \cdot 6 = 24$. Wähle den Verschlüsselungsexponenten $e = 5$; es gilt $\text{ggT}(24, 5) = 1$. Das Verschlüsseln der Nachricht $m = 11$ liefert

$$11^5 \bmod 35 = 16.$$

Verschlüsselt man nun die Nachricht $m' = 16$, so landet man wieder bei der ursprünglichen Nachricht:

$$16^5 \bmod 35 = 11,$$

was keine Überraschung ist, denn der Entschlüsselungsschlüssel d ist in diesem Fall zufällig gleich e : $5^2 \bmod 24 = 1$, also $d = 5 = e$. Tatsächlich ist jede Zahl e mit $\text{ggT}(24, e) = 1$ gleich ihrem Inversen modulo 24, siehe Übung 7.13.

Wählen wir jetzt also $n = 11 \cdot 13 = 143$. Dann ist $\varphi(n) = 10 \cdot 12 = 120$. Der Verschlüsselungsexponent $e = 7$ hat das Inverse $d = 103$ modulo 120, also gilt diesmal $e \neq d$. Dennoch liefert das Verschlüsseln der Nachricht $m = 11$ nun

$$11^7 \bmod 143 = 132 \quad \text{und} \quad 132^7 \bmod 143 = 11.$$

Ohne den privaten Schlüssel $d = 103$ zu kennen, kann ein Kryptoanalytiker also die Originalnachricht einfach durch eine Doppelverschlüsselung erhalten.

Angriff auf kleine Nachrichten (Small-Message Attack)

Wenn sowohl die zu verschlüsselnde Nachricht m als auch der Verschlüsselungsexponent e relativ zum Modul n klein sind, dann ist die RSA-Verschlüsselung nicht sicher. Ist nämlich der Schlüsseltext $c = m^e$ kleiner als n , so kann m aus c durch ganz gewöhnliches Wurzelziehen erhalten werden, ohne dass die Rechnung durch die Arithmetik modulo n schwierig würde. Um das zu verhindern, sollten stets der öffentliche Exponent oder die zu verschlüsselnde Nachricht groß sein. Der letztere Vorschlag ist dabei der nützlichere, denn ein kleiner Verschlüsselungsexponent erhält oft den Vorzug, um die Verschlüsselung zu beschleunigen und um Wieners Angriff zu erschweren.

Wieners Angriff

Wiener schlug einen Angriff auf das RSA-System vor, der eine Kettenbruch-Approximation und den öffentlichen Schlüssel (n, e) benutzt, um den privaten Schlüssel d zu ermitteln. Dieser Angriff ist nur dann effizient und praktikabel und daher nur dann eine echte Bedrohung, wenn der private Schlüssel d relativ zum Modul n klein gewählt wird. Genauer gesagt funktioniert Wieners Angriff genau dann, wenn

$$3d < \sqrt[4]{n} \quad \text{und} \quad q < p < 2q \tag{7.26}$$

gilt, wobei $n = pq$.

Die Idee soll hier grob skizziert werden. Nach (7.2), und weil der Ver- und Entschlüsselungsexponent die Kongruenz $ed \equiv 1 \pmod{\varphi(n)}$ erfüllen, gibt es eine ganze Zahl $k < d$, so dass

$$ed - k\varphi(n) = 1$$

gilt, woraus folgt:

$$\left| \frac{e}{\varphi(n)} - \frac{k}{d} \right| = \frac{1}{d\varphi(n)}. \tag{7.27}$$

Weil $n = pq > q^2$ ist, gilt $q < \sqrt{n}$. Weil weiter $q < p < 2q$ nach (7.26) gilt, ergibt sich:

$$0 < n - \varphi(n) = p + q - 1 < 2q + q - 1 < 3q < 3\sqrt{n}.$$

Folglich ist

$$\left| \frac{e}{n} - \frac{k}{d} \right| = \left| \frac{ed - kn}{dn} \right| = \left| \frac{1 + k(\varphi(n) - n)}{dn} \right| < \frac{3k\sqrt{n}}{dn} = \frac{3k}{d\sqrt{n}} < \frac{1}{d\sqrt[4]{n}}, \quad (7.28)$$

wobei die letztere Ungleichung aus $3k < 3d < \sqrt[4]{n}$ folgt, was sich wiederum aus $k < d$ und (7.26) ergibt. Wenden wir nochmals unsere Voraussetzung $3d < \sqrt[4]{n}$ an, so folgt:

$$\left| \frac{e}{n} - \frac{k}{d} \right| < \frac{1}{3d^2}. \quad (7.29)$$

Der Verschlüsselungsschlüssel (n, e) ist öffentlich. Die Ungleichung (7.29) sagt, dass der Bruch k/d den Bruch e/n sehr gut approximiert. Um also den privaten Schlüssel d aus dem öffentlichen Schlüssel (n, e) zu ermitteln, könnte ein Angreifer versuchen, das folgende Resultat anzuwenden, das aus der Theorie der Kettenbrüche bekannt ist: Jede Approximation von e/n , die so gut wie die in (7.29) ist, muss eine der Konvergenten der Kettenbruchentwicklung von e/n sein. Ein (*endlicher*) Kettenbruch ist dabei eine rationale Zahl

$$c_1 + \frac{1}{c_2 + \frac{1}{c_3 + \dots + \frac{1}{c_t}}}, \quad (7.30)$$

die als das t -Tupel (c_1, c_2, \dots, c_t) natürlicher Zahlen dargestellt wird, wobei $c_t \neq 0$ ist.

Seien a und b positive natürliche Zahlen mit $\text{ggT}(a, b) = 1$, und sei r_0, r_1, \dots, r_t die Folge der ganzen Zahlen, die durch $\text{EUKLID}(a, b)$ erzeugt werden, siehe Abbildung 2.1. Das heißt, es gilt $r_0 = a$, $r_1 = b$ und $r_{i+1} \equiv r_{i-1} \pmod{r_i}$ für $1 \leq i < t$, siehe zum Beispiel die linke Spalte von Tabelle 2.1. Sei $c_i = \lfloor r_{i-1}/r_i \rfloor$ für $1 \leq i \leq t$. Dann ist a/b gleich dem Kettenbruch aus (7.30), und (c_1, c_2, \dots, c_t) heißt die Kettenbruchentwicklung von a/b . Für jedes i mit $1 \leq i \leq t$ heißt $\mathbf{C}_i = (c_1, c_2, \dots, c_i)$ die i -te Konvergente von (c_1, c_2, \dots, c_t) . Diese kann als die rationale Zahl $\mathbf{C}_i = x_i/y_i$ geschrieben werden, wobei x_i und y_i definiert sind als die Lösungen der folgenden beiden Rekurrenzen:

$$x_i = \begin{cases} 1 & \text{falls } i = 0 \\ c_1 & \text{falls } i = 1 \\ c_i x_{i-1} + x_{i-2} & \text{falls } i \geq 2 \end{cases} \quad \text{und} \quad y_i = \begin{cases} 0 & \text{falls } i = 0 \\ 1 & \text{falls } i = 1 \\ c_i y_{i-1} + y_{i-2} & \text{falls } i \geq 2. \end{cases} \quad (7.31)$$

Beispiel 7.37 (Kettenbruchentwicklung). Um die Kettenbruchentwicklung von $a/b = 101/37$ zu bestimmen, berechnen wir zunächst $\text{EUKLID}(101, 37)$. Tabelle 7.12 zeigt die resultierenden Werte. Somit ergibt sich die Kettenbruchentwicklung von $101/37$ als $(2, 1, 2, 1, 2, 3)$, was das Folgende bedeutet:

$$\frac{101}{37} = 2 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{1}{3}}}}}.$$

Tabelle 7.12. Berechnung der Kettenbruchentwicklung von 101/37 und ihrer Konvergenten

i	0	1	2	3	4	5	6
r_i	101	37	27	10	7	3	1
c_i		2	1	2	1	2	3
$\mathbf{C}_i = \frac{x_i}{y_i}$		$\frac{2}{1}$	$\frac{3}{1}$	$\frac{8}{3}$	$\frac{11}{4}$	$\frac{30}{11}$	$\frac{101}{37}$

Tabelle 7.12 listet auch die Konvergenten $\mathbf{C}_i = x_i/y_i$ von $(2, 1, 2, 1, 2, 3)$ auf, wobei $1 \leq i \leq 6$. Man kann leicht verifizieren, dass x_i und y_i die Rekurrenzen (7.31) erfüllen.

Die entscheidende Eigenschaft der Konvergenten einer Kettenbruchentwicklung, mit der RSA gebrochen werden kann, wird im folgenden Resultat ohne Beweis angegeben.

Satz 7.38. Sind a, b, c und d positive natürliche Zahlen mit $|a/b - c/d| < 1/(2d^2)$ und $\text{ggT}(a, b) = \text{ggT}(c, d) = 1$, so ist c/d eine der Konvergenten der Kettenbruchentwicklung von a/b .

Nach Satz 7.38 impliziert (7.29), dass k/d eine der Konvergenten der Kettenbruchentwicklung von e/n ist. Weil e/n bekannt ist, genügt es für die Ermittlung von k/d , alle Konvergenten von e/n zu bestimmen und zu überprüfen, ob eine von ihnen die korrekte ist. Falls eine Konvergente $\mathbf{C}_i = x_i/y_i$ von e/n verdächtigt wird, gleich k/d zu sein, berechnet man nun den Wert von $\varphi(n)$ durch

$$\varphi(n) = \frac{e \cdot d - 1}{k} = \frac{e \cdot y_i - 1}{x_i}.$$

Sind n und $\varphi(n)$ bekannt, kann n durch Lösen der quadratischen Gleichung (7.25) faktorisiert werden, deren Wurzeln die Primfaktoren von n sind. Schlägt dieser Test fehl, dann war \mathbf{C}_i nicht die richtige Konvergente, und man fährt fort mit der Überprüfung der nächsten Verdächtigen. Wurde keine der Konvergenten von e/n erfolgreich getestet, so kann man schließen, dass die in (7.26) gemachte Annahme nicht stimmt. In Übung 7.14(a) soll man sich mit einer konkreten Implementierung des Angriffs von Wiener auseinander setzen.

Beispiel 7.39 (Wiener's Angriff). Sei $n = 60477719$. Dies ist übrigens dieselbe Zahl n wie in Beispiel 7.35. Sei $e = 47318087$ der öffentliche Exponent. Der öffentliche Schlüssel ist also $(n, e) = (60477719, 47318087)$. Ein Kryptoanalytiker kennt demnach den Wert

$$\frac{e}{n} = \frac{47318087}{60477719} = 0.78240528549.$$

Wie oben erklärt wurde, ergibt sich bei der Berechnung von

$$\text{EUKLID}(60477719, 47318087)$$

aus den dabei erhaltenen Werten r_i und c_i die Kettenbruchentwicklung von e/n :

$$(0, 1, 3, 1, 1, 2, 8, 1, 9, 4, 1, 4, 1, 1, 4, 2, 1, 1, 2, 2, 3). \quad (7.32)$$

Mit den Rekurrenzen (7.31) können nun die Werte x_i und y_i berechnet werden, woraus sich die 21 Konvergenten $\mathbf{C}_i = x_i/y_i$ dieser Kettenbruchentwicklung von e/n ergeben. Tabelle 7.13 zeigt die ersten 10 Konvergenten; siehe auch Übung 7.14(c).

Tabelle 7.13. Berechnung der Konvergenten der Kettenbruchentwicklung in Wieners Angriff

i	1	2	3	4	5	6	7	8	9	10	...
c_i	0	1	3	1	1	2	8	1	9	4	...
$\mathbf{C}_i = \frac{x_i}{y_i}$	0	1	$\frac{3}{4}$	$\frac{4}{5}$	$\frac{7}{9}$	$\frac{18}{23}$	$\frac{151}{193}$	$\frac{169}{216}$	$\frac{1672}{2137}$	$\frac{6857}{8764}$...

Jede Konvergente ist der Gleichheit mit k/d verdächtig, und eine nach der anderen muss überprüft werden. Die ersten fünf Tests werden fehlschlagen. Wenn jedoch $\mathbf{C}_6 = 18/23$ überprüft wird, erhält man

$$\varphi(n) = \frac{e \cdot y_6 - 1}{x_6} = \frac{47318087 \cdot 23 - 1}{18} = 60462000,$$

was genau der Wert von $\varphi(n)$ aus Beispiel 7.35 ist. Wie in diesem Beispiel berechnet der Kryptoanalytiker nun die Primfaktoren 6719 und 9001 von $n = 60477719$.

Wieners Angriff gelingt hier, weil die Primfaktoren von n ungefähr gleich groß sind und weil $3 \cdot 23 = 69 < 88 = \lfloor \sqrt[4]{60477719} \rfloor$ gilt, also (7.26) erfüllt ist.

Wie oben erwähnt ist Wieners Angriff nur dann eine wirkliche Bedrohung, wenn die Voraussetzungen in (7.26) erfüllt sind, insbesondere also nur dann, wenn $3d < \sqrt[4]{n}$ gilt. Da der Verschlüsselungsexponent e jedoch zuerst gewählt wird und weil er normalerweise klein gewählt wird, um die Verschlüsselung zu beschleunigen, ist es unwahrscheinlich, dass ein kleines d erzeugt wird. Das heißt, ist e klein genug, dann ist d wahrscheinlich groß genug, um Wieners Angriff zu widerstehen. Man sollte allerdings daran denken, dass es gefährlich sein könnte, wenn man die Entschlüsselung durch die Wahl eines kleinen privaten Schlüssels d zu beschleunigen versucht.

Eine andere interessante Beobachtung ist, dass wenn ein Angreifer den Entschlüsselungsexponenten d kennt, sei dieser klein oder groß, dann kann er n mit einem effizienten randomisierten Algorithmus faktorisieren. Das heißt also, das Berechnen von d ist im Sinne effizienter randomisierter Algorithmen nicht leichter, als n zu faktorisieren, siehe Problem 7.3.

Angriff auf niedrige Exponenten (Low-Exponent Attack)

Ein Wert, der für den Verschlüsselungsexponenten e empfohlen und der heutzutage üblicherweise verwendet wird, ist $e = 2^{16} + 1$. Ein Vorteil dieses Wertes für e ist, dass seine Binärdarstellung nur zwei Einsen hat, weshalb der *Square-and-Multiply*-Algorithmus aus Abbildung 7.2 nur relativ wenige Operationen benötigt.² Somit kann effizient verschlüsselt werden.

Jedoch sollte man vorsichtig sein, den öffentlichen Verschlüsselungsexponenten nicht zu klein zu wählen. Ein beliebter Wert von e , der in der Vergangenheit oft verwendet wurde, ist $e = 3$. Angenommen, es kommunizieren drei Parteien im selben System und alle verschlüsseln dieselbe Nachricht m , wobei sie denselben öffentlichen Exponenten $e = 3$ verwenden, aber verschiedene RSA-Moduln, sagen wir n_1, n_2 und n_3 . Dann kann ein Kryptoanalytiker m leicht aus den folgenden drei Schlüsseltexten berechnen:

$$\begin{aligned} c_1 &\equiv m^3 \pmod{n_1} \\ c_2 &\equiv m^3 \pmod{n_2} \\ c_3 &\equiv m^3 \pmod{n_3}. \end{aligned}$$

Da nämlich die Nachricht m kleiner als jeder der Moduln n_i sein muss, ist m^3 kleiner als das Produkt $n_1 n_2 n_3$. Mit dem Chinesischen Restesatz (siehe Satz 2.46) kann man nun die eindeutige Lösung

$$c \equiv m^3 \pmod{n_1 n_2 n_3} = m^3$$

berechnen. So kann m aus c durch gewöhnliches Wurzelziehen ermittelt werden.

Allgemeiner nehmen wir an, dass k „ähnliche“ Klartexte mit demselben Exponenten e verschlüsselt werden:

$$\begin{aligned} c_1 &\equiv (a_1 m + b_1)^e \pmod{n_1} \\ c_2 &\equiv (a_2 m + b_2)^e \pmod{n_2} \\ &\vdots \\ c_k &\equiv (a_k m + b_k)^e \pmod{n_k}, \end{aligned}$$

wobei a_i und b_i , $1 \leq i \leq k$, bekannte Konstanten sind und $k > e(e+1)/2$ und $\min(n_i) > 2^{e^2}$ gilt. Dann kann ein Angreifer das obige System von k Kongruenzen mit so genannten Gitter-Reduktionstechniken in Polynomialzeit nach m lösen, siehe Micciancio und Goldwasser [MG02]. Diese Beobachtung wurde von Håstad in den späten 1980ern gemacht.

Dieser Angriff gibt Anlass zur Sorge, wenn die Nachrichten in einer bekannten Weise miteinander „verwandt“ sind. In diesem Fall sollten sie nicht mit vielen RSA-Schlüsseln der Form (n_i, e) verschlüsselt werden. Eine zu empfehlende Gegenmaßnahme, mit der diese Art von Angriff in der Praxis unterbunden werden kann, besteht darin, die Nachrichten vor der Verschlüsselung mit pseudozufälligen Wörtern zu „polstern“, siehe z.B. [KR95].

² Wie viele genau?

Fälschen von RSA-Signaturen

Abschließend wird ein Chosen-Plaintext-Angriff auf das RSA-Signatur-Verfahren vorgestellt, der auf der Tatsache beruht, dass die RSA-Verschlüsselungsfunktion ein Homomorphismus ist: Ist (n, e) der öffentliche Schlüssel und sind m_1 und m_2 zwei Nachrichten, dann ist

$$m_1^e \cdot m_2^e \equiv (m_1 \cdot m_2)^e \pmod{n}. \quad (7.33)$$

Eine weitere leicht zu verifizierende Kongruenz ist:

$$(m \cdot r^e)^d \equiv m^d \cdot r \pmod{n}. \quad (7.34)$$

Mit den Kongruenzen (7.33) und (7.34) kann ein Angriff auf das digitale RSA-Signatur-Schema ausgeführt werden, siehe Abbildung 7.3 in Abschnitt 7.1.2. Sind nämlich Paare $\langle m_1, \text{sig}_A(m_1) \rangle, \langle m_2, \text{sig}_A(m_2) \rangle, \dots, \langle m_k, \text{sig}_A(m_k) \rangle$ von zueinander gehörigen Nachrichten und ihren Unterschriften gegeben, so kann Erich mit Hilfe der Kongruenzen (7.33) und (7.34) ein neues Nachricht-Signatur-Paar $\langle m, \text{sig}_A(m) \rangle$ durch

$$\begin{aligned} m &= r^e \prod_{i=1}^k m_i^{e_i} \pmod{n}; \\ \text{sig}_A(m) &= r \prod_{i=1}^k (\text{sig}_A(m_i))^{e_i} \pmod{n} \end{aligned}$$

berechnen, wobei r und die e_i beliebig sind. Folglich kann Erich die Unterschrift von Alice fälschen, ohne ihren privaten Schlüssel zu kennen, und Bob wird die Fälschung nicht entdecken, denn es gilt:

$$m \equiv (\text{sig}_A(m))^e \pmod{n}.$$

Dieser Angriff sieht auf den ersten Blick wie ein Known-Plaintext-Angriff aus. Jedoch ist $m_1 \cdot m_2$ in (7.33) in der Regel kein sinnvoller Klartext, selbst wenn m_1 und m_2 welche waren. Deshalb kann Erich Alice' Signatur so nur für Nachrichten fälschen, die womöglich Sinn haben, höchstwahrscheinlich jedoch nicht. Aber er könnte die Nachrichten m_i so wählen, dass eine sinnvolle Nachricht m mit einer gefälschten digitalen Unterschrift entsteht. Dieser Chosen-Plaintext-Angriff kann wieder durch die Technik des „pseudozufälligen Auspolsterns“, welches algebraische Beziehungen zwischen den Nachrichten zerstört, vermieden werden.

„Pseudozufälliges Auspoltern“ (englisch: „pseudorandom padding“) ist ebenso eine nützliche Maßnahme gegen den folgenden Chosen-Ciphertext-Angriff: Erich fängt einen Schlüsseltext c ab, wählt $r \in \mathbb{N}$ zufällig und berechnet $c \cdot r^e \pmod{n}$, welches er dem legitimen Empfänger Bob schickt. Gemäß (7.34) wird Bob das Wort $\hat{c} = c^d \cdot r \pmod{n}$ entschlüsseln, welches höchstwahrscheinlich wie eine Folge zufälliger Zeichen aussieht. Kann Erich aber \hat{c} in die Hand bekommen, so könnte er die ursprüngliche Nachricht m durch

$$m = r^{-1} \cdot c^d \cdot r \pmod{n}$$

berechnen, d.h., er multipliziert einfach mit r^{-1} , dem Inversen von r modulo n .

7.5 Übungen und Probleme

Aufgabe 7.1 (a) Zeige, dass für die Werte von $\varphi(n) = 660$ und $e = 7$ in Beispiel 7.2 der erweiterte Euklidische Algorithmus aus Abbildung 2.2 tatsächlich den privaten Schlüssel $d = 283$ ermittelt, der das Inverse von $7 \bmod 660$ ist.

- (b) Bestimme für den Klartext aus Tabelle 7.1 in Beispiel 7.2 die Decodierung des Schlüsseltextes in ein Wort über $\Sigma = \{A, B, \dots, Z\}$ für jeden der 17 Blöcke.
- (c) Entschlüsse alle 17 Blöcke des Schlüsseltextes aus Tabelle 7.1 gemäß (7.4) und zeige, dass man so den ursprünglichen Klartext erhält.

Aufgabe 7.2 Beweise, dass das digitale RSA-Signatur-Protokoll aus Abbildung 7.3 funktioniert.

Aufgabe 7.3 Beweise, dass es unendlich viele Primzahlen gibt.

Hinweis: Zeige zunächst, dass jede natürliche Zahl $n \geq 2$ einen Primteiler hat. Zeige mit diesem Resultat, dass die Annahme, es gäbe nur endlich viele Primzahlen, zu einem Widerspruch führt.

Aufgabe 7.4 (a) Zeige, dass die Zahl 2 ein Fermat-Zeuge für jedes $n \leq 340$ ist.

Hinweis: Schreibe ein Computer-Programm zur Überprüfung dieser Eigenschaft.

- (b) Zeige, dass die Zahl 2 ein Fermat-Lügner für 341 ist.
- (c) Beweise, dass für jede ungerade zusammengesetzte Zahl n die Zahlen 1 und $n - 1$ trivialerweise Fermat-Lügner für n sind.
- (d) Welche Laufzeit hat der Fermat-Test aus Abbildung 7.5? Beweise deine Antwort.

Aufgabe 7.5 Im Beweis von Lemma 7.8 wird die folgende Behauptung aufgestellt: $a \in \mathbb{Z}_n^*$ gilt genau dann, wenn es ein $b \in \mathbb{Z}_n$ mit $a \cdot b \equiv 1 \bmod n$ gibt. Beweise diese Behauptung.

Hinweis: Nach Definition bedeutet $a \in \mathbb{Z}_n^*$, dass $\text{ggT}(n, a) = 1$ gilt. Auch nützlich ist der erweiterte Euklidische Algorithmus, der eine Linearkombination der gegebenen Zahlen, a und n , dadurch bestimmt, dass er Zahlen x und y mit $\text{ggT}(n, a) = x \cdot n + y \cdot a$ berechnet.

Aufgabe 7.6 Beweise Lemma 7.11: Die Ordnung einer endlichen Gruppe wird von der Ordnung einer jeden ihrer Untergruppen geteilt.

Hinweis: Definiere für eine gegebene Untergruppe $\mathfrak{H} = (H, \circ)$ einer endlichen Gruppe $\mathfrak{G} = (G, \circ)$ die folgende Relation: $x \cong_H y$ genau dann, wenn $y^{-1} \circ x \in H$. Beweise die folgenden beiden Aussagen:

- (a) \cong_H ist eine Äquivalenzrelation, und
- (b) für jedes $x \in G$ existiert eine Bijektion zwischen H und den Äquivalenzklassen von x bezüglich \cong_H .

Zeige unter Verwendung von (a) und (b), dass die Ordnung von \mathfrak{G} gleich dem Produkt der Anzahl der Äquivalenzklassen von \cong_H mit der Ordnung von \mathfrak{H} ist.

Aufgabe 7.7 Der Beweis von Satz 7.19 benutzt insbesondere bestimmte zahlentheoretische Fakten und Lemmata, von denen einige unten aufgeführt werden.

- (a) Beweise Lemma 7.13: Jede Carmichael-Zahl ist das Produkt von mindestens drei verschiedenen Primzahlen.

Hinweis: Der Beweis dieses Lemmas ist nicht ganz einfach. Unter anderen Argumenten wird die Anwendung des Chinesischen Restesatzes und des binomischen Satzes benutzt, in ähnlicher Weise wie im Beweis von Lemma 7.26. Lemma 7.13 findet man zum Beispiel als Lemma 5.1.8 im Buch von Dietzfelbinger [Die04].

- (b) Beweise Lemma 7.14: Jede ungerade Primzahl n hat genau zwei Quadratwurzeln von 1 modulo n , nämlich $\pm 1 \bmod n$.

Hinweis: Wende die Tatsache an, dass eine Zahl a genau dann eine Quadratwurzel von 1 modulo n ist, wenn n die Zahl $(a-1)(a+1)$ teilt.

- (c) Beweise Lemma 7.18: Wenn es einen MR-Zeugen für n gibt, dann ist n zusammengesetzt.

Hinweis: Ein Blick auf den Beweis von Satz 7.19 könnte nützlich sein.

- (d) Zeige, dass der Miller–Rabin-Test aus Abbildung 7.6 bei Eingaben der Länge N in der Zeit $\mathcal{O}(N^3)$ läuft.

- (e) Verstärke die Erfolgswahrscheinlichkeit des Miller–Rabin-Tests so, dass ein Fehler erreicht wird, der beliebig nahe an null liegt. Zeige also, genauer gesagt, für ein nichtfallendes Polynom q , so dass $q(n) \geq 2$ für jedes n gilt, dass Primes in coRP_q liegt, wobei die Klasse RP_q in Definition 6.5 definiert wurde.

Hinweis: Siehe den Beweis von Satz 6.6.

Aufgabe 7.8 (a) Zeige, dass jede der folgenden Zahlen ein MR-Zeuge für 561 ist:

$$52, 59, 62, 65, 70, 71, 74, 80, 83, 86, 89, 92, 95, 98, 100, 325, 556.$$

(b) Zeige, dass jede der folgenden Zahlen ein MR-Lügner für 561 ist:

$$103, 256, 305, 458, 511.$$

- (c) Stelle eine Tabelle analog zu Tabelle 7.5 für die zusammengesetzten Zahl $325 = 5^2 \cdot 13$ auf. Zeige, dass sowohl 32 als auch 318 MR-Lügner für 325 sind, doch dass ihr Produkt $32 \cdot 318 \equiv 101 \bmod 325$ ein MR-Zeuge für 325 ist. Zeige ebenso, dass 293 ein MR-Lügner für 325, doch das Produkt $293 \cdot 318 \equiv 224 \bmod 325$ ein MR-Zeuge für 325 ist. Gibt es weitere Gegenbeispiele für die (inkorrekte) Behauptung, dass die Menge MR-Liars_{325} eine Untergruppe von \mathbb{Z}_{325}^* sei?

Hinweis: Ein weiteres Beispiel $n = 325$ findet man in Tabelle 5.3 von [Die04].

- (d) Finde ähnliche Gegenbeispiele wie in (c) für die Carmichael-Zahl 561.

Aufgabe 7.9 (a) Entwirf einen Polynomialzeit-Algorithmus zur Berechnung des Jacobi-Symbols, der von den in Behauptung 7.22 angegebenen Eigenschaften des Jacobi-Symbols Gebrauch macht.

- (b) Beweise, dass für jede ungerade natürliche Zahl n gilt:

$$\left(\frac{n-1}{n} \right) = (-1)^{(n-1)/2}.$$

- (c) Werte die folgenden Jacobi-Symbole aus:

$$\left(\frac{4335}{6399} \right), \left(\frac{2222}{1111} \right), \left(\frac{1234}{9876} \right), \left(\frac{2365}{7882} \right), \left(\frac{9275}{6273} \right) \text{ und } \left(\frac{4367}{5932} \right).$$

- (d) Beweise, dass jeder SS-Lügner für $n \geq 3$ auch ein Fermat-Lügner für n ist.

Aufgabe 7.10 Das Faktorisierungsproblem kann sowohl, wie in Definition 7.28, als ein funktionales Problem als auch als ein Entscheidungsproblem definiert werden. Für Letzteres gibt es mehrere Möglichkeiten. Beispielsweise kann die *Entscheidungsversion des Faktorisierungsproblems* durch

$$\text{Factoring} = \{ \langle n, k \rangle \mid n \text{ hat keinen Primfaktor kleiner als oder gleich } k \},$$

definiert werden, wobei n und k wieder binär dargestellt werden.

- (a) Beweise, dass die funktionale Version `factoring` aus Definition 7.28 und die obige Sprachversion `Factoring` dieses Problems unter der polynomialzeitbeschränkten Turing-Reduzierbarkeit aufeinander reduziert werden können:

$$\text{Factoring} \in \text{P}^{\text{factoring}} \quad \text{und} \quad \text{factoring} \in \text{FP}^{\text{Factoring}}.$$

- (b) Wie viele Fragen genügen jeweils in diesen beiden Reduktionen?
 (c) Beweise, dass das Faktorisierungsproblem, geeignet als ein Sprachproblem ausgedrückt, in NP liegt.

Aufgabe 7.11 (a) Faktorisiere $n = 3^{21} + 1 = 10460353204$ durch Probiedivision.

(b) Faktorisiere $n = 1241143$ mit Pollards $(p-1)$ -Methode aus Abbildung 7.8.

(c) Beweise, dass eine Ausführung der $(p-1)$ -Methode von Pollard (ohne Berücksichtigung der Rekursion oder des Neustarts mit einer neuen Schranke) in der Zeit $\mathcal{O}(B \log BN^2 + N^3)$ läuft, wobei B die gewählte Schranke ist und N die Länge der (binär dargestellten) zu faktorisierenden Zahl n .

Hinweis: Betrachte die Algorithmen, die in einem Durchlauf der $(p-1)$ -Methode aufgerufen werden. Wie oft wird dabei insbesondere der *Square-and-Multiply*-Algorithmus benutzt? Welche Laufzeit hat er? Welcher Zeitaufwand ist für die Berechnung des größten gemeinsamen Teilers nötig?

Aufgabe 7.12 (a) Faktorisiere $n = 7429$ mit der Methode des quadratischen Siebs.

Hinweis: Verwende die Faktorbasis $\mathfrak{B} = \{-1, 2, 3, 5, 7\}$ und wende die Methode aus Abschnitt 7.3.3 an.

- (b) Sei $n = 106$, also ist $s = \lfloor \sqrt{n} \rfloor = 10$. Wähle die Faktorbasis $\mathfrak{B} = \{-1, 2, 3, 5, 7\}$ und betrachte die Funktion $\sigma(x) = (x+10)^2 - 106$. Wende wie in Tabelle 7.9 für jede Primzahl p in \mathfrak{B} das Sieb mit p auf das folgende Siebintervall an:

$$S = \{-10, -9, \dots, -1, 0, 1, \dots, 9, 10\}.$$

Zeige insbesondere, dass man für $\sigma(6)$ und $\sigma(10)$ Gleichungen erhält, die multipliziert eine Kongruenz der Form (7.23) ergeben, welche auf beiden Seiten Quadrate hat. Warum sind die resultierenden Zahlen a und b nicht geeignet, n zu faktorisieren?

- (c) Führe eine Literaturrecherche durch und ermittle die Primfaktoren der Fermatzahl $F_{11} = 2^{2^{11}} + 1$. Wer entdeckte diese Faktorisierung zuerst, und wann?

Aufgabe 7.13 Sei $n = 35$ der RSA-Modul aus Beispiel 7.36, also ist $\varphi(n) = 24$. Zeige, dass jede mögliche Wahl des Verschlüsselungsexponenten e mit dem Inversen von e modulo 24 übereinstimmt. In diesem Fall sind der Verschlüsselungs- und der Entschlüsselungsexponent also identisch, und RSA wäre mit diesen Parametern eigentlich ein symmetrisches Kryptosystem.

Aufgabe 7.14 (a) Schreibe ein Programm in Pseudocode, das Wieners Angriff implementiert, der in Abschnitt 7.4 vorgestellt wurde.

Hinweis: Siehe Wiener [Wie90] oder Stinson [Sti05].

- (b) Verifiziere, dass die in (7.32) aus Beispiel 7.39 angegebene Kettenbruchentwicklung von $e/n = 0.78240528549$ wirklich korrekt ist.
(c) Berechne die übrigen 11 Konvergenten, C_{11} bis C_{21} , der Kettenbruchentwicklung von $e/n = 47318087/60477719$, welche in Tabelle 7.13 aus Beispiel 7.39 weggelassen wurden.
-

Problem 7.1 (Verminderung des Fehlers im Miller–Rabin-Test)

Beweise, dass der Miller–Rabin-Test aus Abbildung 7.6 eine Fehlerwahrscheinlichkeit von höchstens $1/4$ hat, und zwar durch eine sorgfältigere Analyse.

Hinweis: Es genügt, das Folgende zu zeigen: Ist $n \geq 3$ eine ungerade zusammengesetzte Zahl, so gibt es höchstens $(n-1)/4$ MR-Lügner für n in \mathbb{Z}_n^* . Sei also a ein MR-Lügner für n , d.h., a erfüllt Bedingung (7.8): $a^m \equiv 1 \pmod{n}$, oder es erfüllt Bedingung (7.9): Für ein $j \in \{0, 1, \dots, k-1\}$ gilt $a^{2^j m} \equiv -1 \pmod{n}$.

Wenn a die Bedingung (7.8) erfüllt, dann erfüllt $-a$ die Bedingung (7.9). Sei also j_{\max} die größte Zahl j , für die ein $a \in \mathbb{Z}_n^*$ existiert, das (7.9) erfüllt. Sei $k = 2^{m \cdot j_{\max}}$. Betrachte die Faktorisierung von $n = \prod p^{e_p}$ in Primzahlpotenzen, wobei $e_p \geq 1$ und das Produkt über alle Primfaktoren p von n genommen wird. Definiere die folgenden vier Teilmengen von \mathbb{Z}_n^* :

$$A = \{a \in \mathbb{Z}_n^* \mid a^{n-1} \equiv 1 \pmod{n}\};$$

$$B = \{a \in \mathbb{Z}_n^* \mid a^k \equiv \pm 1 \pmod{p^{e_p}} \text{ für jeden Primfaktor } p \text{ von } n\};$$

$$C = \{a \in \mathbb{Z}_n^* \mid a^k \equiv \pm 1 \pmod{n}\};$$

$$D = \{a \in \mathbb{Z}_n^* \mid a^k \equiv 1 \pmod{n}\}.$$

Dabei gilt $A \subset B \subset C \subset D \subset \mathbb{Z}_n^*$. Insbesondere gehört jeder MR-Lügner a zu C . Zeige nun, dass jede der Teilmengen A , B , C , und D eine Untergruppe von \mathbb{Z}_n^* ist. Nach Lemma 7.11 teilt die Gruppenordnung von C die Zahl $\varphi(n)$. Um zu zeigen, dass es höchstens $(n - 1)/4$ MR-Lügner für n in \mathbb{Z}_n^* gibt, ist also $\varphi(n)/\|C\| \geq 4$ zu beweisen.

Problem 7.2 (Primzahl-Problem)

Beweise, dass $\text{Primes} \in P$ ist.

Hinweis: Siehe Abschnitt 7.2.4 und [AKS02]. Eine andere empfehlenswerte Quelle für diesen Beweis ist [Die04]. Dieses Problem ist allerdings sehr schwierig zu lösen.

Problem 7.3 (Faktorisieren mit bekanntem RSA-Entschlüsselungsexponenten)

Sei (n, e) ein öffentlicher RSA-Schlüssel, und sei d der entsprechende private RSA-Schlüssel. Entwirf einen effizienten (randomisierten) Las-Vegas-Algorithmus, der n mit einer Wahrscheinlichkeit von mindestens $1/2$ faktorisiert, vorausgesetzt, die Eingabe $\langle n, e, d \rangle$ erfüllt $ed \equiv 1 \pmod{\varphi(n)}$.

Hinweis: Siehe Stinson [Sti05].

7.6 Zusammenfassung und bibliographische Notizen

Allgemeine Bemerkungen: Für ihre fundamentalen, bahnbrechenden Beiträge zur Public-Key-Kryptographie erhielten Rivest, A. Shamir und Adleman 2002 den Turing Award. Die Bedeutung von RSA kann schon allein aus der Tatsache ermessen werden, dass seit den späten 1970ern wahrscheinlich kein Kryptographie-Lehrbuch darauf verzichtet hat, RSA vorzustellen.

Primzahltests und Faktorisierungsmethoden werden ebenfalls in vielen Büchern zur Kryptographie präsentiert; siehe zum Beispiel Stinson [Sti05], Salomaa [Sal96], Goldreich [Gol01] und Buchmann [Buc01a, Buc01b]. Koblitz [Kob97] konzentriert sich auf die algebraischen Aspekte der Kryptographie. Eine der gründlichsten und aktuellsten Quellen zu Primzahltests ist Dietzfelbingers Buch [Die04], welches einerseits randomisierte Primzahltests wie etwa den Miller–Rabin-Test bereitstellt, andererseits aber eine umfangreiche Darstellung des Beweises von Agrawal, Kayal und N. Saxena liefert, dass $\text{Primes} \in P$ ist [AKS02]. D. Bernstein [Ber04] gibt eine umfangreiche Taxonomie von Primzahltests an.

Spezielle Bemerkungen: Die Idee der Public-Key-Kryptographie wurde zuerst von Diffie und Hellman in ihrer bahnbrechenden Arbeit [DH76] publiziert. Das allererste konkrete Public-Key-Kryptosystem, das in der offenen Literatur erschien, ist RSA. Sowohl das RSA-Kryptosystem als auch das verwandte digitale Signatur-Schema, das in Abschnitt 7.1 vorgestellt wurde, sind Rivest, Shamir und Adleman [RSA78] zu verdanken. Jahrzehnte später, im Dezember 1997, enthüllte das British Government Communications Headquarters (GCHQ), dass Ellis, Cocks und Williamson, ange stellt bei der Communications Electronics Security Group des GCHQ, unabhängig

und sogar früher das Prinzip der Public-Key-Kryptographie, das heute als RSA bezeichnete Kryptosystem und das jetzt Diffie–Hellman genannte Schlüsseltausch-Protokoll entdeckt hatten. Letzteres wird in Abschnitt 8.1 präsentiert.

Ellis hatte die prinzipielle Idee für Public-Key-Kryptographie bereits 1969. Cocks erfand das jetzt RSA genannte System 1973, etwa vier Jahre, bevor Rivest, Shamir und Adleman es unabhängig 1977 fanden. Und Williamson entdeckte die mathematischen Prinzipien, auf denen das Diffie–Hellman-Protokoll beruht, etwa zur selben Zeit wie Diffie und Hellman [DH76]. Interessanterweise ging Diffie–Hellman zwar RSA voraus, die entsprechenden Erfindungen am GCHQ aber wurden in umgekehrter Reihenfolge gemacht. Die interessante Geschichte dieser geheimen Entwicklungen im nichtöffentlichen Bereich wird von Singh [Sin99] und auch von anderen detaillierter als hier erzählt.

Auf der Grundlage der Ideen von Miller [Mil76] für einen deterministischen Algorithmus für das Primzahl-Problem entwickelte Rabin [Rab80] den nun als Miller–Rabin-Test bekannten randomisierten Algorithmus, der in Abbildung 7.6 zu sehen ist. Unabhängig entwickelten Solovay und Strassen [SS77] ihren Monte-Carlo-Primzahltest, siehe Abbildung 7.7. Für weitere Informationen über das Primzahl-Problem und Primzahltests sei z.B. auf Adleman und Huang [AH87, AH92a], D. Bernstein [Ber04] und Dietzfelbinger [Die04] verwiesen.

Pollards $(p - 1)$ -Methode findet man in [Pol74]. Das in Abschnitt 7.3 vorgestellte quadratische Sieb wurde von Pomerance [Pom85] vorgeschlagen, dessen Arbeit u.a. auf Ideen von Kraitchik [Kra22] beruht, welche wiederum von Fermats Einsichten inspiriert waren. Dixons Methode der zufälligen Quadrate [Dix81] wird z.B. von Stinson [Sti05] präsentiert. Das Zahlkörpersieb, eine jüngere und auch erfolgreichere Siebmethode für das Faktorisieren, wurde in den späten 1980ern entwickelt, ausgehend von einer Idee von Pollard, siehe Lenstra und Lenstra [LL93]. Die Methode der elliptischen Kurven für das Faktorisieren von Zahlen ist H. Lenstra [Len87] zu danken. Die Übungen 7.11(a), 7.11(b) und 7.12(a) sind aus Buchmann [Buc01b].

Es wurde in Abschnitt 7.3 erwähnt, dass das Faktorisierungsproblem ein Kandidat für ein solches Problem ist, das weder in P zu liegen noch NP-vollständig zu sein scheint, genau wie das Graphisomorphie-Problem. Doch die heute bekannten Indizien gegen die NP-Vollständigkeit des Faktorisierungsproblems sind anderer Natur als die gegen die NP-Vollständigkeit des Graphisomorphie-Problems. Die derzeit stärksten Resultate, die nahelegen, dass Factoring wahrscheinlich nicht NP-vollständig ist, gehen auf Fellows und Koblitz [FK92] sowie auf Kayal und N. Saxena [KS04] zurück. Fellows und Koblitz zeigen, dass das Faktorisierungsproblem in $UP \cap coUP$ liegt. Übrigens zeigen Cai und Threlfall [CT04], dass auch das in Definition 2.43 eingeführte Problem der quadratischen Reste in $UP \cap coUP$ liegt.

Kayal und N. Saxena beweisen, dass das Faktorisierungsproblem auf das Problem, die Anzahl der Automorphismen in Ringen zu bestimmen (und verschiedene verwandte Probleme) unter polynomialzeit-beschränkten randomisierten Turing-Reduktionen mit nullseitiger Fehlerwahrscheinlichkeit reduzierbar ist. Bezeichnen wir insbesondere das Zählproblem für Ringautomorphismen als #RA, so beweisen sie, dass Factoring in $ZPP^{\#RA}$ liegt, und sie zeigen auch, dass #RA in $FP^{AM \cap coAM}$ ist. Daraus folgt, dass Factoring in $ZPP^{AM \cap coAM}$ liegt.

Der Superverschlüsselungsangriff geht auf Simmons und Norris [SN77] zurück. Wieners Angriff findet man in [Wie90]; die in Abschnitt 7.4 präsentierte Skizze der Idee beruht auf Stinsons Präsentation in [Sti05]. Eine Lösung von Problem 7.3 ist ebenfalls in [Sti05] zu finden. Boneh und Durfee [BD00] verbesserten Wieners Angriff, so dass er für jeden privaten Schlüssel $d < n^{0.292}$ funktioniert. Ein verallgemeinerter Wiener-Angriff auf RSA wurde von Blömer und May [BM04] vorgeschlagen. Ihre Methode, den RSA-Modul n zu faktorisieren, ist immer dann erfolgreich, wenn der öffentliche Schlüssel (n, e) die Kongruenz $ex + y \equiv 0 \pmod{\varphi(n)}$ für $3x < \sqrt[4]{n}$ und $|y| \in \mathcal{O}(n^{-3/4} ex)$ erfüllt.

Die Verallgemeinerung des Low-Exponent-Angriffs aus Abschnitt 7.4 wurde von Håstad [Hås88] vorgeschlagen. Dieser Angriff wurde später von Coppersmith [Cop97] verbessert. Bleichenbacher [Ble98] schlug einen adaptiven Chosen-Ciphertext-Angriff auf Protokolle vor, die auf RSA beruhen. May [May04] zeigte, dass das Berechnen des geheimen RSA-Schlüssels äquivalent zum Faktorisieren sogar unter polynomialzeit-beschränkten deterministischen Turing-Reduktionen ist. Genauer gesagt entwarf er einen deterministischen Polynomialzeit-Algorithmus zum Faktorisieren eines gegebenen RSA-Moduls n , vorausgesetzt, die Primfaktoren p und q von n haben dieselbe Bitlänge und neben dem öffentlichen RSA-Exponent e ist der private RSA-Schlüssel d bekannt, wobei $e, d < \varphi(n)$ gilt. Zuvor war nur bekannt, dass die Äquivalenz des Faktorisierens zum Berechnen des geheimen RSA-Schlüssels mit polynomialzeit-beschränkten *randomisierten* Turing-Reduktionen gelingt [RSA78].

Mehr Hintergrund zur Sicherheit des RSA-Systems, zu potenziellen Angriffen auf dieses System und zu wirkungsvollen Gegenmaßnahmen zur Verhütung dieser Angriffe findet man in den Übersichtsarbeiten von Moore [Moo92], A. Shamir [Sha95], Kaliski und Robshaw [KR95], Boneh [Bon99] und Rothe [Rot02].

Weitere Public-Key-Kryptosysteme und Protokolle

In diesem Kapitel werden verschiedene wichtige Kryptosysteme und kryptographische Protokolle vorgestellt. Die meisten sind Public-Key-Kryptosysteme, doch in Abschnitt 8.1 beginnen wir mit der Einführung des Schüsseltausch-Protokolls von Diffie und Hellman, das in der symmetrischen Kryptographie sehr nützlich ist. Das Diffie–Hellman-Protokoll löst das Problem der Verteilung eines gemeinsamen geheimen Schlüssels in einem symmetrischen Kryptosystem über einen unsicheren Kanal. Es ist das erste solche Protokoll in der offenen Literatur, siehe die bibliographischen Anmerkungen in Abschnitt 7.6. Wie bereits erwähnt öffnete die Arbeit von Diffie und Hellman auch der Public-Key-Kryptographie die Tür, welche nicht mehr verlangt, dass Alice und Bob sich auf einen gemeinsamen geheimen Schlüssel einigen müssen, bevor sie verschlüsselte Nachrichten austauschen können.

Die Sicherheit des Diffie–Hellman-Protokolls beruht auf der Annahme, dass das Berechnen diskreter Logarithmen ein hartes Problem ist. Viele andere Protokolle und Kryptosysteme gehen ebenfalls von dieser Annahme aus. Insbesondere werden in Abschnitt 8.2 das Public-Key-Kryptosystem und das digitale Signatur-Protokoll von ElGamal präsentiert, deren Sicherheit auch auf der Härte des Problems des diskreten Logarithmus beruht.

In Abschnitt 8.3 wird Rabins Kryptosystem vorgestellt. Die Sicherheit dieses Public-Key-Kryptosystems beruht auf der Härte des Problems, Quadratwurzeln modulo n zu berechnen, eine ebenso harte Aufgabe wie das Faktorisieren großer Zahlen n . Anders als RSA oder das ElGamal-System ist Rabins Kryptosystem daher beweisbar sicher unter der Annahme, dass das Faktorisierungsproblem praktisch nicht effizient lösbar ist.

In Abschnitt 8.4 kehren wir zum Begriff der in Abschnitt 6.3 eingeführten Arthur-Merlin-Spiele zurück. Wie dort bemerkt stehen diese mit der Theorie der interaktiven Beweissysteme in enger Beziehung, welche sowohl in der Komplexitätstheorie als auch für kryptographische Anwendungen wichtig ist. Insbesondere sind Zero-Knowledge-Protokolle interaktive Beweissysteme, die zum Zweck der Authentifikation verwendet werden können. Abschnitt 8.4 präsentiert ein Zero-Knowledge-Protokoll für das Graphisomorphie-Problem, welches bereits in Abschnitt 6.5 untersucht wurde.

In Abschnitt 8.5 wird ein Public-Key-Kryptosystem diskutiert, das auf Merkle und Hellman zurückgeht und dessen Sicherheit auf einem NP-vollständigen Problem basiert. In Abschnitt 8.6 schließlich werden die Protokolle für den Schlüsseltausch und für digitale Signaturen von Rabi, Rivest und Sherman vorgestellt. Die Sicherheit ihrer Protokolle beruht auf komplexitätstheoretischen Einwegfunktionen (im Worst-Case-Modell) mit bestimmten nützlichen algebraischen und Sicherheitseigenschaften. Das heißt, diese Protokolle verwenden zweistellige Einwegfunktionen, die kommutativ, assoziativ und „stark nichtinvertierbar“ sind. Außerdem wird das Problem erörtert, solche Funktionen unter geeigneten komplexitätstheoretischen Annahmen zu konstruieren.

8.1 Diffie–Hellman und das Problem des diskreten Logarithmus

Public-Key-Kryptosysteme sind in der Regel weniger effizient als symmetrische Kryptosysteme. Angenommen, Alice und Bob verwenden aus Gründen der Effizienz ein symmetrisches Kryptosystem. Die Verschlüsselung und die Entschlüsselung erfolgt also mit demselben Schlüssel. Wie ist es ihnen möglich, sich auf einen solchen gemeinsamen geheimen Schlüssel zu einigen, ohne sich vor dem Austausch verschlüsselter Nachrichten privat getroffen und ohne einen teuren sicheren Kanal für die Schlüsselverteilung benutzt zu haben? Würden sie sich auf einen gemeinsamen Geheimschlüssel, der künftig verwendet werden soll, etwa dadurch einigen wollen, dass sie diesen zunächst mit ihrem symmetrischen Kryptosystem verschlüsseln, welchen geheimen Schlüssel hätten sie denn dann zur Verschlüsselung *dieser* Nachricht verwenden sollen? Diese paradoxe Situation ist als das *Problem des Schlüsseltauschs* (englisch „*secret-key agreement problem*“) bekannt.

Die Schlüsselverteilung für symmetrische Systeme stellt ein Problem dar, und es ist um so schwieriger zu lösen, je mehr Nutzer im selben System teilnehmen. Es kommt in kryptographischen Anwendungen häufig vor, dass eine Partei dieselbe Nachricht an mehrere andere Parteien schicken will, manchmal an mehrere hundert oder noch mehr Parteien. Man stelle sich nur ein militärisches Szenario vor, in dem die Nachricht „*ANGRIFF UM 6 UHR FRÜH; OHNE FRÜHSTÜCK*“ an die Soldaten des Sechsten Infanterieregiments geschickt werden soll. Oder man stelle sich ein Szenario vor, in dem alle weltweit agierenden CIA-Agenten gleichzeitig über die Details eines geplanten Terroranschlags, der entdeckt werden konnte, informiert werden sollen, damit sie rechtzeitig geeignete Gegenmaßnahmen koordinieren und diesen Plan verhindern können.

Das Problem des Schlüsseltauschs war seit den Anfängen der Kryptographie als unlösbar betrachtet worden. Entsprechend groß war die Überraschung, als Diffie und Hellman auf eine geniale, simple Idee kamen, um es zu lösen. Mit ihrem Schlüsseltausch-Protokoll können sich Alice und Bob auf einen gemeinsamen Geheimschlüssel einigen, indem sie ein paar Nachrichten austauschen. Der Lauscher Erich jedoch hat keine Ahnung von ihrem Schlüssel, auch wenn er jedes einzelne ausgetauschte Bit kennt, vorausgesetzt, er kann das Problem des diskreten Logarithmus nicht lösen.

8.1.1 Das Schlüsseltausch-Protokoll von Diffie und Hellman

Abbildung 8.1 zeigt das Schlüsseltausch-Protokoll von Diffie und Hellman. Es beruht auf den folgenden zahlentheoretischen Begriffen und Fakten, die in Abschnitt 2.4.1 eingeführt wurden:

- Eine *Primitivwurzel* γ von n (manchmal auch als ein *primitives Element* γ von \mathbb{Z}_n^* bezeichnet) ist definiert als ein Erzeuger der multiplikativen Gruppe \mathbb{Z}_n^* , welche die Ordnung $\varphi(n)$ hat, siehe Definition 2.40. Es gilt also:

$$\langle \gamma \rangle = \{ \gamma^i \mid 0 \leq i < \varphi(n) \} = \mathbb{Z}_n^*$$

Für jede Primzahl p ist \mathbb{Z}_p^* eine Gruppe der Ordnung $\varphi(p) = p - 1$ und hat genau $\varphi(p - 1)$ primitive Elemente; siehe Beispiel 2.41.

- Sei p eine Primzahl, und sei γ eine Primitivwurzel von p . Die *modulare Exponentialfunktion mit der Basis γ und dem Modul p* wurde in Definition 2.42 durch

$$\exp_{\gamma,p}(a) = \gamma^a \bmod p$$

eingeführt, und ihre Umkehrfunktion heißt der *diskrete Logarithmus zur Basis γ und mit Modul p* . Für $\alpha = \exp_{\gamma,p}(a)$ schreiben wir $a = \log_{\gamma,p}\alpha \bmod p$.

Schritt	Alice	Erich	Bob
1	Alice und Bob einigen sich auf eine große Primzahl p und eine Primitivwurzel γ von p ; p und γ sind öffentlich		
2	wählt eine große Zufallszahl a , hält sie geheim und berechnet $\alpha = \gamma^a \bmod p$		wählt eine große Zufallszahl b , hält sie geheim und berechnet $\beta = \gamma^b \bmod p$
3		$\alpha \Rightarrow$ $\Leftarrow \beta$	
4	berechnet ihren Schlüssel $k_A = \beta^a \bmod p$		berechnet seinen Schlüssel $k_B = \alpha^b \bmod p$

Abb. 8.1. Schlüsseltausch-Protokoll von Diffie und Hellman

Das Diffie–Hellman-Protokoll aus Abbildung 8.1 funktioniert, denn es gilt

$$k_A = \beta^a = \gamma^{ba} = \gamma^{ab} = \alpha^b = k_B$$

in der Arithmetik modulo p . Somit berechnen Alice und Bob tatsächlich denselben Schlüssel. Mit dem *Square-and-Multiply*-Algorithmus aus Abbildung 7.2 in Abschnitt 7.1 können Alice und Bob schnell potenzieren und deshalb können sie diesen

Schlüssel effizient bestimmen. Die Berechnung von $\beta^a \bmod p$ erfordert zwar $a - 1$ Multiplikationen, wenn sie naiv ausgeführt wird, aber nicht mehr als $2\log a$ Multiplikationen, wenn *Square-and-Multiply* angewandt wird.

Beispiel 8.1 (Diffie–Hellman-Protokoll). Angenommen, Alice und Bob haben die Primzahl $p = 17$ gewählt und möchten nun eine Primitivwurzel von 17 finden. Weil γ genau dann eine Primitivwurzel von p ist, wenn γ ganz \mathbb{Z}_p^* erzeugt, gilt:

$$\mathbb{Z}_p^* = \{\gamma^i \mid 0 \leq i < p - 1\}.$$

Jedes Element $x \in \mathbb{Z}_p^*$ kann eindeutig als $x = \gamma^i$ für ein i , $0 \leq i < p - 1$, geschrieben werden. Nach Definition 2.34 ist die Ordnung eines Elements x der Gruppe \mathbb{Z}_p^* die kleinste positive natürliche Zahl k , so dass $x^k = 1$ gilt. Somit ist

$$\frac{p - 1}{\text{ggT}(p - 1, i)} \tag{8.1}$$

die Ordnung von $x = \gamma^i$. Es folgt, dass x selbst genau dann eine Primitivwurzel von p ist, wenn $\text{ggT}(p - 1, i) = 1$ gilt, und deshalb gibt es auch, wie wir schon aus Abschnitt 2.4.1 wissen, genau $\varphi(p - 1)$ Primitivwurzeln von p . Wegen $\mathbb{Z}_{16}^* = \{1, 3, 5, 7, 9, 11, 13, 15\}$ ist $\varphi(16) = 8$ die Anzahl der Primitivwurzeln von 17. Man kann sich leicht davon überzeugen, dass 3 eine Primitivwurzel von 17 ist, weil 3 ganz

$$\mathbb{Z}_{17}^* = \{1, 3, 9, 10, 13, 5, 15, 11, 16, 14, 8, 7, 4, 12, 2, 6\}$$

erzeugt. Die übrigen Primitivwurzeln modulo 17 kann man folgendermaßen bestimmen. Zunächst sind alle aufeinander folgenden Potenzen von 3 modulo 17 zu ermitteln, wie in Tabelle 8.1 dargestellt. Nach (8.1) ist ein Element $3^i \bmod 17$ genau dann primitiv, wenn $\text{ggT}(16, i) = 1$ gilt. Tabelle 8.1 zeigt diese primitiven Elemente in grauen Kästchen.

Tabelle 8.1. Berechnung der Primitivwurzeln von 17

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$3^i \bmod 17$	1	3	9	10	13	5	15	11	16	14	8	7	4	12	2	6

Angenommen, Alice und Bob wählen die Primitivwurzel $\gamma = 12$ von 17, siehe Übung 8.1(a). Weiter wählt Alice die geheime Zufallszahl $a = 10$. Sie möchte die Zahl $\alpha = 12^{10} \bmod 17$ an Bob schicken. Mit dem *Square-and-Multiply*-Algorithmus aus Abbildung 7.2 berechnet sie zunächst die Binärentwicklung des Exponenten, $10 = 2^1 + 2^3$, und dann die Werte $12^{2^i} \bmod 17$ für $0 \leq i \leq 3$, siehe Tabelle 8.2.

Multipliziert sie nun die Werte in den grauen Kästchen von Tabelle 8.2, so erhält sie

$$\alpha = 12^{10} \equiv 9 \bmod 17$$

und schickt $\alpha = 9$ an Bob. Inzwischen hat Bob seinen geheimen Exponenten $b = 15$ zufällig gewählt und hat seinen Wert $\beta = 12^{15} \equiv 10 \bmod 17$ mit derselben Prozedur

Tabelle 8.2. Berechnung von $\alpha = 12^{10} \bmod 17$ beim Diffie–Hellman-Protokoll

$12^{2^0} \bmod 17$	$12^{2^1} \bmod 17$	$12^{2^2} \bmod 17$	$12^{2^3} \bmod 17$	$\alpha = 12^{10} \bmod 17$
12	8	13	16	9

bestimmt, siehe Übung 8.1(b). Bob schickt $\beta = 10$ an Alice. Nun berechnen Alice und Bob

$$k_A = 10^{10} \equiv 2 \bmod 17 \quad \text{und} \quad k_B = 9^{15} \equiv 2 \bmod 17$$

und haben so ihren gemeinsamen Geheimschlüssel bestimmt, $k_A = 2 = k_B$.

Erich jedoch steht einem schwierigen Problem gegenüber, wenn er versucht, den Geheimschlüssel von Alice und Bob zu bestimmen, vorausgesetzt, deren Zahlen waren groß genug gewählt. (Unnötig zu sagen, dass die Zahlen im obigen Spielzeugbeispiel viel zu klein sind, einem *Brute-Force*-Angriff zu widerstehen.) Wenn er ihre Kommunikation sorgfältig belauscht hat – und Erich ist dafür berüchtigt, ein wachsamster Lauscher zu sein – dann kennt er lediglich die öffentlichen Werte p und γ und die übertragenen Werte α und β . Um die geheimen Werte a und b aus α und β zu ermitteln, muss er anscheinend einen diskreten Logarithmus lösen. Das Problem des diskreten Logarithmus aber gilt als schwer, siehe Abschnitt 8.1.2. Deshalb betrachtet man $\exp_{\gamma,p}$, die modulare Exponentialfunktion, als eine Kandidatin einer Einwegfunktion, eine Funktion, die zwar leicht zu berechnen, aber nur schwer zu invertieren ist.

Um Verwechslungen zu vermeiden, ist hier zu betonen, dass Definition 3.78 in Abschnitt 3.6.2 den Begriff der (komplexitätstheoretischen) Einwegfunktion im Worst-Case-Modell einführt. Im Gegensatz dazu steht $\exp_{\gamma,p}$ im Verdacht, sogar im Mittel schwer zu invertieren zu sein, ein Merkmal der „Einwegigkeit“, das einerseits eine größere Herausforderung darstellt, das zu verlangen aber andererseits in kryptographischen Anwendungen weitaus sinnvoller ist. Es ist schlimm. Bis heute ist nicht bekannt, ob es Einwegfunktionen gibt oder nicht, noch nicht einmal im weniger anspruchsvollen Worst-Case-Modell. Es ist noch schlimmer. Obwohl wir nicht wissen, ob es irgendeine Einwegfunktion gibt, gehen viele Kryptosysteme und Protokolle von der Annahme aus, dass einige ihrer grundlegenden Bausteine vermutlich Einwegfunktionen sind, und somit basiert ihre Sicherheit lediglich auf einer Annahme oder Hoffnung, die sich als falsch oder trügerisch erweisen kann.

Da das Problem des diskreten Logarithmus hart zu sein scheint, ist der oben erwähnte direkte Angriff auf das Schlüsseltausch-Protokoll von Diffie und Hellman derzeit eigentlich kein Anlass zur Sorge, vorausgesetzt, die Primzahl p und die privaten Exponenten a und b werden hinreichend groß gewählt. Genauer gesagt sollten diese Exponenten nicht kleiner als 2^{160} sein. Im nächsten Abschnitt werden Algorithmen zur Berechnung des diskreten Logarithmus vorgestellt und der oben erwähnte direkte Angriff wird im Zusammenhang mit dem so genannten „Diffie–Hellman–Problem“ diskutiert.

Neben diesem ganz offensichtlichen Angriff gibt es noch andere, indirekte Angriffe auf das Diffie–Hellman-Protokoll, bei denen der Kryptoanalytiker nicht nur ein

passiver Lauscher ist, der versucht, das Protokoll zu brechen, indem er den geheimen Schlüssel aus den Werten α und β bestimmt. Beispielsweise ist Diffie–Hellman gegen den „*Man-in-the-Middle*“-Angriff anfällig, ein *aktiver Angriff*, bei dem der Angreifer versucht, das Protokoll zu seinem Vorteil abzuändern. Angenommen, Erich, der „Mann in der Mitte“, fängt Alice’ Zahl $\alpha = \gamma^a \bmod p$ ab, die an Bob geschickt wurde, und er fängt ebenso Bobs Zahl $\beta = \gamma^b \bmod p$ auf ihrem Weg zu Alice ab. Erich ersetzt dann α und β durch seine eigenen Werte und leitet $\alpha_E = \gamma^e \bmod p$ an Bob und $\beta_E = \gamma^e \bmod p$ an Alice weiter, wobei e Erichs privater Exponent ist. Dem Protokoll gemäß berechnen Alice und Bob nun die Schlüssel

$$k_A = (\beta_E)^a \bmod p \quad \text{und} \quad k_B = (\alpha_E)^b \bmod p,$$

die sie vermeintlich mit ihrem jeweiligen Partner teilen. Jedoch ist k_A in Wirklichkeit ein Schlüssel, den Alice mit Erich teilt, der selbenen Schlüssel in künftigen Kommunikationen mit ihr verwenden wird, vorgeblich als Bob. Er kann diesen Schlüssel einfach bestimmen, indem er

$$k_E = \alpha^e = \gamma^{ae} = \gamma^{ea} = (\beta_E)^a = k_A$$

in der Arithmetik modulo p berechnet. Ebenso kann Erich den Schlüssel k_B für künftige Kommunikationen mit Bob benutzen, in denen Erich so tut, als sei er Alice. Der „*Man-in-the-Middle*“-Angriff hat mit dem Problem der *Authentifikation* zu tun, welches später in Abschnitt 8.4 diskutiert werden wird.

8.1.2 Diskrete Logarithmen und das Diffie–Hellman–Problem

Angenommen, Erich kennt die Werte p , γ , α und β des Diffie–Hellman–Protokolls aus Abbildung 8.1, aber er kennt nicht die geheimen Exponenten a und b . Sein Ziel ist es, den gemeinsamen Geheimschlüssel von Alice and Bob zu ermitteln:

$$k_A = k_B \equiv \gamma^{ab} \bmod p.$$

Dieses Problem, das man das *Diffie–Hellman–Problem* nennt, ist formal als ein funktionales Problem definiert. Aber man kann es auch, alternativ zur unten definierten funktionalen Version `diffie-hellman`, als ein Entscheidungsproblem definieren, das hier zur Unterscheidung mit Diffie–Hellman bezeichnet wird. Beide Versionen sind unter polynomialzeit-beschränkten Turing–Reduktionen äquivalent, siehe Übung 8.2(a). Die funktionale Version ist jedoch für kryptographische Anwendungen wichtiger oder passender.

Ebenso kann man das (*funktionale*) *Problem des diskreten Logarithmus* aus Definition 2.42 alternativ als ein Entscheidungsproblem definieren, siehe Übung 8.2(b). Auch hier ist die funktionale Version in kryptographischer Hinsicht relevanter. Wir definieren das funktionale Problem des diskreten Logarithmus unten für beliebige multiplikative Gruppen und verallgemeinern so Definition 2.42. Wir nehmen dabei an, dass Gruppen geeignet durch einen Erzeuger repräsentiert sind, siehe Abschnitt 2.4.2.

Definition 8.2 (Diskreter Logarithmus und Diffie–Hellman–Problem).

- Das (funktionale) Problem des diskreten Logarithmus, *kurz mit dlog bezeichnet, ist folgendermaßen definiert: Gegeben eine multiplikative Gruppe (G, \cdot) , ein Element $\gamma \in G$ der Ordnung n und ein Element $\alpha \in \langle \gamma \rangle$, berechne das eindeutige Element a mit $0 \leq a \leq n - 1$, so dass*

$$a = \log_{\gamma} \alpha.$$

(Äquivalent: Gegeben γ und α , berechne das eindeutige Element a mit $\gamma^a = \alpha$.)

- Das (funktionale) Diffie–Hellman–Problem, *kurz mit diffie–hellman bezeichnet, ist folgendermaßen definiert: Gegeben eine multiplikative Gruppe (G, \cdot) , ein Element $\gamma \in G$ der Ordnung n und zwei Elemente α und β in $\langle \gamma \rangle$, berechne ein Element $\delta \in \langle \gamma \rangle$, so dass*

$$\log_{\gamma} \delta \equiv (\log_{\gamma} \alpha)(\log_{\gamma} \beta) \pmod{n}.$$

(Äquivalent: Gegeben $\gamma^a \pmod{n}$ und $\gamma^b \pmod{n}$, berechne $\gamma^{ab} \pmod{n}$.)

Wäre Erich in der Lage, diskrete Logarithmen effizient zu berechnen, so könnte er das Diffie–Hellman–Problem lösen, denn er könnte Alice' privaten Exponenten $a = \log_{\gamma} \alpha \pmod{p}$ aus p , γ und α bestimmen, und ebenso könnte er Bobs privaten Exponenten $b = \log_{\gamma} \beta \pmod{p}$ aus p , γ und β bestimmen. Also ist das Berechnen diskreter Logarithmen nicht leichter als das Lösen des Diffie–Hellman–Problems. Dieses Argument kann leicht von \mathbb{Z}_p^* auf beliebige endliche multiplikative Gruppen verallgemeinert werden, was den folgenden Fakt beweist.

Fakt 8.3 Das Diffie–Hellman–Problem ist unter polynomialzeit-beschränkten Turing–Reduktionen auf das Problem des diskreten Logarithmus reduzierbar. Das heißt, $\text{diffie–hellman} \in \text{FP}^{\text{dlog}}$.

Die umgekehrte Frage, ob das Problem des diskreten Logarithmus mindestens so schwer wie das Diffie–Hellman–Problem ist, bleibt eine unbewiesene Vermutung. Das Diffie–Hellman–Protokoll hat derzeit keinen Beweis der Sicherheit, noch nicht einmal in dem Sinn, dass es zu brechen so schwer wie das Berechnen diskreter Logarithmen wäre, was ja selbst ein Problem ist, dessen genaue Komplexität eine offene Frage ist. Im Rest dieses Abschnitts werden kurz einige Algorithmen für das Problem des diskreten Logarithmus diskutiert.

Erschöpfende Suche

Die erste Beobachtung ist, dass das Problem des diskreten Logarithmus durch eine erschöpfende Suche gelöst werden kann: Gegeben γ und α wie in Definition 8.2, berechne sukzessive

$$\gamma, \gamma^2, \gamma^3, \dots,$$

bis der eindeutige Exponent a mit $\gamma^a = \alpha$ gefunden ist. Dazu muss man also $\gamma^i = \gamma \cdot \gamma^{i-1}$ für $i = 2, 3, \dots, n - 1$ berechnen. Nehmen wir an, dass das Ausführen

einer Gruppenoperation konstante Zeit kostet, so benötigt dieser naive *Brute-Force*-Algorithmus die Zeit $\mathcal{O}(n)$, was exponentiell in der Länge von n und somit exponentiell in der Eingabelänge ist.

Shanks' „Baby-Step Giant-Step“-Algorithmus

Abbildung 8.2 präsentiert Shanks' „Baby-Step Giant-Step“-Algorithmus, der effizienter als der obige Algorithmus zur erschöpfenden Suche für das Problem des diskreten Logarithmus ist. Shanks' Algorithmus erreicht diese Beschleunigung allerdings nur auf Kosten eines größeren Speicherplatzbedarfs.

```

SHANKS( $G, n, \gamma, \alpha$ ) {
    //  $G$  eine multiplikative Gruppe,  $\gamma \in G$  ein primitives Element der Ordnung  $n$ ,  $\alpha \in \langle \gamma \rangle$ 
     $s := \lceil \sqrt{n} \rceil$ ;
    for ( $i = 0, 1, \dots, s - 1$ ) { Füge  $(\gamma^{is}, i)$  zur Liste  $\mathcal{L}_1$  hinzu; }
    Sortiere die Elemente von  $\mathcal{L}_1$  bezüglich ihrer ersten Komponenten;
    for ( $j = 0, 1, \dots, s - 1$ ) { Füge  $(\alpha\gamma^{-j}, j)$  zur Liste  $\mathcal{L}_2$  hinzu; }
    Sortiere die Elemente von  $\mathcal{L}_2$  bezüglich ihrer ersten Komponenten;
    Finde ein Paar  $(\delta, i) \in \mathcal{L}_1$  und ein Paar  $(\delta, j) \in \mathcal{L}_2$ , d.h., finde zwei Paare mit identischen ersten Komponenten;
    return „ $\log_{\gamma} \alpha = is + j$ “ und halte;
}

```

Abb. 8.2. Shanks' „Baby-Step Giant-Step“-Algorithmus

Um $\log_{\gamma} \alpha$ für gegebene Werte α und γ zu berechnen, wobei γ ein primitives Element der Ordnung n ist, ermittelt Shanks' Algorithmus zunächst $s = \lceil \sqrt{n} \rceil$. Setzen wir nun

$$a = is + j, \quad 0 \leq j < s,$$

so ergibt sich

$$\alpha = \gamma^a = \gamma^{is+j}. \tag{8.2}$$

Wir möchten $a = \log_{\gamma} \alpha$ bestimmen. Aus der Gleichung (8.2) folgt $\alpha\gamma^{-j} = (\gamma^s)^i$. Die Paare $(\alpha\gamma^{-j}, j)$ mit $0 \leq j < s$ sind die (bezüglich der ersten Komponenten sortierten) Elemente der Liste \mathcal{L}_2 , welche die „Baby“-Schritte darstellen. Kommt das Paar $(1, j)$ in \mathcal{L}_2 für ein j vor, so sind wir fertig, denn aus $\alpha\gamma^{-j} = 1$ folgt $\alpha = \gamma^j$, also ist in diesem Fall $a = j$ die Lösung des Problems des diskreten Logarithmus. Andernfalls bestimmen wir

$$\delta = \gamma^s$$

und suchen ein Gruppenelement δ^i , $1 \leq i < s$, das als die erste Komponente eines Elements in \mathcal{L}_2 vorkommt. Die Elemente $(\gamma^s)^i = \gamma^{is}$ werden in der Liste \mathcal{L}_1 gesammelt, wieder bezüglich der ersten Komponenten sortiert, und diese Elemente stellen

die großen (d.h. „giant“) Schritte dar. Wird ein Paar (γ^{is}, i) in \mathcal{L}_1 gefunden, so dass (γ^{is}, j) in der Liste \mathcal{L}_2 der „Baby“–Schritte vorkommt, so ist das Problem des diskreten Logarithmus gelöst, denn aus

$$\alpha \gamma^{-j} = \delta^i = \gamma^{is}$$

folgt $\alpha = \gamma^{is+j}$, also gilt $a = \log_\gamma \alpha = is + j$.

Beispiel 8.4 (Shanks’ Algorithmus). Angenommen, wir wollen $a = \log_2 47 \bmod 101$ in der Gruppe \mathbb{Z}_{101}^* mit Shanks’ Algorithmus finden. Das heißt, $p = 101$, $\gamma = 2$ und $\alpha = 47$ sind gegeben. Dabei ist 101 eine Primzahl und 2 ein primitives Element von \mathbb{Z}_{101}^* , siehe Übung 8.3(a). Da $n = p - 1 = 100$ die Ordnung von 2 ist, gilt $s = \lceil \sqrt{100} \rceil = 10$. Es folgt $\gamma^s \bmod p = 2^{10} \bmod p = 14$. Nun ergeben sich die sortierten Listen \mathcal{L}_1 und \mathcal{L}_2 wie in Tabelle 8.3 dargestellt.

Tabelle 8.3. Berechnung der Listen \mathcal{L}_1 und \mathcal{L}_2 für Shanks’ Algorithmus

\mathcal{L}_1	(1, 0)	(14, 1)	(95, 2)	(17, 3)	(36, 4)	(100, 5)	(87, 6)	(6, 7)	(84, 8)	(65, 9)
\mathcal{L}_1 sortiert	(1, 0)	(6, 7)	(14, 1)	(17, 3)	(36, 4)	(65, 9)	(84, 8)	(87, 6)	(95, 2)	(100, 5)
\mathcal{L}_2	(47, 0)	(74, 1)	(37, 2)	(69, 3)	(85, 4)	(93, 5)	(97, 6)	(99, 7)	(100, 8)	(50, 9)
\mathcal{L}_2 sortiert	(37, 2)	(47, 0)	(50, 9)	(69, 3)	(74, 1)	(85, 4)	(93, 5)	(97, 6)	(99, 7)	(100, 8)

Weil $(100, 5)$ in \mathcal{L}_1 und $(100, 8)$ in \mathcal{L}_2 ist, erhalten wir $a = 5 \cdot 10 + 8 = 58$. Man kann nachrechnen, dass wie gewünscht $2^{58} \bmod 101 = 47$ gilt.

Die erste `for`-Schleife in Abbildung 8.2 kann so implementiert werden, dass erst γ^s berechnet und dann durch Multiplikation mit γ^s potenziert wird. Ähnlich kann die zweite `for`-Schleife in Abbildung 8.2 realisiert werden, indem erst das inverse Element γ^{-1} von γ in der Gruppe berechnet und dieses dann potenziert wird. Beide `for`-Schleifen benötigen die Zeit $\mathcal{O}(s)$. Mit einem effizienten Sortieralgorithmus wie etwa Quicksort können die Listen \mathcal{L}_1 und \mathcal{L}_2 in der Zeit $\mathcal{O}(s \log s)$ sortiert werden. Schließlich können die beiden Paare, deren ersten Komponenten in beiden Listen auftauchen, in der Zeit $\mathcal{O}(s)$ gefunden werden, indem man simultan beide Listen durchläuft. Zusammengefasst (und unter Vernachlässigung logarithmischer Faktoren; siehe die Bemerkungen zur $\tilde{\mathcal{O}}$ -Notation in Abschnitt 6.1) kann Shanks’ Algorithmus so implementiert werden, dass er in der Zeit $\mathcal{O}(s) = \mathcal{O}(\sqrt{n})$ läuft und denselben Platzbedarf hat. Wie eben erwähnt ist es bei der Analyse von Algorithmen zur Berechnung des diskreten Logarithmus üblich, logarithmische Faktoren zu vernachlässigen.

Wenn auch Shanks’ Algorithmus effizienter als der Algorithmus zur erschöpfenden Suche ist, so ist er doch kein effizienter Algorithmus. Es gibt eine Vielzahl anderer Algorithmen für das Problem des diskreten Logarithmus, von denen einige besser als Shanks’ Algorithmus sind. Zu den populärsten solcher Algorithmen gehören der so genannte ρ -Algorithmus von Pollard, der Pohlig–Hellman–Algorithmus, die

so genannte „*Index Calculus*“-Methode und ihre Varianten. Letztere ist besonders gut für die Berechnung diskreter Logarithmen in \mathbb{Z}_p^* für Primzahlen p geeignet, und sie ist eng mit Faktorisierungsmethoden wie dem quadratischen Sieb oder dem Zahlkörpersieb verwandt, siehe die Abschnitte 7.3.3 und 7.3.4. Für eine detaillierte Beschreibung und Analyse dieser alternativen Algorithmen für den diskreten Logarithmus sei z.B. auf Stinson [Sti05] und Buchmann [Buc01a, Buc01b] verwiesen. Keiner der bekannten Algorithmen für den diskreten Logarithmus ist effizient. Die genaue Komplexität von $d\log$ bleibt somit vorerst eine interessante offene Forschungsfrage.

8.2 Die Protokolle von ElGamal

Das Diffie–Hellman-Protokoll kann so modifiziert werden, dass man entweder ein Public-Key-Kryptosystem oder aber ein digitales Signatur-Protokoll erhält. Diese Modifikationen gehen auf ElGamal [ElG85] zurück. Die Sicherheit der Protokolle von ElGamal beruht wieder auf der vermuteten Härte des Problems des diskreten Logarithmus. Die für diese Protokolle benötigten zahlentheoretischen Begriffe wurden in Abschnitt 8.1 erklärt.

8.2.1 ElGamals Public-Key-Kryptosystem

Abbildung 8.3 zeigt übersichtlich die einzelnen Schritte des Public-Key-Kryptosystems von ElGamal. Eine genauere Beschreibung und Erklärung dieser Schritte folgt nun.

Schritt 1: Vorbereitung. Wie im Diffie–Hellman-Protokoll einigen sich Alice und Bob auf eine große Primzahl p , so dass das Problem des diskreten Logarithmus in \mathbb{Z}_p^* schwer ist. Sie einigen sich auch auf eine Primitivwurzel γ von p . Sowohl p als auch γ sind öffentlich.

Schritt 2: Schlüsselerzeugung. Bob erzeugt seinen privaten Schlüssel b zufällig und berechnet seinen öffentlichen Schlüssel $\beta = \gamma^b \bmod p$.

Schritt 3: Kommunikation. Bobs öffentlicher Schlüssel β ist nun Alice bekannt.

Schritt 4: Verschlüsselung. Wie üblich werden Nachrichten blockweise verschlüsselt, wobei jeder Block durch ein Element des Klartextraums \mathbb{Z}_p^* repräsentiert wird. Angenommen, Alice möchte Bob den Nachrichtenblock $m \in \mathbb{Z}_p^*$ schicken. Der Schlüsselraum ist $\mathbb{Z}_p^* \times \mathbb{Z}_p^*$, und die beiden Komponenten des Schlüsseltextes $c = (\alpha_1, \alpha_2)$, die m verschlüsseln, werden berechnet durch:

$$\alpha_1 = \gamma^a \bmod p \tag{8.3}$$

$$\alpha_2 = m\beta^a \bmod p. \tag{8.4}$$

Die Verschlüsselungsfunktion $E_{(p, \gamma, \beta, a)}(m) = (\alpha_1, \alpha_2)$ ist gemäß (8.3) und (8.4) definiert. Alice „maskiert“ ihren Klartext m , indem sie ihn mit ihrem „Diffie–Hellman-Schlüssel“ $\beta^a \equiv \gamma^{ba} \bmod p$ multipliziert und so α_2 erhält. Der Wert $\alpha_1 = \gamma^a \bmod p$ ist ebenfalls Teil des Schlüsseltextes und ermöglicht dem legitmierten Empfänger Bob die Entschlüsselung.

Schritt	Alice	Erich	Bob
1	Alice und Bob einigen sich auf eine große Primzahl p und eine Primitivwurzel γ von p ; p und γ sind öffentlich		
2			wählt eine große Zufallszahl b als seinen privaten Schlüssel und berechnet $\beta = \gamma^b \text{ mod } p$
3		$\Leftarrow \beta$	
4	wählt eine große Zufallszahl a und verschlüsselt die Nachricht m so: $\alpha_1 = \gamma^a \text{ mod } p$ $\alpha_2 = m\beta^a \text{ mod } p$		
5		$(\alpha_1, \alpha_2) \Rightarrow$	
6			entschlüsselt durch $\alpha_2 (\alpha_1)^{-b} \text{ mod } p$

Abb. 8.3. ElGamals Public-Key-Kryptosystem

Schritt 5: Kommunikation. Alice schickt Bob den Schlüsseltext $c = (\alpha_1, \alpha_2)$.

Schritt 6: Entschlüsselung. Die Entschlüsselungsfunktion ist durch

$$D_{(p, \gamma, b)}(\alpha_1, \alpha_2) = \alpha_2 (\alpha_1)^{-b} \text{ mod } p \quad (8.5)$$

gegeben. Gemäß (8.5) verwendet Bob seinen privaten Schlüssel b , um zunächst $\gamma^{-ab} \text{ mod } p$ aus $\alpha_1 = \gamma^a \text{ mod } p$ zu berechnen. Dann entfernt er die „Maske“ β^a vom Klartext, indem er α_2 mit γ^{-ab} multipliziert. Zusammengefasst entschlüsselt Bob den Schlüsseltext c durch

$$\alpha_2 (\alpha_1)^{-b} \equiv m\beta^a (\gamma^a)^{-b} \equiv m\gamma^{ba}\gamma^{-ab} \equiv m \text{ mod } p$$

und erhält so den ursprünglichen Klartext m .

Beispiel 8.5 (ElGamals Public-Key-Kryptosystem). Angenommen, Alice und Bob einigen sich auf die Primzahl $p = 101$ und auf die Primitivwurzel $\gamma = 8$ von 101. Bob wählt seinen privaten Schlüssel mit $b = 12$ und berechnet seinen öffentlichen Schlüssel durch

$$\beta = 8^{12} \text{ mod } 101 = 78.$$

Dann wählt Alice ihren privaten Exponenten $a = 33$ und berechnet

$$\beta^a = 78^{33} \text{ mod } 101 = 92.$$

Angenommen, Alice möchte die Nachricht $m = 53$ schicken. Um m zu verschlüsseln, berechnet sie

$$\begin{aligned}\alpha_1 &= \gamma^a \bmod p = 8^{33} \bmod 101 = 51 \\ \alpha_2 &= m\beta^a \bmod p = 53 \cdot 92 \bmod 101 = 28\end{aligned}$$

und schickt Bob den Schlüsseltext $c = (51, 28)$. Am anderen Ende der Leitung empfängt Bob den Schlüsseltext c und entschlüsselt ihn, indem er

$$\alpha_2(\alpha_1)^{-b} \equiv 28(51)^{-12} \equiv 28(51^{-1})^{12} \equiv 28 \cdot 2^{12} \equiv 28 \cdot 56 \equiv 53 \bmod 101$$

berechnet, woraus sich wie gewünscht der Klartext $m = 53$ ergibt.

ElGamals System modifiziert das Diffie–Hellman-Protokoll in der folgenden Weise. Während Alice und Bob ihre „Teilschlüssel“ α bzw. β im Diffie–Hellman-Schema *simultan* berechnen und verschicken, tun sie dies im ElGamal-Protokoll *nacheinander*. Das heißt, Alice muss auf Bobs Wert β warten, damit sie die zweite Komponente ihres Schlüsseltextes, α_2 , berechnen kann, in welcher ihre Nachricht m durch β^a „maskiert“ wird.

Ein weiterer Unterschied zwischen den beiden Protokollen ist, dass Bob seinen öffentlichen Schlüssel β im ElGamal-Protokoll ein für alle Mal (oder zumindest für mehrfachen Gebrauch) erzeugt. So kann er β in mehr als einer Kommunikation mit Alice verwenden, und auch für andere Nutzer als Alice, die ihm womöglich verschlüsselte Nachrichten schicken wollen. Alice jedoch (oder jeder andere Nutzer, der Bob eine Nachricht schicken möchte) muss den geheimen Exponenten a und somit $\alpha_1 = \gamma^a \bmod p$ für jedes Kommunikation mit Bob neu erzeugen, genau wie im Diffie–Hellman-Protokoll.

Bevor im späteren Abschnitt 8.2.3 die Sicherheitsprobleme für das ElGamal-Kryptosystem diskutiert werden, wird nun das digitale ElGamal-Signatur-Schema vorgestellt.

8.2.2 Digitale Signaturen mit ElGamal

ElGamals Public-Key-Kryptosystem aus Abbildung 8.3 kann so modifiziert werden, dass man ein digitales Signatur-Schema erhält, welches in Abbildung 8.4 dargestellt ist. Eine besonders effiziente Variante dieses Protokolls, die aus einer genialen Idee von Schnorr [Sch90] hervorgeht, ist nun der *Digital Signature Standard* der Vereinigten Staaten, siehe [Nat91, Nat92]. Die einzelnen Schritte des ursprünglichen digitalen Signatur-Protokolls von ElGamal werden nun im Detail beschrieben.

Schritt 1: Vorbereitung. Alice und Bob einigen sich auf eine große Primzahl p , so dass das Problem des diskreten Logarithmus in \mathbb{Z}_p^* schwer ist. Sie einigen sich auch auf eine Primitivwurzel γ von p . Sowohl p als auch γ sind öffentlich.

Schritt 2: Signieren der Nachricht. Angenommen, Bob möchte Alice eine Nachricht m schicken. Wie im ElGamal-Kryptosystem wählt Bob zunächst seinen privaten Exponenten b und berechnet $\beta = \gamma^b \bmod p$. Zusätzlich wählt er nun eine geheime Zahl s , die mit $p - 1$ teilerfremd ist, und er hält b und s geheim. Um m zu signieren, berechnet Bob erst $\sigma = \gamma^s \bmod p$ und dann eine Lösung ρ der Kongruenz

Schritt	Alice	Erich	Bob
1	Alice und Bob einigen sich auf eine große Primzahl p und eine Primitivwurzel γ von p ; p und γ sind öffentlich		
2			wählt zwei große Zufallszahlen b und s mit $\text{ggT}(s, p - 1) = 1$ und berechnet seine Signatur für die Nachricht m durch $\text{sig}_B(m) = (\sigma, \rho)$ so: $\beta = \gamma^b \pmod{p}$ $\sigma = \gamma^s \pmod{p}$ $\rho = (m - b\sigma)s^{-1} \pmod{p-1}$
3	$\Leftarrow \langle m, \beta, \text{sig}_B(m) \rangle$		
4	verifiziert Bobs Signatur, indem sie überprüft, ob gilt: $\gamma^m \equiv \beta^\sigma \sigma^\rho \pmod{p}$		

Abb. 8.4. ElGamals digitales Signatur-Schema

$$b\sigma + s\rho \equiv m \pmod{p-1} \quad (8.6)$$

mit dem erweiterten Algorithmus von Euklid, siehe Abbildung 2.1. Dann ist seine Signatur für m durch $\text{sig}_B(m) = (\sigma, \rho)$ definiert.

Schritt 3: Kommunikation. Zusammen mit seiner Nachricht m schickt Bob seine digitale Signatur $\text{sig}_B(m) = (\sigma, \rho)$ und den Wert β an Alice.

Schritt 4: Verifizieren der Signatur. Alice überprüft die Gültigkeit der Signatur gemäß der folgenden Kongruenz:

$$\gamma^m \equiv \beta^\sigma \sigma^\rho \pmod{p}. \quad (8.7)$$

Nach dem Kleinen Fermat (siehe Korollar 2.39) ergibt sich aus (8.6):

$$\gamma^m \equiv \gamma^{b\sigma+s\rho} \equiv \beta^\sigma \sigma^\rho \pmod{p}.$$

Wie gewünscht verifiziert die Bedingung (8.7) demnach korrekt, ob Bobs Signatur gültig ist, womit gezeigt ist, dass ElGamals digitales Signatur-Protokoll funktioniert. Sehen wir uns ein kleines Beispiel an.

Beispiel 8.6 (ElGamals digitales Signatur-Protokoll). Seien $p = 1367$ die Primzahl und $\gamma = 5$ die Primitivwurzel von 1367, die Alice und Bob verwenden. Bob wählt die privaten Exponenten $b = 513$ und $s = 129$; es gilt $\text{ggT}(129, 1366) = 1$. Zunächst berechnet Bob

$$\beta = 5^{513} \pmod{1367} = 855 \quad \text{und} \quad \sigma = 5^{129} \pmod{1367} = 1180.$$

Angenommen, Bob möchte die Nachricht $m = 457$ signieren. Dann muss Bob die Kongruenz

$$513 \cdot 1180 + 129\rho \equiv 457 \pmod{1366}$$

nach ρ lösen. Mit dem erweiterten Algorithmus von Euklid aus Abbildung 2.1 bestimmt er das inverse Element $s^{-1} = 593$ von $s = 129$ modulo 1366 und erhält so die Lösung

$$\rho = (457 - 513 \cdot 1180)593 \pmod{1366} = 955.$$

Nun ist Bobs Signatur für $m = 457$ durch $\text{sig}_B(457) = (1180, 955)$ gegeben, und er schickt Alice das Tripel $\langle 457, 855, (1180, 955) \rangle$. Am anderen Ende der Leitung verifiziert Alice, dass die Signatur gültig ist, indem sie die Kongruenz

$$5^{457} \equiv 1280 \equiv 749 \cdot 750 \equiv 855^{1180} \cdot 1180^{955} \pmod{1367} \quad (8.8)$$

überprüft. Wie üblich verwendet Alice dafür den *Square-and-Multiply*-Algorithmus aus Abbildung 7.2, um die Werte γ^m , β^σ und σ^ρ in der Arithmetik modulo p zu berechnen, siehe Tabelle 8.4. (Die erste Zeile von Tabelle 8.4 gibt die Exponenten 2^i für die jeweilige Basis an, nämlich für $\gamma = 5$, $\beta = 855$ und $\sigma = 1180$.)

Tabelle 8.4. Verifikation der Signatur in ElGamals digitalem Signatur-Protokoll

2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	
5	25	625	1030	108	728	955	236	1016			$\gamma^m \pmod{p} = 1280$
855	1047	1242	588	1260	513	705	804	1192	551	127	$\beta^\sigma \pmod{p} = 749$
1180	794	249	486	1072	904	1117	985	1022	96		$\sigma^\rho \pmod{p} = 750$

Die grauen Kästchen dieser Tabelle enthalten die Werte, die gemäß der Binärenentwicklung der Exponenten multipliziert werden sollen:

$$\begin{aligned} m &= 457 = 2^0 + 2^3 + 2^6 + 2^7 + 2^8; \\ \sigma &= 1180 = 2^2 + 2^3 + 2^4 + 2^7 + 2^{10}; \\ \rho &= 955 = 2^0 + 2^1 + 2^3 + 2^4 + 2^5 + 2^7 + 2^8 + 2^9. \end{aligned}$$

8.2.3 Sicherheit der Protokolle von ElGamal

In diesem Abschnitt wird die Sicherheit der Protokolle von ElGamal diskutiert. Genau wie beim Diffie–Hellman-Protokoll beruht die Sicherheit des ElGamal-Kryptosystems aus Abbildung 8.3 und des digitalen Signatur-Schemas von ElGamal aus Abbildung 8.4 auf der vermuteten Härte des Problems, diskrete Logarithmen zu berechnen. Das heißt, wenn Erich diskrete Logarithmen effizient berechnen kann, dann kann er die ElGamal-Protokolle brechen. Um zum Beispiel das ElGamal-System aus Abbildung 8.3 zu brechen, muss er lediglich Bobs privaten Schlüssel

$$b = \log_\gamma \beta \pmod{p}$$

aus Bobs öffentlichem Schlüssel β und aus der öffentlichen Primzahl p mit der öffentlichen Primitivwurzel γ berechnen.

Andererseits ist es nicht bekannt, ob das Berechnen diskreter Logarithmen und das Brechen eines der ElGamal-Protokolle gleich schwere Probleme sind. Es kann jedoch gezeigt werden, dass das Brechen des ElGamal Public-Key-Kryptosystems und das in Definition 8.2 eingeführte Diffie–Hellman-Problem zwei hinsichtlich ihrer Berechnungshärte äquivalente Probleme sind.

Brechen von ElGamal und das Diffie–Hellman-Problem

Definition 8.7 (Problem des Brechens von ElGamal). Definiere das (funktionale) Problem des Brechens von ElGamal, bezeichnet mit break-elgamal , folgendermaßen: Gegeben ist ein Quintupel $\langle p, \gamma, \beta, \alpha_1, \alpha_2 \rangle$, wobei p eine Primzahl und γ eine Primitivwurzel von p ist, und β, α_1 , und α_2 sind wie in Abbildung 8.3 für eine beliebige Nachricht m definiert. Die Aufgabe besteht darin, m zu berechnen.

Satz 8.8. Das Problem, ElGamal zu brechen, und das Diffie–Hellman-Problem sind unter polynomialzeit-beschränkten Turing-Reduktionen äquivalent:

$$\text{diffie-hellman} \in \text{FP}^{\text{break-elgamal}} \quad \text{und} \quad \text{break-elgamal} \in \text{FP}^{\text{diffie-hellman}}.$$

Beweis. Angenommen, der Lauscher Erich hat einen Algorithmus D zur Lösung des Diffie–Hellman-Problems. Er möchte mit Hilfe von D ElGamals Kryptosystem brechen. Seien p eine Primzahl und γ eine Primitivwurzel von p . Seien β, α_1 und α_2 die Erich bekannten Werte, die für eine gegebene Nachricht m wie in Abbildung 8.3 gezeigt übertragen werden. Bei Eingabe von $\langle p, \gamma, \beta, \alpha_1, \alpha_2 \rangle$ möchte Erich die entsprechende Nachricht m berechnen. Ein Blick auf Abbildung 8.3 verrät, dass $\alpha_1 = \gamma^a \pmod{p}$ und $\beta = \gamma^b \pmod{p}$ gilt. Mit seinem Algorithmus D berechnet Erich also $\gamma^{ab} \pmod{p}$ aus α_1 und β . Außerdem gilt

$$\alpha_2 = m\beta^a \equiv m\gamma^{ab} \pmod{p}.$$

Folglich kann Erich die Nachricht m erhalten, indem er $\alpha_2 \gamma^{-ab} \pmod{p} = m$ mit dem erweiterten Algorithmus von Euklid aus Abbildung 2.1 berechnet.

Sei umgekehrt angenommen, dass Erich einen Algorithmus E zum Brechen des ElGamal-Kryptosystems hat. Seien $p, \gamma, \beta, \alpha_1$ und α_2 wie in Abbildung 8.3 für eine beliebige Nachricht m gegeben. Mit Hilfe von E kann Erich m aus $\langle p, \gamma, \beta, \alpha_1, \alpha_2 \rangle$ ermitteln. Um das Diffie–Hellman-Problem für die gegebenen Werte $\alpha_1 = \gamma^a \pmod{p}$ und $\beta = \gamma^b \pmod{p}$ zu lösen, simuliert Erich den Algorithmus E mit der Eingabe $\langle p, \gamma, \beta, \alpha_1, 1 \rangle$, also für den spezifischen Wert von $\alpha_2 = 1$, und erhält so die entsprechende Nachricht m . Es folgt:

$$m\beta^a \equiv m\gamma^{ab} \equiv 1 \pmod{p}.$$

Um also $\gamma^{ab} = m^{-1} \pmod{p}$ zu bestimmen, genügt es, das inverse Element von m modulo p wieder mit dem erweiterten Algorithmus von Euklid aus Abbildung 2.1 zu berechnen. \square

Bit-Sicherheit diskreter Logarithmen

Wir haben gesehen, dass sowohl das Diffie–Hellman-Protokoll als auch die Protokolle von ElGamal nur dann sicher sind, wenn es schwer ist, diskrete Logarithmen zu berechnen. Auch haben wir bemerkt, dass das Problem des diskreten Logarithmus allgemein für ein hartes Problem gehalten wird, auch wenn ein wirklicher Beweis seiner Härte bisher nicht gelungen ist. In diesem Abschnitt betrachten wir einige eingeschränkte Varianten dieses Problems, bei denen danach gefragt wird, einzelne Bits eines diskreten Logarithmus zu bestimmen. Insbesondere betrachten wir das folgende Problem und untersuchen die Frage, ob es leicht oder schwer ist.

Definition 8.9 (Bit-Problem des diskreten Logarithmus). Definiere das (funktionale) Bit-Problem des diskreten Logarithmus, bezeichnet mit dlogbit , folgendermaßen: Gegeben ist ein Quadrupel $\langle p, \gamma, \alpha, i \rangle$, wobei p eine Primzahl und γ eine Primitivwurzel von p , $\alpha \in \mathbb{Z}_p^*$ und i eine ganze Zahl mit $1 \leq i \leq \lceil \log(p-1) \rceil$ ist. Berechnet werden soll das i -te am wenigsten signifikante Bit in der Binärdarstellung von $\log_\gamma \alpha \bmod p$.

Tabelle 8.5. Instanzen des Bit-Problems des diskreten Logarithmus in Beispiel 8.10

i	1	2	3	4	5	6	7
$\text{dlogbit}(\langle 101, 2, 47, i \rangle)$	0	1	0	1	1	1	0

Beispiel 8.10 (Bit-Problem des diskreten Logarithmus). In Beispiel 8.4 wurde mit Shanks' Algorithmus der Wert $\log_2 47 \bmod 101 = 58$ berechnet. Jedes Element von \mathbb{Z}_{101}^* kann binär mit höchstens $\lceil \log 100 \rceil = 7$ Bits dargestellt werden. Wegen $58 = 2^5 + 2^4 + 2^3 + 2^1$ hat die Binärdarstellung von 58, $\text{bin}(58) = 111010$, sechs Bits, wobei führende Nullen weggelassen werden. Das am wenigsten signifikante Bit von $\text{bin}(58)$ ist hier die am weitesten rechts stehende Null. Allgemein ist das am wenigsten signifikante Bit von $\text{bin}(n)$ der Koeffizient von 2^0 in der Binärentwicklung von n . Dieses Bit bestimmt die Geradzahligkeit (oder „Parität“) von n : Es ist eins, falls n ungerade ist, und es ist null, falls n gerade ist. Für die Instanzen $\langle 101, 2, 47, i \rangle$ von dlogbit zeigt Tabelle 8.5 die Funktionswerte von $\text{dlogbit}(\langle 101, 2, 47, i \rangle)$ für alle möglichen Werte von i , $1 \leq i \leq 7$, wobei führende Nullen weggelassen werden.

Nun wird gezeigt, dass das Bit-Problem des diskreten Logarithmus für Instanzen mit $i = 1$ effizient gelöst werden kann. Anders gesagt kann die Parität diskreter Logarithmen leicht berechnet werden. In Abschnitt 2.4.1 wurden der Begriff des quadratischen Rests modulo einer Primzahl (siehe Definition 2.43) sowie das Kriterium von Euler (siehe Theorem 2.44) angegeben, die im Folgenden eine Rolle spielen.

Satz 8.11. Ist $\langle p, \gamma, \alpha, 1 \rangle$ eine Instanz des Bit-Problems des diskreten Logarithmus, so kann $\text{dlogbit}(\langle p, \gamma, \alpha, 1 \rangle)$ in Polynomialzeit berechnet werden.

Beweis. Sei $\langle p, \gamma, \alpha, 1 \rangle$ eine gegebene Instanz des Bit-Problems des diskreten Logarithmus, d.h., p ist prim, γ eine Primitivwurzel von p und $\alpha \in \mathbb{Z}_p^*$. Das am wenigsten signifikante Bit der Binärdarstellung von $\log_\gamma \alpha \bmod p$ soll bestimmt werden.

Definiere die Funktion $s : \mathbb{Z}_p^* \rightarrow \mathbb{Z}_p^*$ durch

$$s(w) = w^2 \bmod p.$$

Nach Definition 2.43 ist $\text{QR} = \{(x, n) \mid x \in \mathbb{Z}_n^*, n \in \mathbb{N} \text{ und } x \equiv w^2 \bmod n\}$ die Menge der quadratischen Reste modulo einer ganzen Zahl. Definiere für unsere Primzahl p :

$$\text{QR}_p = \{w^2 \bmod p \mid w \in \mathbb{Z}_p^*\}.$$

Wegen $p \equiv 0 \bmod p$ gilt $s(w) = s(p - w)$. Außerdem gilt:

$$\begin{aligned} x^2 \equiv w^2 \bmod p &\iff p \text{ teilt } (x - w)(x + w) \\ &\iff x \equiv \pm w \bmod p. \end{aligned}$$

Folglich hat jedes $z \in \text{QR}_p$ bezüglich s genau zwei Urbilder. Es folgt

$$\|\text{QR}_p\| = \frac{p-1}{2}.$$

Genau die Hälfte der Elemente von \mathbb{Z}_p^* sind also quadratische Reste modulo p , und die übrige Hälfte der Elemente von \mathbb{Z}_p^* sind quadratische Nichtreste modulo p .

Weil γ eine Primitivwurzel von p ist, gilt $\gamma^a \in \text{QR}_p$, falls der Exponent a gerade ist. Da die $(p-1)/2$ Elemente $\gamma^0, \gamma^2, \dots, \gamma^{p-3}$ paarweise verschieden sind, sind sie genau die Elemente von QR_p , d.h.,

$$\text{QR}_p = \{\gamma^{2i} \bmod p \mid 0 \leq i \leq (p-3)/2\}.$$

Demnach ist ein Element α genau dann ein quadratischer Rest modulo p , wenn $\log_\gamma \alpha$ gerade ist. Das heißt, das am wenigsten signifikante Bit der Binärdarstellung von $\log_\gamma \alpha$ ist genau dann null, wenn α in QR_p ist, was nach Eulers Kriterium zu der Bedingung $\alpha^{(p-1)/2} \equiv 1 \bmod p$ äquivalent ist. Folglich gilt:

$$\text{dlogbit}(\langle p, \gamma, \alpha, 1 \rangle) = 0 \iff \alpha^{(p-1)/2} \equiv 1 \bmod p.$$

Da aber $\alpha^{(p-1)/2} \equiv 1 \bmod p$ mit der Methode des schnellen Potenzierens (siehe Abbildung 7.2) effizient berechnet werden kann, stellt das Euler-Kriterium einen effizienten Algorithmus zur Berechnung von $\text{dlogbit}(\langle p, \gamma, \alpha, 1 \rangle)$ bereit. \square

Wie schwer ist es aber, $\text{dlogbit}(\langle p, \gamma, \alpha, i \rangle)$ für Werte $i > 0$ zu berechnen? Schreiben wir $p-1 = r2^q$, wobei r ungerade ist, so ist $\text{dlogbit}(\langle p, \gamma, \alpha, i \rangle)$ für alle $i \leq q$ leicht zu berechnen. Die Berechnung von $\text{dlogbit}(\langle p, \gamma, \alpha, q+1 \rangle)$ ist im Gegensatz dazu vermutlich eine schwere Aufgabe: Sie ist bezüglich polynomialzeit-beschränkter Turing-Reduktionen mindestens so schwer, wie das allgemeine Problem des diskreten Logarithmus in \mathbb{Z}_p^* zu lösen. Der Beweis von Satz 8.12 wird dem Leser als Problem 8.1 überlassen.

Satz 8.12. Sei $\langle p, \gamma, \alpha, i \rangle$ eine Instanz des Bit-Problems des diskreten Logarithmus, und sei $p - 1 = r2^q$ für eine ungerade Zahl r . Dann gilt:

- Für jedes $i \leq q$ kann $\text{dlogbit}(\langle p, \gamma, \alpha, i \rangle)$ in Polynomialzeit berechnet werden, und
- $\log_\gamma \alpha \bmod p$ kann in $\text{FP}^{\text{dlogbit}(\langle p, \gamma, \alpha, q+1 \rangle)}$ berechnet werden.

Brechen des digitalen Signatur-Schemas von ElGamal

In Abschnitt 4.1 wurden verschiedene Angriffsarten auf ein Kryptosystem vorgestellt, um unterschiedliche Stufen der Sicherheit (oder der Anfälligkeit) eines Kryptosystems zu charakterisieren. Insbesondere wurden dort *Ciphertext-only*-, *Known-Plaintext*-, *Chosen-Plaintext*-, *Chosen-Ciphertext*- und *Key-only*-Angriffe unterschieden. Beispiele dieser Angriffsarten wurden in den Kapiteln 4 und 7 gegeben.

Beim Brechen eines Kryptosystems ist das Ziel des Kryptoanalytikers gewöhnlich, den verwendeten privaten Schlüssel und den verschlüsselten Klartext zu bestimmen. Beim Versuch, ein digitales Signatur-Schema zu brechen, ist der Kryptoanalytiker hingegen auf ein anderes Ziel aus, nämlich auf das Fälschen der Unterschrift von signierten Nachrichten. Die folgenden spezifischen Fälschungsarten werden dabei üblicherweise unterschieden:

- **Totaler Bruch:** Dem Kryptoanalytiker gelingt es, den privaten Schlüssel des Senders in einem digitalen Signatur-Schema zu ermitteln; zum Beispiel Bobs geheime Zahlen b und s in ElGamals digitalem Signatur-Schema, das in Abbildung 8.4 präsentiert wurde, oder Alice' geheimen Schlüssel d im digitalen RSA-Signatur-Schema, das in Abbildung 7.3 präsentiert wurde. Mit Hilfe dieses privaten Schlüssels kann der Kryptoanalytiker Erich eine gültige Unterschrift für jede Nachricht seiner Wahl erzeugen.
- **Selektive Fälschung:** Dem Kryptoanalytiker gelingt es mit einer nicht vernachlässigbaren Wahrscheinlichkeit, eine gültige Signatur für eine Nachricht zu erzeugen, die nicht von ihm gewählt wurde. Das heißt, wenn Erich eine Nachricht m abfängt, die zuvor von Bob nicht signiert worden war, so ist Erich in der Lage, eine gültige Signatur für m mit einer bestimmten, nicht zu kleinen Erfolgswahrscheinlichkeit zu erzeugen.
- **Existenzielle Fälschung:** Dem Kryptoanalytiker gelingt es, eine gültige Unterschrift für mindestens eine Nachricht zu erzeugen, die von Bob zuvor nicht unterschrieben wurde. Hier ist keine spezifische Erfolgswahrscheinlichkeit erforderlich.

Wieder kann man verschiedene Stufen der Sicherheit für ein digitales Signatur-Schema unterscheiden, abhängig davon, welche Information dem Kryptoanalytiker während des Angriffs zur Verfügung steht. Insbesondere werden gewöhnlich die folgenden Angriffsarten betrachtet (wobei wieder die üblichen englischen Bezeichnungen verwendet und nicht ins Deutsche übertragen werden):

- **Key-only-Angriff:** Der Kryptoanalytiker Erich kennt lediglich Bobs öffentlichen Schlüssel.

- **Known-Message-Angriff:** Erich kennt zusätzlich zum öffentlichen Schlüssel einige Paare von Nachrichten und zugehörigen Signaturen.
- **Chosen-Message-Angriff:** Erich kennt den öffentlichen Schlüssel und erhält eine Liste von Signaturen Bobs, die zu einer Liste von Nachrichten gehören, die Erich selbst gewählt hat.

Die Begriffe „*Known-Plaintext-Angriff*“ bzw. „*Chosen-Plaintext-Angriff*“ werden manchmal an Stelle von „*Known-Message-Angriff*“ bzw. „*Chosen-Message-Angriff*“ verwendet. Ein Beispiel eines Chosen-Plaintext-Angriffs auf das digitale RSA-Signatur-Schema findet man in Abschnitt 7.4.

Wenden wir uns nun der Sicherheit des digitalen Signatur-Schemas von ElGamal zu. Welche Möglichkeiten hat ein Angreifer wie Erich, Bobs Signatur in diesem Schema zu fälschen, ohne dass er Bobs private Exponenten b und s kennt? Natürlich kennt Erich Bobs öffentlichen Schlüssel $\beta = \gamma^b \bmod p$, wobei γ eine Primitivwurzel der Primzahl p ist. Angenommen, Erich möchte die Nachricht m mit einer gefälschten Signatur unterschreiben, die ganz wie Bobs Signatur aussieht. Dem Protokoll aus Abbildung 8.4 gemäß muss Erich Elemente σ und ρ so wählen, dass gilt:

$$\begin{aligned}\sigma &= \gamma^s \bmod p \\ \rho &= (m - b\sigma)s^{-1} \bmod (p - 1).\end{aligned}$$

Die Reihenfolge, in der σ und ρ gewählt werden, spielt hier eine Rolle. Nehmen wir zunächst an, dass Erich σ zuerst wählt und dann das entsprechende ρ . Nach (8.7) muss er dann offenbar den diskreten Logarithmus $\log_{\sigma} \beta^{-\sigma} \gamma^m \bmod p$ lösen.

Zieht Erich es andererseits vor, ρ zuerst zu wählen und dann das entsprechende σ , so steht er dem Problem gegenüber, die ElGamal-Verifizierungsbedingung (8.7)

$$\gamma^m \equiv \beta^{\sigma} \sigma^{\rho} \bmod p$$

nach der Unbekannten σ zu lösen. Für dieses Problem kennt man ebenfalls keinen effizienten Algorithmus. Allerdings scheint es nicht eng mit einem anderen gründlich untersuchten Problem wie dem des diskreten Logarithmus verwandt zu sein. Es könnte daher durchaus sein, dass es einen solchen effizienten Algorithmus für dieses Problem gibt, der uns bisher nur entgangen ist. Auch könnte es sein, dass es eine raffinierte Methode gibt, σ und ρ simultan so zu ermitteln, dass (σ, ρ) eine gültige Signatur für m ist, die Alice akzeptieren würde, wenn sie sie mit Bobs öffentlichem Schlüssel β verifiziert.

Schließlich könnte Erich versuchen, σ und ρ gleichzeitig zu wählen und dann (8.7) nach dem unbekannten Wert m zu lösen. In diesem Fall steht er wieder dem Problem gegenüber, einen diskreten Logarithmus zu berechnen, nämlich

$$\log_{\gamma} \beta^{\sigma} \sigma^{\rho} \bmod p.$$

Dieser Zugang hat den Nachteil, dass die signierte Nachricht in Abhängigkeit von der Wahl von σ und ρ womöglich nicht sinnvoll ist. Jedoch ist auch dieser Angriff nicht praktikabel, weil das Lösen diskreter Logarithmen eine schwere Aufgabe ist.

Jedoch ist Erich in der Lage, eine gültige ElGamal-Signatur für eine zufällig gewählte Nachricht m zu erzeugen, indem er σ , ρ und m simultan wählt. Dieser Key-only-Angriff erlaubt also eine existentielle Fälschung, und er funktioniert folgendermaßen. Seien x und y ganze Zahlen mit $0 \leq x \leq p - 2$ und $0 \leq y \leq p - 2$. Schreiben wir σ als $\sigma = \gamma^x \beta^y \pmod{p}$, so nimmt die ElGamal-Verifikationsbedingung (8.7) die Gestalt

$$\gamma^m \equiv \beta^\sigma (\gamma^x \beta^y)^\rho \pmod{p}$$

an, welche äquivalent zu

$$\gamma^{m-x\rho} \equiv \beta^{\sigma+y\rho} \pmod{p} \quad (8.9)$$

ist. Nun ist aber (8.9) genau dann wahr, wenn eine der folgenden beiden Bedingungen erfüllt ist:

$$m - x\rho \equiv 0 \pmod{p-1}; \quad (8.10)$$

$$\sigma + y\rho \equiv 0 \pmod{p-1}. \quad (8.11)$$

Für gegebene x und y und unter der Annahme, dass $\text{ggT}(y, p-1) = 1$ gilt, können (8.10) und (8.11) leicht nach ρ und m gelöst werden, und es ergibt sich:

$$\begin{aligned} \sigma &= \gamma^x \beta^y \pmod{p}; \\ \rho &= -\sigma y^{-1} \pmod{p-1}; \\ m &= -x\sigma y^{-1} \pmod{p-1}. \end{aligned}$$

Nach Konstruktion ist (σ, ρ) eine gültige ElGamal-Signatur für die Nachricht m .

Beispiel 8.13 (Key-only-Angriff auf ElGamals digitales Signatur-Protokoll). Wie in Beispiel 8.6 seien $p = 1367$ eine gegebene Primzahl und $\gamma = 5$ eine gegebene Primivwurzel von 1367. Bobs private Exponenten, die Erich nicht kennt, sind $b = 513$ und $s = 129$. Erich kennt jedoch Bobs öffentlichen Schlüssel $\beta = 855$. Angenommen, er wählt $x = 33$ und $y = 77$. Mit dem erweiterten Euklidischen Algorithmus aus Abbildung 2.2 überprüft er, dass $\text{ggT}(77, 1366) = 1$ gilt, und bestimmt dann das inverse Element $y^{-1} = 479$ von $y = 77$ modulo $p - 1$. Dann berechnet Erich:

$$\begin{aligned} \sigma &= 5^{33} \cdot 855^{77} \equiv 906 \cdot 343 \equiv 449 \pmod{1367}; \\ \rho &= -449 \cdot 479 \equiv 757 \pmod{1366}; \\ m &= -33 \cdot 449 \cdot 479 \equiv 393 \pmod{1366}. \end{aligned}$$

Folglich ist $(449, 757)$ eine gültige ElGamal-Signatur für die Nachricht 393. Zur Probe überzeugt man sich davon, dass Alice mit der Verifikationsbedingung (8.7):

$$5^{393} \equiv 1125 \equiv 930 \cdot 817 \equiv 855^{449} \cdot 449^{757} \pmod{1367}$$

Erichs Fälschung akzeptieren wird. Es ist allerdings nicht gesagt, dass 393 tatsächlich eine Nachricht ist, die Erich gern an Alice mit Bobs gefälschter Signatur schicken würde.

Eine andere existenzielle Fälschung von ElGamal-Signaturen kann mit einem Known-Message-Angriff erreicht werden. Angenommen, Erich kennt eine frühere Signatur $(\hat{\sigma}, \hat{\rho})$ für eine Nachricht \hat{m} . Er kann dann durch eine Fälschung von Bobs Signatur neue Nachrichten signieren. Wie üblich seien p eine Primzahl mit einer Primitivwurzel γ und β Bobs öffentlicher Schlüssel. Seien $x, y, z \in \mathbb{Z}_{p-1}$ so gewählt, dass $\text{ggT}(x\hat{\sigma} - z\hat{\rho}, p-1) = 1$ gilt. Erich berechnet:

$$\begin{aligned}\sigma &= \hat{\sigma}^x \gamma^y \beta^z \bmod p; \\ \rho &= \hat{\rho} \sigma (x\hat{\sigma} - z\hat{\rho})^{-1} \bmod (p-1); \\ m &= \sigma (x\hat{m} + y\hat{\rho}) (x\hat{\sigma} - z\hat{\rho})^{-1} \bmod (p-1).\end{aligned}\tag{8.12}$$

Man kann leicht überprüfen, dass die ElGamal-Verifikationsbedingung (8.7),

$$\gamma^m \equiv \beta^\sigma \sigma^\rho \bmod p,$$

erfüllt ist, siehe Übung 8.7(a). Folglich ist (σ, ρ) eine gültige Signatur für die Nachricht m . Hier ist ein kleines Beispiel zur Illustration.

Beispiel 8.14 (Known-Message-Angriff auf ElGamals digitales Signatur-Protokoll). Wie in den Beispielen 8.6 und 8.13 wählt Bob die Primzahl $p = 1367$, die Primitivwurzel $\gamma = 5$ von 1367 und den öffentlichen Schlüssel $\beta = 855$. Angenommen, Erich kennt Bobs Signatur $(\hat{\sigma}, \hat{\rho}) = (1180, 955)$ für die Nachricht $\hat{m} = 457$, siehe Beispiel 8.6. Erich wählt nun die ganzen Zahlen $x = 19$, $y = 65$ und $z = 23$. Mit dem erweiterten Euklidischen Algorithmus aus Abbildung 2.2 überprüft er, dass

$$\text{ggT}(x\hat{\sigma} - z\hat{\rho}, p-1) = \text{ggT}(455, 1366) = 1$$

gilt, und berechnet das inverse Element $455^{-1} \bmod 1366 = 1363$. Gemäß (8.12) berechnet Erich schließlich:

$$\begin{aligned}\sigma &= 1180^{19} \cdot 5^{65} \cdot 855^{23} \equiv 963 \cdot 674 \cdot 219 \equiv 1184 \bmod 1367; \\ \rho &= 955 \cdot 1184 \cdot 455^{-1} \equiv 984 \bmod 1366; \\ m &= 1184 \cdot (19 \cdot 457 + 65 \cdot 955) \cdot 455^{-1} \equiv 656 \bmod 1366.\end{aligned}$$

Folglich ist $(1184, 984)$ eine gültige Signatur für die Nachricht 656, die der ElGamal-Verifikationsbedingung (8.7) standhält:

$$5^{656} \equiv 452 \equiv 698 \cdot 314 \equiv 855^{1184} \cdot 1184^{984} \bmod 1367.$$

Also akzeptiert Alice Erichs gefälschte Signatur.

Die oben angegebenen Angriffe auf das ElGamal-Signatur-Schema bewirken beide eine existenzielle Fälschung. Es ist derzeit nicht bekannt, ob diese Angriffe so verschärft werden können, dass sie sogar eine selektive Fälschung ermöglichen. Insofern stellen diese Angriffe keine praktisch bedeutsame Bedrohung für das ElGamal-Signatur-Schema dar. Zur Verhinderung dieser Angriffe ist als eine Gegenmaßnahme der Gebrauch kryptographischer Hash-Funktionen zu empfehlen, wie unten skizziert wird.

Das Prinzip des *Hashing* wurde bereits in Abschnitt 6.5 erläutert, dort allerdings in einem ganz und gar anderen Zusammenhang. In der Kryptographie kann eine Hash-Funktion $h : \Sigma^* \rightarrow T$ dazu verwendet werden, einen so genannten „*message digest*“ vorbestimmter Länge aus einer gegebenen Nachricht beliebiger Länge zu erzeugen. Als eine übliche Länge für die Hash-Werte in T wählt man z.B. 160 Bits. Möchte Bob eine Nachricht m signieren, so berechnet er zunächst den *message digest* $\tau = h(m)$, welcher ein Element der Hash-Tabelle T ist. Dann berechnet er seine Signatur $\sigma = \text{sig}_B(\tau)$ für τ , wobei er ein digitales Signatur-Schema wie ElGamal verwendet, und schickt (m, σ) an Alice. Um die Signatur zu verifizieren, rekonstruiert sie erst den *message digest* $\tau = h(m)$, wofür sie die öffentliche Hash-Funktion h benutzt, und überprüft dann die Gültigkeit der Signatur σ für τ .

Um existenzielle Fälschungen durch Key-only- oder Known-Message-Angriffe zu verhindern, müssen Hash-Funktionen bestimmte Eigenschaften besitzen, damit sie als kryptographisch sicher gelten können. Nehmen wir beispielsweise an, dass Erich einen Known-Message-Angriff ausführt. Also kennt er bereits ein Paar (m, σ) , wobei m eine zuvor von Bob signierte Nachricht und σ die Signatur für den *message digest* $\tau = h(m)$ ist, der aus m mittels einer Hash-Funktion h erzeugt wurde. Da h öffentlich ist, kann Erich τ ermitteln. Er könnte dann versuchen, eine andere Nachricht $\tilde{m} \neq m$ zu finden, so dass $h(\tilde{m}) = h(m)$ gilt. Dies würde ihn in die Lage versetzen, Bobs Signatur für die Nachricht \tilde{m} zu fälschen, denn das Paar (\tilde{m}, σ) enthält eine gültige Signatur σ für \tilde{m} . Diese Art eines Angriffs kann verhindert werden, indem man von der verwendeten Hash-Funktion verlangt, dass sie auf dem relevanten Bereich „kollisionsfrei“ in dem Sinn ist, dass es für ein gegebenes m sehr schwer ist, eine Nachricht \tilde{m} mit $\tilde{m} \neq m$ und $h(\tilde{m}) = h(m)$ zu ermitteln.

Zum Abschluss dieses Abschnitts sei erwähnt, dass Vorsicht bei der Wahl und der Verwendung der Parameter des Systems geboten ist, um zu vermeiden, dass Erich ein totaler Bruch des digitalen ElGamal-Signatur-Schemas gelingt. Insbesondere darf Bobs geheimer Exponent s (siehe Abbildung 8.4) niemals enthüllt werden. Wenn Erich s kennt, dann ist es für ihn nur eine Sache der Routine, mittels (8.6) Bobs geheimen Exponenten b aus m und der Signatur (σ, ρ) durch

$$b \equiv (m - s\rho)\sigma^{-1} \pmod{p-1}$$

zu berechnen. Dieser Known-Message-Angriff führt zu einem totalen Bruch des digitalen Signatur-Schemas von ElGamal, und Erich kann fortan Bobs Signatur nach Belieben fälschen. Ähnlich kann ein Known-Message-Angriff so durchgeführt werden, dass ElGamals Schema total gebrochen wird, wenn derselbe Wert von s zweimal zum Signieren verschiedener Nachrichten verwendet wird, siehe Übung 8.8.

8.3 Rabins Public-Key-Kryptosystem

Im Jahr 1979 entwickelte Rabin ein Public-Key-Kryptosystem, dessen Sicherheit auf der Schwierigkeit beruht, Quadratwurzeln modulo einer Zahl n zu berechnen. Sein Kryptosystem ist beweisbar sicher gegen Chosen-Plaintext-Angriffe, sofern man die

Berechnungshärte des Faktorisierungsproblems voraussetzt, d.h., unter der Annahme, dass es schwer ist, die Primfaktoren p und q von $n = pq$ zu finden.

8.3.1 Rabins Kryptosystem

Abbildung 8.5 präsentiert die einzelnen Schritte des Kryptosystems von Rabin, welche nun detaillierter erklärt werden.

Schritt	Alice	Erich	Bob
1			wählt zufällig zwei große verschiedene Primzahlen p und q mit $p \equiv q \equiv 3 \pmod{4}$, hält diese geheim und berechnet seinen öffentlichen Schlüssel $n = pq$
2		$\Leftarrow n$	
3	verschlüsselt die Nachricht m durch $c = m^2 \pmod{n}$		
4		$c \Rightarrow$	
5			entschlüsselt c durch Berechnen von $m = \sqrt{c} \pmod{n}$

Abb. 8.5. Rabins Public-Key-Kryptosystem

Schritt 1: Schlüsselerzeugung. Bob wählt zufällig zwei große verschiedene Primzahlen p und q , für die $p \equiv q \equiv 3 \pmod{4}$ gilt. Das Paar (p, q) ist sein privater Schlüssel. Dann berechnet er den Modul $n = pq$, seinen öffentlichen Schlüssel.

Schritt 2: Kommunikation. Alice kennt nun Bobs öffentlichen Schlüssel n .

Schritt 3: Verschlüsselung. Mit dem öffentlichen Schlüssel n berechnet Alice ihren Schlüsseltext c durch Quadrieren ihrer Nachricht m modulo n , d.h., die Verschlüsselungsfunktion $E_n : \mathbb{Z}_n^* \rightarrow \mathbb{Z}_n^*$ ist definiert durch

$$E_n(m) = c = m^2 \pmod{n}.$$

Schritt 4: Kommunikation. Alice schickt den Schlüsseltext c an Bob.

Schritt 5: Entschlüsselung. Die Entschlüsselungsfunktion ist gegeben durch

$$D_{(p,q)}(c) = \sqrt{c} \pmod{n}. \quad (8.13)$$

Es ist bisher noch nicht klar, wie der private Schlüssel (p, q) zur (autorisierten) Entschlüsselung benutzt wird. Im Allgemeinen gilt das Berechnen von Quadratwurzeln modulo einer Zahl mit unbekannten Primfaktoren als ein schweres Problem. Da Bob jedoch die Primfaktoren p und q von n kennt, kann er von der

Tatsache Gebrauch machen, dass das Berechnen von m gemäß (8.13) äquivalent zum Lösen der folgenden beiden Kongruenzen nach den Werten m_p und m_q ist:

$$(m_p)^2 \equiv c \pmod{p}; \quad (8.14)$$

$$(m_q)^2 \equiv c \pmod{q}. \quad (8.15)$$

Nach dem Kriterium von Euler (siehe Satz 2.44) kann Bob effizient entscheiden, ob c ein quadratischer Rest modulo p ist oder nicht, und ebenso, ob c ein quadratischer Rest modulo q ist oder nicht. Doch mit Eulers Kriterium kann man diese Quadratwurzeln nicht wirklich bestimmen. Glücklicherweise kann Bob die Annahme, dass $p \equiv q \equiv 3 \pmod{4}$ gilt, benutzen und folgendermaßen vorgehen. Zunächst berechnet er

$$m_p = c^{(p+1)/4} \pmod{p} \quad \text{und} \quad m_q = c^{(q+1)/4} \pmod{q}.$$

Dabei muss c eine Quadratwurzel modulo p sein, sofern c ein gültiger Schlüsseltext ist, falls also c tatsächlich durch Verschlüsselung einer Nachricht entstanden ist. Wieder nach Eulers Kriterium ist c genau dann eine Quadratwurzel modulo p , wenn $c^{(p-1)/2} \equiv 1 \pmod{p}$ gilt. Somit gilt

$$(\pm m_p)^2 \equiv (\pm c^{(p+1)/4})^2 \equiv c^{(p+1)/2} \equiv c^{(p-1)/2} c \pmod{p} \equiv c \pmod{p},$$

womit (8.14) bewiesen ist. Also sind $\pm m_p$ die beiden Quadratwurzeln von c modulo p . Analog sind $\pm m_q$ die beiden Quadratwurzeln von c modulo q , womit (8.15) gezeigt ist. Dann bestimmt Bob mit dem Chinesischen Restesatz die vier Quadratwurzeln von c modulo n . Zu diesem Zweck wendet er erst den erweiterten Euklidischen Algorithmus aus Abbildung 2.2 an, um ganzzahlige Koeffizienten z_p und z_q zu berechnen, so dass gilt:

$$z_p p + z_q q = 1.$$

Schließlich berechnet er gemäß Satz 2.46:

$$s = (z_p p m_q + z_q q m_p) \pmod{n} \quad \text{und} \quad t = (z_p p m_q - z_q q m_p) \pmod{n}.$$

Man kann leicht überprüfen, dass $\pm s$ und $\pm t$ die vier Quadratwurzeln von c modulo n sind. Welche davon den „richtigen“ Klartext repräsentiert, ist nicht unmittelbar klar, siehe Anmerkung 8.15.1.

Anmerkung 8.15. 1. Die Verschlüsselung in Rabins System ist nicht injektiv. Das heißt, weil n das Produkt zweier Primzahlen ist, hat jeder Schlüsseltext c vier Quadratwurzeln modulo n ; vergleiche Lemma 7.14 und den nachfolgenden Absatz. Deshalb hat Rabins System den Nachteil, dass bei der Entschlüsselung nicht nur der ursprüngliche Klartext zum Vorschein kommt, sondern auch drei andere Quadratwurzeln von c , die hoffentlich „hinreichend sinnlos“ sind, damit sie eliminiert werden können.

Mit welcher Methode zur Unterscheidung könnte Bob die „richtige“ Entschlüsselung und die drei „falschen“ Entschlüsselungen auseinander halten? Eine Möglichkeit besteht darin, dem Klartext eine spezielle Struktur zu geben, die den ursprünglichen Klartext kennzeichnet. Beispielsweise könnte man einen spezifizierten Klartext-Block wiederholen oder man könnte die letzten 64 Bits von m an m anhängen. Allerdings wäre in diesem Fall der Beweis, dass das Brechen des Rabin-Systems bezüglich der Berechnungshärte äquivalent zum Faktorisierungsproblem ist, nicht mehr gültig, siehe Satz 8.18.

2. Rabins System funktioniert auch für Primfaktoren, die nicht so genannte *Blum-Zahlen* sind, für die also nicht $p \equiv q \equiv 3 \pmod{4}$ verlangt wird. Die Verwendung von Blum-Zahlen vereinfacht jedoch die Analyse dieses Systems. Gilt beispielsweise $p \equiv 1 \pmod{4}$, so ist kein *deterministischer* Polynomialzeit-Algorithmus zur Berechnung der Quadratwurzeln modulo p bekannt, was aber für die effiziente Entschlüsselung nötig ist. Allerdings gibt es für dieses Problem einen effizienten *randomisierten*, genauer einen Las-Vegas-Algorithmus.
3. Schließlich sei angemerkt, dass man in Rabins System auch \mathbb{Z}_n statt \mathbb{Z}_n^* als den Klartext- und Schlüsselraum verwenden könnte.

Beispiel 8.16 (Rabins Public-Key-Kryptosystem). Angenommen, Bob wählt die Primzahlen $p = 43$ und $q = 47$. Es gilt $43 \equiv 47 \equiv 3 \pmod{4}$. Dann berechnet er den Rabin-Modul $n = pq = 2021$. Um die Nachricht $m = 741$ zu verschlüsseln, berechnet Alice

$$c = 741^2 = 549081 \equiv 1390 \pmod{2021}$$

und schickt $c = 1390$ an Bob. Zur Entschlüsselung des Schlüsseltextes c bestimmt Bob zunächst mit dem Verfahren zur schnellen Potenzierung aus Abbildung 7.2 die folgenden Werte:

$$m_p = 1390^{(43+1)/4} = 1390^{11} \equiv 10 \pmod{43};$$

$$m_q = 1390^{(47+1)/4} = 1390^{12} \equiv 36 \pmod{47}.$$

Nun berechnet er mit dem erweiterten Euklidischen Algorithmus aus Abbildung 2.2 die ganzzahligen Koeffizienten $z_p = -12$ und $z_q = 11$, die

$$z_p p + z_q q = -12 \cdot 43 + 11 \cdot 47 = 1$$

erfüllen. Nach Satz 2.46 berechnet er schließlich

$$s = z_p pm_q + z_q qm_p = -12 \cdot 43 \cdot 36 + 11 \cdot 47 \cdot 10 \equiv 741 \pmod{2021};$$

$$t = z_p pm_q - z_q qm_p = -12 \cdot 43 \cdot 36 - 11 \cdot 47 \cdot 10 \equiv 506 \pmod{2021}.$$

Wie man leicht überprüfen kann, sind $\pm s$ und $\pm t$ (also 741, 1280, 506 und 1515) die vier Klartexte, die in denselben Schlüsseltext $c = 1390$ verschlüsselt werden.

8.3.2 Sicherheit des Systems von Rabin

Angenommen, Erich gelingt es, den Rabin-Modul n zu faktorisieren. Er erhält somit Bobs privaten Schlüssel und kann nun jede an Bob geschickte Nachricht entschlüsseln. Das heißt, Rabins System zu brechen ist bezüglich der Berechnungshärte

nicht schwerer als das Faktorisierungsproblem. Umgekehrt wird nun gezeigt, dass das Faktorisieren großer Zahlen bezüglich einer randomisierten Turing-Reduktion mit nullseitigem Fehler nicht schwerer als das Brechen des Rabin-Systems ist, also sind diese beiden Probleme in diesem Sinne gleich schwer. Somit hat Rabins Kryptosystem einen Beweis der Sicherheit, der auf der Annahme beruht, dass das Faktorisierungsproblem nicht praktikabel lösbar ist. In dieser Hinsicht ist Rabins System anderen Public-Key-Systemen wie RSA oder ElGamal überlegen.

Wie oben erwähnt wird zum Beweis dieses Resultats eine polynomialzeit-beschränkte *randomisierte* Turing-Reduktion mit nullseitigem Fehler konstruiert, die das Faktorisierungsproblem auf das (funktionale) Problem, Rabins System zu brechen, reduziert. Das letztgenannte Problem wird formal in Definition 8.17 definiert, wobei

$$\text{QR}_n = \{x^2 \bmod n \mid x \in \mathbb{Z}_n^*\}$$

wie in Abschnitt 8.2.3 die Menge der quadratischen Reste modulo n bezeichnet.

Definition 8.17 (Problem des Brechens von Rabin). Definiere das (funktionale) Problem des Brechens von Rabin, bezeichnet mit `break-rabin`, folgendermaßen: Gegeben ist ein Paar $\langle n, c \rangle$, wobei n das Produkt zweier (unbekannter) Primzahlen in $3 + 4\mathbb{Z}$ und $c \in \text{QR}_n$ ist. Die Aufgabe besteht darin, ein $m \in \mathbb{Z}_n^*$ mit $c = m^2 \bmod n$ zu finden.

Randomisierte Algorithmen, einschließlich der Las-Vegas-Algorithmen, wurden in Abschnitt 6.2.1 eingeführt. Insbesondere besteht die Klasse ZPP aus genau den Entscheidungsproblemen, die durch polynomialzeit-beschränkte randomisierte Algorithmen mit nullseitiger Fehlerwahrscheinlichkeit gelöst werden können. Solche Algorithmen bezeichnet man auch als Las-Vegas-Algorithmen, siehe Definition 6.8. Die randomisierte Turing-Reduktion, die im Beweis von Satz 8.18 konstruiert werden soll, ist ein solcher Las-Vegas-Algorithmus. Statt Entscheidungsproblemen betrifft dieser jedoch Funktionen: er reduziert das funktionale Faktorisierungsproblem auf das funktionale Problem `break-rabin`. Der Einfachheit halber beschränken wir uns auf den Fall, dass die zu faktorisierende Zahl n tatsächlich ein Rabin-Modul ist. Diese Einschränkung ist dadurch gerechtfertigt, dass diese restriktivere Version des Faktorisierungsproblems nicht leichter als das allgemeine Faktorisierungsproblem ist.

Satz 8.18. Der in Abbildung 8.6 dargestellte Algorithmus RANDOM-FACTOR ist ein polynomialzeit-beschränkter Las-Vegas-Algorithmus, der bei Eingabe einer Zahl $n = pq$ mit $p \equiv q \equiv 3 \pmod 4$ sein Funktionsorakel `break-rabin` verwendet, um die Primfaktoren von n mit einer Wahrscheinlichkeit von mindestens $1/2$ zu finden.

Beweis. Sei $n = pq$ der zu faktorisierende Rabin-Modul, d.h., p und q sind Primzahlen mit $p \equiv q \equiv 3 \pmod 4$. Der Algorithmus RANDOM-FACTOR mit Orakel `break-rabin` wird in Abbildung 8.6 präsentiert.

Bei Eingabe n wählt RANDOM-FACTOR zufällig ein Element $x \in \mathbb{Z}_n^*$ unter Gleichverteilung und quadriert es modulo n , um $c \in \text{QR}_n$ zu erhalten. Dann fragt

```
RANDOM-FACTORbreak-rabin(n) {
    // Rabin-Modul  $n = pq$ , für Primzahlen  $p$  und  $q$  mit  $p \equiv q \equiv 3 \pmod{4}$ 
    Wähle zufällig eine Zahl  $x \in \mathbb{Z}_n^*$  unter Gleichverteilung;
     $c := x^2 \pmod{n}$ ;
     $m := \text{break-rabin}(\langle n, c \rangle)$ ;
    // Frage das Orakel nach  $\langle n, c \rangle$  und erhalte eine Antwort  $m$  mit  $c := m^2 \pmod{n}$ 
    if ( $m \equiv \pm x \pmod{n}$ ) return „erfolgloser Abbruch“ und halte;
    else
         $p := \text{ggT}(m - x, n)$ ;
         $q := n/p$ ;
        return „ $p$  und  $q$  sind die Primfaktoren von  $n$ “ und halte;
}
```

Abb. 8.6. Faktorisieren eines Rabin-Moduls mit Orakel zum Brechen von Rabins System

der Algorithmus sein Orakel `break-rabin` nach dem Paar $\langle n, c \rangle$ und erhält die Antwort m , die eine der Quadratwurzeln von c modulo n ist. Die beiden Quadratwurzeln m und x von c modulo n müssen nicht gleich sein. Jedoch müssen m und x einen der folgenden beiden Fälle erfüllen.

Fall 1: $m \equiv \pm x \pmod{n}$. Dann gilt entweder $m = x$ oder $m + x = n$. Somit ist $\text{ggT}(m - x, n)$ entweder n oder 1. In beiden Fällen findet der Algorithmus keinen Primfaktor von n und bricht erfolglos ab.

Fall 2: $m \equiv \pm \alpha x \pmod{n}$, wobei α eine nichttriviale Quadratwurzel von $1 \pmod{n}$ ist. In diesem Fall gilt $m^2 \equiv x^2 \pmod{n}$ und $m \not\equiv \pm x \pmod{n}$. Somit ist $\text{ggT}(m - x, n)$ entweder p oder q , und n ist faktorisierbar.

Um die Erfolgswahrscheinlichkeit von RANDOM-FACTOR abzuschätzen, sei x ein beliebiges zufällig unter Gleichverteilung in \mathbb{Z}_n^* gewähltes Element. Sei α eine nichttriviale Quadratwurzel von $1 \pmod{n}$. Betrachte die Menge

$$R_x = \{\pm x \pmod{n}\} \cup \{\pm \alpha x \pmod{n}\}.$$

Quadriert man irgendein Element r von R_x , so erhält man dasselbe $c = r^2 = x^2 \pmod{n}$. Insbesondere ist die Orakelantwort $m = \text{break-rabin}(\langle n, c \rangle)$ ein Element von R_x , und zwar unabhängig davon, welches der vier Elemente von R_x gewählt wurde, um c zu erhalten. In Fall 2 oben wurde erwähnt, dass der Algorithmus die Primfaktoren von n genau dann findet, wenn $m \equiv \pm \alpha x \pmod{n}$ gilt. Für festes m ist die Wahrscheinlichkeit dafür, dass ein $x \in R_x$ mit $m \equiv \pm \alpha x \pmod{n}$ gewählt wurde, gleich $1/2$. Folglich ist $1/2$ auch die Erfolgswahrscheinlichkeit von RANDOM-FACTOR. \square

Mit den Techniken aus Abschnitt 6.2 kann die Erfolgswahrscheinlichkeit von RANDOM-FACTOR so verstärkt werden, dass sie nahe eins ist.

Satz 8.18 hat zwei interessante Konsequenzen. Einerseits sagt er, dass Rabins Kryptosystem durch einen Chosen-Plaintext-Angriff nicht gebrochen werden kann, sofern Faktorisieren praktisch nicht machbar ist. Dies kann man als einen Vorteil des Systems von Rabin sehen.

Korollar 8.19. Unter der Annahme, dass es für das Faktorisierungsproblem keinen effizienten randomisierten Algorithmus mit nicht vernachlässigbarer Erfolgswahrscheinlichkeit gibt, ist Rabins Kryptosystem sicher gegen Chosen-Plaintext-Angriffe.

Andererseits folgt aber aus Satz 8.18, dass das Rabin-System unsicher gegen Chosen-Ciphertext-Angriffe ist. Das Szenario eines Chosen-Ciphertext-Angriffs geht davon aus, dass ein Kryptoanalytiker zeitweilig Zugriff auf das Entschlüsselungsgerät hat. Wählt er nun nach Belieben einen Schlüsseltext c , so kann er sich dessen zugehörigen Klartext m beschaffen. Dies kann man auch so sagen, dass er (nicht ein hypothetisches Orakel, sondern) einen effizienten Algorithmus zur Berechnung von `break-rabin` besitzt. Nach Satz 8.18 kann sich der Angreifer dies folgendermaßen zunutze machen. Er wählt zufällig einen Klartext x , berechnet $c = x^2 \bmod n$ und entschlüsselt c , um eine Quadratwurzel m von c modulo n zu erhalten. Wie im Beweis von Satz 8.18 kann er dann den Rabin-Modul n mit hinreichend großer Wahrscheinlichkeit faktorisieren und sich so den privaten Schlüssel verschaffen.

Korollar 8.20. Rabins Kryptosystem ist unsicher gegen Chosen-Ciphertext-Angriffe.

Das folgende Spielzeugbeispiel illustriert Satz 8.18 und seine Folgerungen.

Beispiel 8.21 (Faktorisieren durch das Brechen des Systems von Rabin). Sei $n = 23 \cdot 7 = 161$ der gegebene Rabin-Modul. Angenommen, Erich kennt zwar die Primfaktoren 7 und 23 nicht, besitzt aber das Orakel `break-rabin` (alternativ dazu hat er einen effizienten Algorithmus zum Berechnen von `break-rabin`) und kann somit die Quadratwurzeln modulo 161 bestimmen. Nun simuliert Erich den Algorithmus RANDOM-FACTOR aus Abbildung 8.6 für die Eingabe $n = 161$, wählt dabei die Zahl $x = 13$ zufällig; es gilt $\text{ggT}(161, 13) = 1$, also ist 13 in \mathbb{Z}_{161}^* . Dann kann er $c = 13^2 \bmod 161 = 8$ berechnen. Die vier Quadratwurzeln von 8 mod 161 sind in $R_{13} = \{13, 36, 125, 148\}$. Sei m die Orakelantwort auf die Frage $\langle 161, 8 \rangle$, d.h., $m = \text{break-rabin}(\langle 161, 8 \rangle)$.

Für jede mögliche Antwort $m \in R_{13}$ bestimmen wir nun $\text{ggT}(m - x, n)$. Ist $m = 13$, so ist $\text{ggT}(m - x, n) = \text{ggT}(0, 161) = 161$. Und ist $m = 148$, so ist $\text{ggT}(m - x, n) = \text{ggT}(135, 161) = 1$. In beiden Fällen versagt RANDOM-FACTOR beim Versuch, die Primfaktoren von 161 zu finden. Aber wenn $m = 36$ ist, dann gilt $\text{ggT}(m - x, n) = \text{ggT}(23, 161) = 23$, und wenn $m = 125$ ist, gilt $\text{ggT}(m - x, n) = \text{ggT}(112, 161) = 7$. In diesen beiden Fällen hat RANDOM-FACTOR Erfolg und liefert Erich die Primfaktoren von 161. Somit hat Erich eine fünfzigprozentige Chance, n zu faktorisieren, und er kann diese Wahrscheinlichkeit durch die üblichen Amplifikationstechniken (also durch wiederholte unabhängige Versuche) beliebig nahe an eins bringen.

8.4 Arthur-Merlin-Spiele und Zero-Knowledge

„There are known knowns. These are things we know that we know. There are known unknowns. That is to say, there are things we know we don't know. But, there are

also unknown unknowns. These are things we don't know we don't know“, wird ein früherer US-Verteidigungsminister gern zitiert. Man könnte hinzufügen:¹ „*And there is zero-knowledge. These are things we know that somebody else knows, and we provably cannot know what they are.*“

Zero-Knowledge betrifft die Frage, wie man die Kenntnis eines Geheimnisses beweisen kann, ohne es zu verraten. Dieser Begriff wird in diesem Abschnitt formal eingeführt. Er steht in enger Beziehung zur Authentikation, einer zentralen Aufgabe in der Kryptographie, wie schon in Abschnitt 4.1 erwähnt wurde.

Beispielsweise ist der in Abschnitt 8.1 genannte *Man-in-the-Middle*-Angriff auf das Diffie–Hellman-Protokoll möglich, weil Alice Bob gegenüber ihre Identität nicht bewiesen hat, bevor das Protokoll ausgeführt wurde, und weil Bob nicht verifiziert hat, dass Alice tatsächlich die Person ist, die zu sein sie vorgab. Deshalb konnte Erich, der arglistige Mann in der Mitte, in der Kommunikation mit Bob so tun, als sei er Alice. Ähnlich konnte er in der Kommunikation mit Alice so tun, als sei er Bob, da auch sie Bobs Authentizität nicht verifizierte. In diesem Abschnitt wird ein Zero-Knowledge-Protokoll vorgestellt, mit dessen Hilfe Alice und Bob der Falle des schlimmen Erich aus dem Wege gehen können.

Um ihre Identität zu authentifizieren, könnte Alice etwa eine private Information benutzen – ein ganz persönliches *Geheimnis*, das niemand außer ihr kennt. Angenommen, eine „*Trusted Third Party*“ (kurz: TTP, eine dritte Partei also, der alle vertrauen) zertifiziert Alice' Geheimnis, beglaubigt also, dass dieses Geheimnis mit ihr allein verknüpft ist. Beispielsweise könnte sie sich am Bankautomaten mit ihrer PIN (*Personal Identification Number*) authentifizieren. Aber hier ist der Haken: Würde sie Bob ihre Identität dadurch beweisen, dass sie ihm ihr Geheimnis mitteilt, so hat sie es verraten! Es ist dann gar kein Geheimnis mehr! Da er nun Alice' Geheimnis kennt, könnte sich Bob für sie ausgeben, wenn er mit Charlie kommuniziert, einer dritten Partei. Wie kann Alice beweisen, dass sie ihr Geheimnis kennt, ohne auch nur ein Bit an Information darüber zu verraten? Genau darum geht es beim Zero-Knowledge.

Die gerade aufgeworfene Frage ist diese: Wie kann Alice ihr Geheimnis benutzen, um zweifelsfrei ihre Identität zu *beweisen*, so dass Bob ihre Authentizität *verifizieren* kann, aber dabei nichts von ihrem Geheimnis erfährt? Beweiser und Verifizierer sind die Gegenspieler in einem interaktiven Beweissystem, ein Begriff, der in Abschnitt 6.3 in der spezifischen Form von Arthur-Merlin-Spielen eingeführt wurde. Im Vergleich der beiden Terminologien entspricht Merlin dem Beweiser, der durch eine NP-Maschine repräsentiert wird, und Arthur entspricht dem Verifizierer, der durch eine BPP-Maschine repräsentiert wird. Die Idee des Zero-Knowledge wird nun durch die folgende kurze Geschichte illustriert.

Story 8.22 (Zero-Knowledge-Protokoll) *Arthur und Merlin spielen wieder eines ihrer Spiele. Diesmal möchte Arthur die Identität Merlins verifizieren, da er sich nicht sicher ist, ob er mit Merlin oder mit einem anderen Zauberer spricht, der lediglich vorgibt, Merlin zu sein. Um Merlins Identität zu verifizieren, fordert Arthur*

¹ Anmerkung des Übersetzers: Auf eine Übersetzung soll hier und beim erstgenannten Zitat sowie beim Begriff „Zero-Knowledge“ bewusst verzichtet werden.

ihn heraus: Er verlangt von ihm ein Geheimnis, das seine Identität beweist. Genauer gesagt soll Merlin beweisen, dass er einen Zauberspruch kennt, mit dem ein gefährlicher, feuerspeiender Drachen in Schlaf versetzt werden kann. Nur Merlin kennt diesen Spruch.

Der Drache lebt in einem geheimen, unterirdischen Labyrinth, das in Abbildung 8.7 schematisch dargestellt ist und das nur mit Arthurs ausdrücklicher Erlaubnis betreten werden darf. Der Drache sitzt darin genau in der Mitte zwischen dem Heiligen Gral und dem Einen Ring, Der Sie Alle Beherrscht. Wird das Labyrinth durch den linken Eingang betreten, während der Drache wach ist, so kann man nur den Heiligen Gral und nicht den Einen Ring erreichen. Wird das Labyrinth durch den rechten Eingang betreten, während der Drache wach ist, so erreicht man nur den Einen Ring und nicht den Heiligen Gral. Der einzige Weg im Labyrinth vom Heiligen Gral zum Einen Ring oder zurück verläuft am Drachen vorbei und ist nur dann begehbar, wenn der Drache schläft. (Wie der Leser vielleicht weiß, schlafen Drachen nie, außer wenn man sie mit einem Zauberspruch dazu zwingt.)

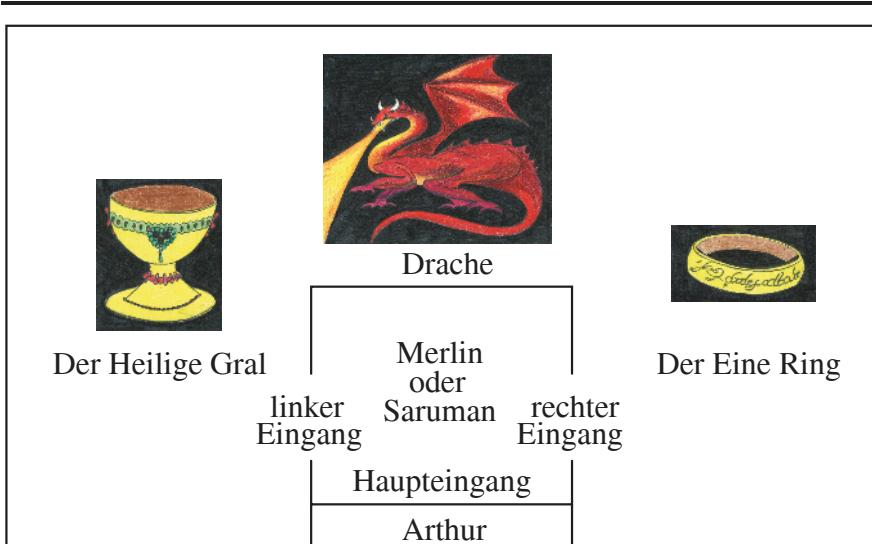


Abb. 8.7. Arthurs Labyrinth

Das Arthur-Merlin-Spiel geht so. Zuerst tritt Merlin durch den Haupteingang in den Vorraum zum Labyrinth, schließt diese Tür und entscheidet sich für entweder den linken oder den rechten Eingang ins Labyrinth. Arthur folgt ihm nun durch den Haupteingang und bleibt im Vorraum stehen; für ihn wäre es viel zu gefährlich, das Labyrinth durch einen der beiden Eingänge zu betreten, denn ihm steht Merlins Zauberspruch nicht zur Verfügung. Er weiß nicht, ob Merlin den linken oder rechten Eingang ins Labyrinth gewählt hat. Aber er fordert Merlin nun heraus, indem er ent-

weder den Heiligen Gral oder den Einen Ring zu sehen verlangt. Wenn Merlin den linken Eingang gewählt hat und Arthur den Heiligen Gral zu sehen wünscht, nimmt Merlin diesen einfach und verlässt das Labyrinth. Ebenso hat Merlin kein Problem, sich zu authentifizieren, wenn er den rechten Eingang gewählt hat und Arthur den Einen Ring zu sehen verlangt. Hat Merlin jedoch den linken Eingang gewählt und Arthur nach dem Einen Ring verlangt, oder hat Merlin den rechten Eingang gewählt und Arthur wünscht den Heiligen Gral zu sehen, so ist Merlin (und nur Merlin) in der Lage, sich als er selbst zu authentifizieren, indem er den Drachen mit seinem (nur ihm bekannten) Zauberspruch in Schlaf versetzt.

Inzwischen hat Saruman der Weiße, der arglistige Zauberer von Orthanc, ebenfalls Camelot erreicht. Er hatte letzthin eine schwere Zeit in Orthanc, aber es gelang ihm, von dort zu fliehen. Nun, von Gandalf dem Grauen geschlagen, ist er auf Rache aus und benötigt die Macht des Einen Rings mehr als je zuvor. Er hat Gerüchte gehört, dass Arthur diesen in seinem geheimen Labyrinth aufbewahrt. Mittels Magie erscheint er daher in der Gestalt Merlins in Camelot und verlangt Einlass in das versteckte Labyrinth. Er verfügt jedoch nicht über Merlins Zauberkraft und kennt nicht dessen geheime Sprüche. Tatsächlich ist Sarumans Zauber nicht mächtiger als eine polynomialzeit-beschränkte randomisierte Turingmaschine. Dennoch will er den Einen Ring stehlen und gibt deshalb vor, Merlins Geheimnis zu kennen.

Wenn Saruman den linken oder den rechten Eingang ins Labyrinth wählt, kennt er König Arthurs Herausforderung noch nicht. Er kann also lediglich eine Münze werfen; sagen wir, Kopf für links und Zahl für rechts. Mit einem glücklichen Wurf wählt er den richtigen Eingang, der Arthurs darauf folgender Forderung entspricht, und ist erfolgreich. In diesem Fall fällt Arthur auf ihn herein. Ist Sarumans Münzwurf jedoch unglücklich, so kann er Arthur nicht hereinlegen. Denn wenn Saruman das Labyrinth zum Beispiel durch den rechten Eingang betritt, Arthur aber den Heiligen Gral zu sehen verlangt, dann hat Saruman verloren. Er kommt am Drachen nicht vorbei, da er Merlins geheimen Spruch nicht kennt und so den Drachen nicht in Schlaf versetzen kann.

Natürlich ist bei nur einem Test Sarumans Erfolgswahrscheinlichkeit mit $1/2$ noch viel zu hoch. Durch wiederholte Herausforderungen an ihn wird Arthur den Betrugsversuch aber sehr wahrscheinlich entdecken und den Betrüger entlarven. Sobald es Saruman auch nur einmal nicht gelingt, den verlangten Heiligen Gral oder Einen Ring zu präsentieren, weiß Arthur sicher, dass er nicht Merlin ist.

Die in dieser Geschichte geschilderte Authentifikationsprozedur hat noch nichts mit Zero-Knowledge zu tun; stattdessen handelt es sich hier um ein ganz gewöhnliches „Challenge-and-Response“-Authentifikationsprotokoll. Zur Illustration der nachfolgenden formalen Definition des Begriffs *Zero-Knowledge* wird als ein konkretes Beispiel das Zero-Knowledge-Protokoll von Goldreich, Micali und Wigderson angegeben. Dieses ist ein Protokoll für GI, das Graphisomorphie-Problem. Zur Erinnerung: In Lemma 6.44 wurde gezeigt, dass GI in $\text{NP} \cap \text{coAM}$ liegt.

Beispiel 8.23 (Zero-Knowledge-Protokoll für Graphisomorphie). Wie in Story 8.22 möchte sich Merlin authentifizieren, indem er nachweist, dass er sein Geheimnis

Schritt	Merlin	Saruman	Arthur
1	wählt zufällig einen großen Graphen G_0 mit n Knoten und eine Permutation $\pi \in \mathfrak{S}_n$ und berechnet $G_1 = \pi(G_0)$; (G_0, G_1) ist öffentlich und π ist geheim		
2		$(G_0, G_1) \Rightarrow$	
3	wählt zufällig eine Permutation $\mu \in \mathfrak{S}_n$ und ein Bit $m \in \{0, 1\}$ und berechnet $H = \mu(G_m)$		
4		$H \Rightarrow$	
5			wählt zufällig ein Bit $a \in \{0, 1\}$ und verlangt einen Isomorphismus aus $\text{Iso}(G_a, H)$
6		$\Leftarrow a$	
7	berechnet $\alpha \in \text{Iso}(G_a, H)$ durch: $\alpha = \mu$, falls $a = m$; $\alpha = \pi\mu$, falls $0 = a \neq m = 1$; $\alpha = \pi^{-1}\mu$, falls $1 = a \neq m = 0$		
8		$\alpha \Rightarrow$	
9			verifiziert $\alpha(G_a) = H$ und akzeptiert entsprechend

Abb. 8.8. Zero-Knowledge–Protokoll für Graphisomorphie

kennt. Merlin hat die Kraft einer NP-Maschine, und sein Geheimnis ist ein Isomorphismus zwischen zwei großen isomorphen Graphen. Man weiß nicht, ob GI in Polynomialzeit gelöst werden kann, und nicht einmal, ob GI mit randomisierten Algorithmen in Polynomialzeit gelöst werden kann. Deshalb dürfen wir annehmen, dass weder Arthur noch der betrügerische Zauberer Saruman, deren Berechnungskraft ja einem randomisierten Polynomialzeit-Algorithmus entspricht, in der Lage sind, Merlins Geheimnis zu enthüllen.

Merlin kann sein Geheimnis leicht erwerben bzw. erzeugen und braucht dafür nicht einmal seine volle NP-Kraft. Erst wählt er zufällig einen großen Graphen G_0 mit n Knoten und eine Permutation $\pi \in \mathfrak{S}_n$. Dann berechnet er den Graphen $G_1 = \pi(G_0)$. Natürlich sind G_0 und G_1 isomorphe Graphen, und π ist ein Isomorphismus zwischen ihnen. Merlin stellt das Paar (G_0, G_1) in einem öffentlichen Verzeichnis zur Verfügung und hält den Isomorphismus $\pi \in \text{Iso}(G_0, G_1)$ geheim.

Angenommen, Gandalf, eine *Trusted Third Party*, zertifiziert, dass (G_0, G_1) wirklich von Merlin erzeugt wurde. Abbildung 8.8 präsentiert ein *Challenge-and-Response*-Authentifikationsprotokoll zwischen Arthur und Merlin. Wir werden später sehen, dass dieses Protokoll außerdem die Zero-Knowledge-Eigenschaft besitzt.

Natürlich kann Merlin nicht einfach π an Arthur schicken, denn dann hätte er sein Geheimnis verraten. Um zu beweisen, dass G_0 und G_1 tatsächlich isomorph sind, wählt Merlin stattdessen eine zufällige Permutation $\mu \in \mathfrak{S}_n$ und ein zufälliges

Bit $m \in \{0, 1\}$ unter Gleichverteilung und berechnet den Graphen $H = \mu(G_m)$. Das heißt, m bestimmt, welcher der Graphen G_0 oder G_1 mittels μ zu permutieren ist, um H zu erzeugen, einen zu G_m isomorphen Graphen. Merlin schickt H an Arthur, der unter Gleichverteilung ein Zufallsbit $a \in \{0, 1\}$ wählt. Arthur schickt dann a als seine Herausforderung an Merlin und verlangt von Merlin, mit einem Isomorphismus α zwischen G_a und H zu antworten. Arthur akzeptiert Merlins Antwort α genau dann, wenn $\alpha(G_a) = H$ gilt.

Dieses Protokoll funktioniert, weil Merlin seinen geheimen Isomorphismus $\pi \in \text{Iso}(G_0, G_1)$, sein Zufallsbit $m \in \{0, 1\}$ und seine Zufallspermutation $\mu \in \text{Iso}(G_m, H)$ kennt. Er kann somit leicht einen Isomorphismus $\alpha \in \text{Iso}(G_a, H)$ bestimmen, mit dem er sich authentifizieren kann; siehe Abbildung 8.8 und Übung 8.10. Weil G_0 und G_1 isomorph sind, akzeptiert Arthur mit Wahrscheinlichkeit eins.²

Führt hingegen Saruman, als Merlin verkleidet, das Protokoll aus Abbildung 8.8 mit Arthur aus, so kennt Saruman zwar das öffentliche Paar (G_0, G_1) isomorpher Graphen, aber nicht Merlins Isomorphismus π , der während des Protokolls geheim gehalten wird. Dennoch möchte er sich als Merlin ausgeben. Er wählt also zufällig eine Permutation $\sigma \in \mathfrak{S}_n$ und sein Bit $s \in \{0, 1\}$ und berechnet den Graphen $H_S = \sigma(G_s)$. Dann schickt Saruman H_S an Arthur und erhält dessen Herausforderung a . Wenn er Glück hat und $s = a$ gilt, dann gewinnt Saruman. Gilt jedoch $s \neq a$, dann würde die Berechnung von $\alpha = \pi\sigma$ oder $\alpha = \pi^{-1}\sigma$ offenbar Kenntnis von π erfordern. Weil das Berechnen eines Isomorphismus aber selbst für randomisierte Polynomialzeit-Algorithmen zu schwer ist, kann Saruman π nicht ermitteln, sofern die Graphen G_0 und G_1 nur groß genug gewählt sind.

Ohne π zu kennen, kann er nur raten. Seine Chancen, zufällig ein Bit s mit $s = a$ zu erwischen, sind höchstens $1/2$. Natürlich kann Saruman immer raten, und somit ist seine Erfolgswahrscheinlichkeit genau $1/2$. Fordert Arthur ihn oft genug heraus, etwa in k unabhängigen Runden dieses Protokolls, dann ist die Betrugswahrscheinlichkeit mit 2^{-k} verschwindend gering. Bereits für $k = 20$ kann diese Wahrscheinlichkeit vernachlässigt werden: Sarumans Erfolgswahrscheinlichkeit ist dann kleiner als eins zu einer Million.

Zero-Knowledge-Protokolle haben, anders als andere *Challenge-and-Response*-Authentifikationsprotokolle, eine wichtige zusätzliche Eigenschaft: Man kann nämlich (unter geeigneten Annahmen) mathematisch streng beweisen, dass das Protokoll keinerlei Information über das Geheimnis durchsickern lässt. Anders gesagt, auch wenn der Verifizierer Arthur schließlich überzeugt sein wird, dass der Beweiser Merlin sein Geheimnis kennt, können weder Arthur noch ein Lauscher irgendetwas über Merlins Geheimnis herausfinden.

Die Zero-Knowledge-Eigenschaft wird nun formal definiert. Nützlich dafür ist der Begriff der Arthur-Merlin-Spiele und der mit AMH bezeichneten Arthur-Merlin-

² Der Fall nicht isomorpher Graphen tritt in diesem Protokoll nicht auf. Das Protokoll aus Abbildung 8.8 kann jedoch so modifiziert werden, dass Arthur und Merlin das Problem GI entscheiden, vergleiche Lemma 6.44. Das Paar (G_0, G_1) ist dann ihre Eingabe und wird nicht von Merlin gewählt. Sind G_0 und G_1 nicht isomorph, so lehnt Arthur Merlins falschen Beweis mit Wahrscheinlichkeit $1/2$ ab.

Hierarchie aus Abschnitt 6.3. Im Zusammenhang mit der kryptographischen Authentikation nennt man ein Arthur-Merlin-Spiel auch Arthur-Merlin-Protokoll.

Definition 8.24 (Zero-Knowledge-Protokoll). Sei (M, A) ein Arthur-Merlin-Protokoll, das eine Menge $L \in \text{AMH}$ akzeptiert, wobei M eine NP-Maschine und A eine polynomialezeit-beschränkte randomisierte Turingmaschine ist. (M, A) heißt Zero-Knowledge-Protokoll für L , falls es eine polynomialezeit-beschränkte randomisierte Turingmaschine S gibt, die als der Simulator bezeichnet wird, so dass gilt:

- (S, A) simuliert das Originalprotokoll (M, A) , und
- für jedes $x \in L$ sind die Tupel (m_1, m_2, \dots, m_k) und (s_1, s_2, \dots, s_k) , die die in (M, A) bzw. in (S, A) ausgetauschte Information beschreiben, über die Münzwürfe in (M, A) bzw. in (S, A) identisch verteilt.

In Definition 8.24 ist zu berücksichtigen, dass Merlin, kraft seines Nichtdeterminismus, auch in der Lage ist, Zufallswahlen zu treffen oder „Münzen zu werfen“. Eine weitere Anmerkung ist, dass der oben definierte Begriff in der Literatur als „*honest-verifier perfect zero-knowledge*“ bezeichnet wird. Das heißt, (a) es wird angenommen, dass der Verifizierer Arthur *ehrlich* ist und nicht versucht, unerlaubte Änderungen am Protokoll vorzunehmen oder anderweitig von ihm abzuweichen, um es zu seinem Vorteil abzuändern, und (b) die Definition verlangt, dass die im simulierten Protokoll übertragene Information *perfekt* mit der im Originalprotokoll übertragenen Information übereinstimmt. Die Annahme (a) scheint für wirkliche kryptographische Anwendungen etwas zu idealistisch zu sein. Die Forderung (b) ist hingegen vielleicht etwas zu strikt; mit schwächeren Anforderungen könnte man ebenso gut auskommen. Deshalb betrachtet man auch andere Varianten von Zero-Knowledge, die weniger restriktiv sind; siehe die Bemerkungen in Abschnitt 8.8.

Story 8.22 wird nun weitererzählt, und man wird sehen, dass Arthur und Merlins *Challenge-and-Response*-Authentifikationsprotokoll tatsächlich die Zero-Knowledge-Eigenschaft hat.

Story 8.25 (Zero-Knowledge-Protokoll – Fortsetzung von Story 8.22)

Nicht mehr als drei Versuche benötigte Arthur, bis er Sarumans Schwundel schließlich aufdecken und seine wahre Identität enthüllen konnte. Den König erzürnte die dreiste Art und Weise, mit der er beinahe hinters Licht geführt worden wäre. Es fehlte nicht viel und er hätte dem um Gnade winselnden Zauberer in seiner Wut den Kopf abgeschlagen. Doch dann beruhigte er sich zum Glück wieder, steckte sein Schwert zurück in die Scheide und beschloss, mit Sarumans Hilfe zu beweisen, dass das Protokoll, das er gewöhnlich mit Merlin auszuführen pflegte, die Zero-Knowledge-Eigenschaft besitzt. Unterwürfig und dankbar schlug Saruman ein.

Also führen Arthur und Saruman erneut mehrere Runden des Protokolls aus. Arthur zeichnet dieses simulierte Protokoll sorgfältig auf. Seine Protokollaufzeichnungen zeigen Saruman, der durch den Haupteingang tritt und die Tür hinter sich schließt, so dass Arthur nicht sieht, ob Saruman das Labyrinth dann durch den linken oder den rechten Eingang betritt. Anschließend zeigen Arthurs Aufzeichnungen seine eigene Herausforderung, mit der er entweder den Heiligen Gral oder den Einen Ring

zu sehen verlangt, und Sarumans Antwort, die entweder erfolgreich oder ein Fehlschlag ist. Hat Saruman Glück und kann die Forderung Arthurs erfüllen, so sieht diese Runde des Protokolls gerade wie eine Runde des Originalprotokolls mit Merlin aus (welcher bei keiner Herausforderung Arthurs versagt). Hat Saruman jedoch Pech und kann Arthurs Forderung nicht erfüllen, dann unterscheidet sich diese Runde ihres Protokolls natürlich vom Originalprotokoll.

Doch wann immer der Zauberer Saruman versagt, wird diese Runde des Protokolls einfach nicht aufgezeichnet, d.h., Arthur vernichtet den entsprechenden Teil des Transkripts. Dieses simulierte Protokoll wurde ja nicht zum Zwecke der Authentikation ausgeführt, sondern um zu überprüfen, ob das Protokoll eine bestimmte Eigenschaft (nämlich Zero-Knowledge) besitzt; das simulierte Protokoll hat also eher den Charakter eines Gedankenexperiments als den einer konkreten Anwendung.

Dann geht Arthur zu Lancelot und zeigt ihm seine Aufzeichnung des simulierten Protokolls sowie eine Aufzeichnung eines früheren Originalprotokolls zwischen ihm und Merlin. Er bittet Lancelot, sich beide Aufzeichnungen anzusehen und ihm dann zu sagen, welches das Originalprotokoll mit Merlin und welches das gefälschte mit Saruman ist. Natürlich kann Lancelot sie nicht auseinander halten; zu sehen sind in den Aufzeichnungen lediglich Arthurs Herausforderungen und des Zauberers erfolgreiche Antworten in jeder Runde, da Sarumans Fehlschläge ja sorgsam gelöscht worden sind.

Aber wenn Saruman (mit Arthurs Hilfe) in der Lage ist, ein gefälschtes Arthur-Merlin-Protokoll zu erstellen, das vom originalen nicht zu unterscheiden ist, so schließt Arthur nun, dann hat das Protokoll keinerlei Information, welcher Art auch immer, über Merlins Geheimnis durchsickern lassen. Saruman kennt ja Merlins Geheimnis nicht. Er konnte also gar keine Information darüber in das simulierte Protokoll einfließen lassen. Und wo nichts drin ist, kann man auch nichts herausholen.

Beispiel 8.23 wird nun fortgesetzt, um zu zeigen, dass das Protokoll von Goldreich, Micali und Wigderson für das Graphisomorphie-Problem die Zero-Knowledge-Eigenschaft hat.

Beispiel 8.26 (Zero-Knowledge-Protokoll für Graphisomorphie – Fortsetzung). Abbildung 8.8 zeigt das Originalprotokoll zwischen Arthur und Merlin. Saruman ist dort nur ein Lauscher, der sich als Merlin auszugeben versucht, und wir haben gesehen, dass seine Chancen, Arthur erfolgreich hinters Licht zu führen, klein sind.

Um nun zu zeigen, dass das Originalprotokoll zwischen Arthur und Merlin aus Abbildung 8.8 die Zero-Knowledge-Eigenschaft hat, wird es durch das in Abbildung 8.9 dargestellte Protokoll simuliert, in welchem Saruman Merlins Stelle einnimmt. Wie oben bereits erwähnt wurde, dient das simulierte Protokoll nicht dem Zweck der Authentikation, sondern wird lediglich für den Nachweis benötigt, dass das Originalprotokoll die Zero-Knowledge-Eigenschaft besitzt.

Im simulierten Protokoll kennt Saruman Merlins geheime Isomorphismus π nicht, tut aber so, als würde er ihn kennen. Nehmen wir an, dass Arthur und Saruman mehrere Runden des Protokolls durchführen, wobei sie stets dasselbe Paar (G_0, G_1) isomorpher Graphen verwenden, das ihnen als Merlins öffentliche Information bekannt ist. Auf die Schritte 1 und 2 des Originalprotokolls kann also hier verzichtet

Schritt	Saruman	Arthur
1 & 2	Merlins Paar (G_0, G_1) isomorpher Graphen ist öffentliche Information	
3	wählt zufällig eine Permutation $\sigma \in \mathfrak{S}_n$ und ein Bit $s \in \{0, 1\}$ und berechnet $H = \sigma(G_s)$	
4		$H \Rightarrow$
5		wählt zufällig ein Bit $a \in \{0, 1\}$ und verlangt einen Isomorphismus aus $\text{Iso}(G_a, H)$
6		$\Leftarrow a$
7	Saruman berechnet $\alpha = \sigma$, falls $a = s$; falls $a \neq s$, wird diese Runde gelöscht	
8		$\alpha \Rightarrow$
9		Arthur muss Sarumans falsche Identität akzeptieren, denn aus $a = s$ folgt $\alpha(G_a) = H$

Abb. 8.9. Simulation des Zero-Knowledge-Protokolls für Graphisomorphie

werden. Die in einer Runde des Protokolls übertragene Information hat die Form eines Tripels, (H, a, α) . Immer dann, wenn Saruman mit Glück ein Zufallsbit s mit $s = a$ erwischt, schickt er einfach $\alpha = \sigma$ an Arthur und gewinnt: Arthur muss ihn als Merlin akzeptieren. Ist andererseits $s \neq a$, so kann Saruman Arthur nicht hereinlegen und scheitert. Doch das ist kein Problem für die beiden. Sie löschen diese Runde einfach vom Protokoll und starten neu. In dieser Weise können sie eine Folge von Tripeln der Form (H, a, α) erzeugen, die von der entsprechenden Tripelfolge im Originalprotokoll ununterscheidbar ist. Es folgt, dass das Protokoll aus Abbildung 8.8 die Zero-Knowledge-Eigenschaft besitzt.

Satz 8.27 (Goldreich, Micali und Wigderson). *Das Protokoll aus Abbildung 8.8 ist ein Zero-Knowledge-Protokoll.*

Zum Abschluss dieses Abschnitts wird das in Abbildung 8.10 dargestellte Zero-Knowledge-Protokoll präsentiert, das 1986 von Fiat und A. Shamir entwickelt wurde. Es verwendet das zahlentheoretische Problem QR, das in Definition 2.43 definiert wurde. Die Sicherheit dieses Protokolls beruht auf der Annahme, dass sowohl das Berechnen von Quadratwurzeln in \mathbb{Z}_n^* als auch das Faktorisieren des Moduls n nicht praktikable Aufgaben sind.

Satz 8.28 (Fiat und Shamir). *Das in Abbildung 8.10 dargestellte Identifikationsschema von Fiat und Shamir ist ein Zero-Knowledge-Protokoll.*

In Übung 8.11 soll gezeigt werden, dass dieses Protokoll korrekt funktioniert, dass sich ein Betrüger (unter vernünftigen Annahmen) nur mit einer Wahrscheinlichkeit von 2^{-k} erfolgreich für jemand anderen ausgeben kann und dass es die Zero-Knowledge-Eigenschaft hat.

Schritt	Merlin	Saruman	Arthur
1	wählt zwei große Primzahlen, p und q , und eine Geheimzahl $s \in \mathbb{Z}_n^*$ und berechnet $n = pq$ und $v = s^2 \bmod n$		
2		$(n, v) \Rightarrow$	
3	wählt zufällig $r \in \mathbb{Z}_n^*$ und berechnet $x = r^2 \bmod n$		
4		$x \Rightarrow$	
5			wählt ein Zufallsbit $a \in \{0, 1\}$
6		$\Leftarrow a$	
7	berechnet $y = r \cdot s^a \bmod n$		
8		$y \Rightarrow$	
9			verifiziert $y^2 \equiv x \cdot v^a \bmod n$ und akzeptiert entsprechend

Abb. 8.10. Zero-Knowledge-Identifikationsschema von Fiat und Shamir

Anmerkung 8.29. Im Gegensatz zu digitalen Signaturen, welche die Echtheit elektronisch übermittelnder *Dokumente* wie E-Mails oder digitaler Verträge beglaubigen, können die in diesem Abschnitt präsentierten Protokolle zur Authentikation von *Individuen* verwendet werden, die in einem Computer-Netzwerk oder als die an einem kryptographischen Protokoll beteiligten Parteien mitwirken. Wie in Abschnitt 4.1 erwähnt sind unter „Individuen“ hier nicht unbedingt Menschen zu verstehen, sondern der Begriff ist weiter gefasst und schließt beispielsweise Computer ein, die automatisch in einem Netzwerk Protokolle mit anderen Computern ausführen.

8.5 Das Public-Key-Kryptosystem von Merkle und Hellman

In diesem Abschnitt wird ein Public-Key-Kryptosystem vorgestellt, das Merkle und Hellman 1978 vorschlugen. Obwohl es in den frühen 1980ern von Shamir gebrochen wurde,³ ist es dennoch lohnend, sich damit zu befassen, denn es ist einfach und elegant und zur Illustration der grundlegenden Entwurfsprinzipien der Public-Key-Kryptographie besonders gut geeignet.

Alle bisher betrachteten Public-Key-Kryptosysteme beruhen auf der Idee der *Falltür-Einwegfunktion* (englisch: „trapdoor one-way function“). Zum Beispiel wird beim RSA-Kryptosystem aus Abbildung 7.1 in Abschnitt 7.1 die Tatsache benutzt, dass das modulare Potenzieren mit dem *Square-and-Multiply*-Algorithmus aus Abbildung 7.2 effizient ausgeführt werden kann. Somit sind die Verschlüsselung und die autorisierte Entschlüsselung leicht. Im Gegensatz dazu scheint aber die nicht autorisierte Entschlüsselung durch einen Angreifer schwer zu sein, da sowohl das Berechnen von m aus e, n und $c = m^e \bmod n$ (also das Ziehen der e -ten Wurzel von c)

³ Es gibt jedoch auch Varianten dieses Kryptosystems, die bis heute ungebrochen sind, siehe die Anmerkungen in Abschnitt 8.8.

modulo n) als auch das Faktorisieren des RSA-Moduls n als schwere Berechnungsprobleme gelten.

Für die effiziente autorisierte Entschlüsselung ist es wichtig, dass Bob die Primfaktoren p und q des öffentlichen RSA-Moduls n kennt, wodurch er seinen privaten Entschlüsselungsschlüssel leicht bestimmen kann. Anders gesagt hat Bob eine *Falltür-Information*, die ihm gegenüber einem Lauscher, welchem diese Information fehlt, einen Vorteil verschafft.

Ein anderes Beispiel ist das ElGamal-Kryptosystem aus Abbildung 8.3 in Abschnitt 8.2. Wieder sind Verschlüsselung und autorisierte Entschlüsselung leicht, wobei letztere Bobs privaten Schlüssel b als Falltür-Information benutzt. Im Gegensatz dazu scheint die nicht autorisierte Entschlüsselung schwer zu sein, weil das Berechnen diskreter Logarithmen als praktisch nicht machbar gilt. Sowohl das Berechnen diskreter Logarithmen als auch das Ziehen von Wurzeln modulo einer Zahl kann als Umkehrfunktion der modularen Potenzfunktion betrachtet werden. Der Unterschied zwischen diesen beiden Umkehrfunktionen ist, dass das Wurzelziehen für $\alpha = \beta^a \bmod n$ das Berechnen der Basis β aus α , a und n bedeutet, wohingegen beim diskreten Logarithmus von α der Exponent a für gegebene α , β und n berechnet wird.

Das Merkle–Hellman-Kryptosystem basiert ebenfalls auf der Idee der Falltür-Einwegfunktion. Seine Sicherheit beruht nämlich auf der Härte des *Subset-of-Sums*-Problems, das hier mit SOS bezeichnet wird und formal in Definition 3.66 von Abschnitt 3.5.3 eingeführt wurde. SOS ist eine eingeschränkte Variante des Rucksack-Problems, und wir haben in Satz 3.67 gesehen, dass SOS ein NP-vollständiges Problem ist. Daher ist kein deterministischer oder randomisierter Polynomialzeit-Algorithmus zur Lösung dieses Problems bekannt. Die hier verwendete Falltür-Information macht sich die Tatsache zunutze, dass bestimmte Instanzen von SOS trotz der NP-Härte des allgemeinen Problems leicht zu lösen sind.

Definition 8.30. Sei $\langle \mathbf{s}, T \rangle$ eine gegebene SOS-Instanz, d.h., $\mathbf{s} = (s_1, s_2, \dots, s_n)$ ist eine Folge positiver natürlicher Zahlen (die als Größen bezeichnet werden) und T ist eine positive natürliche Zahl (die Zielsumme). Eine Folge \mathbf{s} von Größen heißt superwachsend, falls für jedes i mit $2 \leq i \leq n$ gilt:

$$s_i > \sum_{j=1}^{i-1} s_j.$$

Behauptung 8.31 sagt, dass das *Subset-of-Sums*-Problem effizient lösbar ist, falls die gegebene Folge von Größen superwachsend ist. Der einfache Beweis von Behauptung 8.31 wird dem Leser als Übung 8.12(a) überlassen.

Behauptung 8.31. Für Instanzen $\langle \mathbf{s}, T \rangle$ mit einer superwachsenden Folge \mathbf{s} von Größen kann das Problem SOS in deterministischer Polynomialzeit gelöst werden.

Abbildung 8.11 zeigt das Merkle–Hellman-System. Damit die nicht autorisierte Entschlüsselung noch schwerer wird, kann man zusätzlich eine Zufallspermutation auf den Vektor \mathbf{t} anwenden. Dieses Public-Key-Kryptosystem macht von Behauptung 8.31 Gebrauch, wie im Folgenden erklärt wird.

Schritt	Alice	Erich	Bob
1		wählt eine superwachsende Folge von Größen, $\mathbf{s} = (s_1, s_2, \dots, s_n)$, eine Primzahl $p > \sum_{i=1}^n s_i$ und einen Faktor $b \in \mathbb{Z}_p$ und berechnet den Vektor $\mathbf{t} = (t_1, t_2, \dots, t_n)$ mit der linearen modularen Transformation $t_i = b s_i \bmod p;$ \mathbf{t} ist öffentlich und \mathbf{s}, p und b sind privat	
2		$\Leftarrow \mathbf{t}$	
3	verschlüsselt die Nachricht $\mathbf{m} = (m_1, m_2, \dots, m_n)$ als $c = \sum_{i=1}^n m_i t_i$		
4		$c \Rightarrow$	
5			entschlüsselt c durch Lösen der SOS-Instanz $\langle \mathbf{s}, T \rangle$, wobei $T = b^{-1} c \bmod p$ gesetzt wird

Abb. 8.11. Public-Key-Kryptosystem von Merkle und Hellman

Schritt 1: Schlüsselerzeugung. Der legitime Empfänger Bob wählt eine superwachsende Folge \mathbf{s} von Größen, eine Primzahl p und einen Faktor $b \in \mathbb{Z}_p$. Diese Werte stellen seinen privaten Schlüssel dar, also seine Falltür-Information. Er bestimmt dann seinen öffentlichen Schlüssel, nämlich einen neuen Vektor $\mathbf{t} = (t_1, t_2, \dots, t_n)$, der durch die folgende lineare modulare Transformation erhalten wird:

$$t_i = b s_i \bmod p.$$

Schritt 2: Kommunikation. Bobs öffentlicher Schlüssel \mathbf{t} ist Alice nun bekannt.

Schritt 3: Verschlüsselung. Ist $\mathbf{m} = (m_1, m_2, \dots, m_n) \in \{0, 1\}^n$ die Nachricht, die Alice verschlüsseln möchte, so berechnet sie

$$c = \sum_{i=1}^n m_i t_i.$$

Schritt 4: Kommunikation. Alice schickt den Schlüsseltext c an Bob.

Schritt 5: Entschlüsselung. Wenn Bob den Schlüsseltext c erhält, wobei $0 \leq c \leq n(p-1)$ gilt, dann verwendet er seine Falltür-Information, um die zuvor angewandte lineare modulare Transformation zu invertieren. Insbesondere berechnet er die Zielsumme

$$T = b^{-1} c \bmod p,$$

die zur Wiederherstellung der ursprünglichen Nachricht benötigt wird. Weil \mathbf{s} eine superwachsende Folge von Größen ist, kann Bob nun die SOS-Instanz $\langle \mathbf{s}, T \rangle$ mit dem Algorithmus aus Behauptung 8.31 effizient zu lösen. Somit kann er Alice' Nachricht leicht entschlüsseln.

Andererseits scheint die nicht autorisierte Entschlüsselung schwer zu sein. Ein Lauscher (dem Bobs Falltür-Information nicht zur Verfügung steht und der daher nicht weiß, wie die lineare modulare Transformation invertiert werden kann) steht einer Instanz des allgemeinen S0S-Problems gegenüber, also einem NP-vollständigen Problem. Allerdings gibt es, wie bereits erwähnt, dennoch Möglichkeiten, das Merkle–Hellman-System zu brechen, siehe Problem 8.2.

Beispiel 8.32 (Merkle–Hellman-Kryptosystem). Angenommen, Bob wählt die superwachsende Folge

$$\mathbf{s} = (2, 3, 6, 12, 25, 51, 101, 203, 415)$$

mit $n = 9$ Größen, die Primzahl $p = 821$, für die $p > \sum_{i=1}^n s_i$ gilt, und den Faktor $b = 444$. Dann berechnet er seinen öffentlichen Schlüssel

$$\mathbf{t} = (67, 511, 201, 402, 427, 477, 510, 643, 356),$$

der aus einer nicht superwachsenden Folge besteht. Sei $\mathbf{m} = (1, 0, 1, 1, 0, 1, 0, 0, 1)$ die Nachricht, die Alice verschlüsseln möchte. Da sie \mathbf{t} kennt, kann sie den Schlüsseltext

$$c = 67 + 201 + 402 + 427 + 477 + 356 = 1503$$

berechnen und c an Bob schicken. Um c zu entschlüsseln, berechnet dieser zunächst mit dem erweiterten Euklidischen Algorithmus aus Abbildung 2.2 das Inverse von 444 modulo 821, nämlich $b^{-1} = 723$. Dann berechnet er die Zielsumme

$$T = 723 \cdot 1503 \bmod 821 = 486.$$

Schließlich stellt Bob die ursprüngliche Nachricht wieder her, indem er mit dem Algorithmus aus Behauptung 8.31 (siehe Übung 8.12(a)), die S0S-Instanz $\langle \mathbf{s}, T \rangle$ löst. Da seine private Folge \mathbf{s} von Größen superwachsend ist, gelingt ihm dies effizient.

8.6 Die Protokolle von Rabi, Rivest und Sherman

Abbildung 8.12 präsentiert ein weiteres Protokoll für den Schlüsseltausch, welches 1984 von Rivest und Sherman vorgeschlagen wurde. Es kann so modifiziert werden, dass man ein Protokoll für digitale Signaturen erhält, siehe Übung 8.13(a). Dieses digitale Signatur-Schema wurde von Rabi und Sherman [RS97] entwickelt. Die Unterschiede zwischen den Schlüsseltausch-Protokollen von Diffie und Hellman bzw. von Rivest und Sherman werden in Abschnitt 8.8 diskutiert.

Das Rivest–Sherman-Protokoll in Abbildung 8.12 verwendet eine „stark nichtinvertierbare, assoziative Einwegfunktion“, σ . Bevor formale Definitionen dieser Eigenschaften bereitgestellt werden, soll kurz die intuitive Idee erklärt und das verwendete Komplexitätsmodell diskutiert werden. Eine *Einwegfunktion* ist eine ehrliche und leicht berechenbare Funktion, die aber schwer invertierbar ist. Um den Begriff der „Nichtinvertierbarkeit“ zu erfassen, wurden bisher verschiedene Modelle

Schritt	Alice	Erich	Bob
1	wählt zufällig zwei große Wörter, x und y , hält x geheim und berechnet $x\sigma y$		
2		$\langle y, x\sigma y \rangle \Rightarrow$	
3			wählt zufällig ein großes Wort z , hält z geheim und berechnet $y\sigma z$
4		$\Leftarrow y\sigma z$	
5	berechnet ihren Schlüssel $k_A = x\sigma(y\sigma z)$		berechnet seinen Schlüssel $k_B = (x\sigma y)\sigma z$

Abb. 8.12. Rivest–Sherman-Protokoll für den Schlüsseltausch

vorgeschlagen.⁴ Je nach dem gewählten Modell gibt es verschiedene Kandidaten für Einwegfunktionen, von denen einige in Abschnitt 8.5 erwähnt wurden.

Definition 3.78 in Abschnitt 3.6.2 führte den Begriff der *komplexitätstheoretischen Einwegfunktion im Worst-Case-Modell* ein, und in diesem Abschnitt konzentrieren wir uns ausschließlich auf diesen Begriff der Einwegigkeit. Das Average-Case-Modell ist sowohl in kryptographischen Anwendungen als auch in der Komplexitätstheorie sehr wichtig; beispielsweise in Bezug auf die Derandomisierung durch Pseudozufallsgeneratoren. Jedoch bleibt der Anspruch, einen Beweis für die Existenz von Einwegfunktionen zu finden, vorerst ein offenes Problem, selbst im „weniger anspruchsvollen“ Worst-Case-Modell. Daher ist es sinnvoll, zu fragen, unter welchen komplexitätstheoretischen Annahmen es bestimmte Typen von Komplexitätstheoretischen Einwegfunktionen gibt, gewissermaßen als ein erster, bescheidener Schritt in Richtung der Entwicklung einer Einwegfunktion im anspruchsvollerem Average-Case-Modell unter geeigneten Annahmen.

Eigenschaften wie die Assoziativität sind nur für Funktionen mit zwei Argumenten sinnvoll; betrachten wir also solche Funktionen, die von $\Sigma^* \times \Sigma^*$ in Σ^* abbilden.⁵ Weiter nehmen wir an, dass Einwegfunktionen partiell sein können und nicht injektiv sein müssen. Das heißt, es wird nicht verlangt, dass sie totale (also überall definierte) Funktionen auf $\Sigma^* \times \Sigma^*$ sein müssen, und sie bilden womöglich verschiedene Urbilder auf denselben Funktionswert ab. Für zweistellige Funktionen verwenden wir die Infix-Notation (z.B. $x\sigma y$) statt der Präfix-Notation (z.B. $\sigma(x,y)$). Informal gesagt heißt eine zweistellige Funktion σ stark nichtinvertierbar, falls es für einen gegebenen Funktionswert und eine der beiden Komponenten des entsprechenden Arguments schwer ist, eine zugehörige andere Komponente dieses Arguments zu bestimmen. Diese Eigenschaft verlangt man von der Funktion σ in Abbildung 8.12, um den offensichtlichsten direkten Angriff zu verhindern: Wäre σ nicht stark nichtinvertierbar, dann könnte ein Angreifer x und y ausrechnen, was z sein muss, um $x\sigma y$ zu erhalten.

⁴ In diesem Abschnitt bezieht sich „Nichtinvertierbarkeit“ auf Nicht-FP-Invertierbarkeit, siehe Definition 3.78.

⁵ Die Wörter x , y und z in Abbildung 8.12 können ebenso als binär dargestellte natürliche Zahlen aufgefasst werden.

tierbar, so könnte ein Lauscher leicht die geheimen Wörter x und z (und somit auch die geheimen Schlüssel k_A und k_B) aus den übertragenen Werten y , $x\sigma y$ und $y\sigma z$ ermitteln.

Der in Definition 3.78 eingeführte Begriff der Ehrlichkeit einer Funktion wird benötigt, um zu verhindern, dass sie trivialerweise nichtinvertierbar ist. Das heißt, eine unehrliche Funktion, wie etwa $\delta(x) = 0^{\lceil \log \log \max(|x|, 2) \rceil}$, kann ihre Eingabe mehr als nur polynomiell schrumpfen lassen und ist daher zwar nichtinvertierbar, aber nur aus dem trivialen Grunde, dass schon das Hinschreiben der Ausgabe ihrer Umkehrfunktion – gemessen in der Länge der Eingabe derselben, also in der Länge des mehr als polynomiell geschrumpften Wertes der unehrlichen Funktion – mehr als Polynomialzeit braucht. Dieser künstliche Längentrick ist bei der Nichtinvertierbarkeit in der Kryptographie oder Komplexitätstheorie jedoch eigentlich nicht hilfreich und wird deshalb in der Definition von Einwegfunktionen durch die Forderung der Ehrlichkeit ausgeschlossen.

Die formale Definition der starken Nichtinvertierbarkeit erfordert eine geeignete Variante der Ehrlichkeit, die hier mit *s-Ehrlichkeit* bezeichnet wird. Intuitiv nimmt ein starker Invertierer für eine Funktion σ als Eingabe einen Funktionswert von σ und eine Komponente eines zu diesem Funktionswert gehörigen Arguments von σ und berechnet eine zugehörige andere Komponente dieses Arguments. Um also σ vor einer trivialen Nichtinvertierbarkeit zu bewahren, verlangt die s-Ehrlichkeit, dass σ keine Komponente eines zu diesem Funktionswert gehörigen Arguments mehr als polynomiell bezüglich der Länge des Funktionswerts *und* der Länge einer zugehörigen anderen Komponente dieses Arguments schrumpfen kann. Man kann sich leicht überlegen, dass es unehrliche Funktionen gibt, die s-ehrlich sind, und dass es ehrliche Funktionen gibt, die nicht s-ehrlich sind; siehe Übung 8.13(b).

Definition 8.33 (Starke Einwegfunktion). Sei $\sigma : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ eine partielle Funktion. D_σ bezeichnet den Definitionsbereich von σ und R_σ den Wertebereich von σ .

1. Wir sagen, σ ist s-ehrlich, falls es ein Polynom p gibt, so dass sowohl (a) als auch (b) gilt:
 - (a) Für alle $x, z \in \Sigma^*$ mit $x\sigma y = z$ für ein $y \in \Sigma^*$ gibt es ein Wort $\tilde{y} \in \Sigma^*$, so dass $x\sigma \tilde{y} = z$ und $|\tilde{y}| \leq p(|x| + |z|)$ gilt.
 - (b) Für alle $y, z \in \Sigma^*$ mit $x\sigma y = z$ für ein $x \in \Sigma^*$ gibt es ein Wort $\tilde{x} \in \Sigma^*$, so dass $\tilde{x}\sigma y = z$ und $|\tilde{x}| \leq p(|y| + |z|)$ gilt.
2. Wir sagen, σ ist (in Polynomialzeit) invertierbar bezüglich der ersten Komponente eines Arguments, falls es einen Invertierer $g_1 \in \text{FP}$ gibt, so dass

$$x\sigma g_1(\langle x, z \rangle) = z$$

für jedes $z \in R_\sigma$ und für alle $x, y \in \Sigma^*$ mit $(x, y) \in D_\sigma$ und $x\sigma y = z$ gilt.

3. Wir sagen, σ ist (in Polynomialzeit) invertierbar bezüglich der zweiten Komponente eines Arguments, falls es einen Invertierer $g_2 \in \text{FP}$ gibt, so dass

$$g_2(\langle y, z \rangle)\sigma y = z$$

für jedes $z \in R_\sigma$ und für alle $x, y \in \Sigma^*$ mit $(x, y) \in D_\sigma$ und $x\sigma y = z$ gilt.

4. Wir sagen, σ ist stark nichtinvertierbar, falls σ weder bezüglich der ersten Komponente eines Arguments noch bezüglich der zweiten Komponente eines Arguments invertierbar ist.
5. Wir nennen σ stark, falls σ in Polynomialzeit berechenbar, s-ehrlich und stark nichtinvertierbar ist.

Nehmen wir für den Augenblick einmal an, dass die Funktion σ aus Abbildung 8.12 total ist. Dann bedeutet die Assoziativität von σ , dass $(x\sigma y)\sigma z = x\sigma(y\sigma z)$ für alle $x, y, z \in \Sigma^*$ gilt. Diese Eigenschaft garantiert, dass Alice und Bob tatsächlich denselben Schlüssel berechnen:

$$k_A = x\sigma(y\sigma z) = (x\sigma y)\sigma z = k_B.$$

Dank der Assoziativität von σ funktioniert das Rivest–Sherman-Protokoll also.

Die oben angegebene Definition der Assoziativität ist für totale Funktionen sinnvoll, aber sie ist nicht sinnvoll für zweistellige Funktionen, die nicht total sind. Zur Veranschaulichung wollen wir einmal versuchen, mit der folgenden Definition den Begriff der Assoziativität auf nicht-totale Funktionen auszudehnen: σ ist *schwach assoziativ*, falls

$$(x\sigma y)\sigma z = x\sigma(y\sigma z) \quad (8.16)$$

für alle $x, y, z \in \Sigma^*$ gilt, für die die Werte $x\sigma y$, $y\sigma z$, $(x\sigma y)\sigma z$ und $x\sigma(y\sigma z)$ jeweils definiert sind. Dieser Versuch einer Definition scheitert jedoch für nicht-totale Funktionen. Was stimmt nicht damit? Betrachte beispielsweise eine Funktion σ mit $0\sigma 1 = 0$ und $1\sigma 0 = 1$, aber für das Paar $(0, 0)$ ist σ nicht definiert. Dann ist $0\sigma(1\sigma 0) = 0\sigma 1 = 0$, aber für $((0\sigma 1), 0) = (0, 0)$ ist σ nicht definiert. Somit hat (8.16) die Form „undefined = 0“. Die Definition der schwachen Assoziativität scheitert in diesem Beispiel also daran, dass sie (8.16) für die Werte $x = 0$, $y = 1$ und $z = 0$ nicht als falsch bewertet, denn eine „Gleichheit“ zwischen einem nicht definierten und einem definierten Wert ist nicht „falsch“, sondern selbst einfach „undefined“.

Definiert man also den Begriff der Assoziativität für partielle Funktionen (zu denen sowohl die totalen als auch die nicht-totalen Funktionen gehören), so scheint es natürlicher zu sein, zu verlangen, dass beide Seiten von (8.16) gemeinsam stehen oder fallen. Das heißt, entweder sollten beide Seiten von (8.16) definiert und gleich sein oder sie sollten beide undefined sein. Diese Beobachtung hat mit Kleenes sorgfältiger Unterscheidung von „vollständiger Gleichheit“ und „schwacher Gleichheit“ partieller Funktionen zu tun [Kle52, pp. 327–328]. Dieses natürliche Verhalten kann mit der folgenden Definition von Assoziativität und Kommutativität erzielt werden, die beide auf der „vollständigen Gleichheit“ im Sinne von Kleene beruhen. Die Kommutativität wird benötigt, damit das Rivest–Sherman-Protokoll aus Abbildung 8.12 auch für mehr als zwei Parteien funktioniert, siehe Übung 8.13(c).

Definition 8.34 (Assoziativität). Sei $\sigma : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ eine partielle Funktion. Sei \perp ein Symbol, das in der Verwendung „ $x\sigma y = \perp$ “ anzeigt, dass σ für (x, y) undefined ist. Sei $\Gamma = \Sigma^* \cup \{\perp\}$ eine Erweiterung von Σ^* . Definiere eine Fortsetzung $\hat{\sigma} : \Gamma \times \Gamma \rightarrow \Gamma$ von σ durch

$$x\widehat{\sigma}y = \begin{cases} x\sigma y & \text{falls } x \neq \perp \neq y \text{ und } (x,y) \in D_\sigma \\ \perp & \text{sonst.} \end{cases} \quad (8.17)$$

Wir sagen, σ ist assoziativ, falls für alle $x, y, z \in \Sigma^*$ gilt:

$$(x\widehat{\sigma}y)\widehat{\sigma}z = x\widehat{\sigma}(y\widehat{\sigma}z).$$

Wir sagen, σ ist kommutativ, falls für alle $x, y \in \Sigma^*$ gilt:

$$x\widehat{\sigma}y = y\widehat{\sigma}x.$$

Nun, da alle technischen Voraussetzungen bereitgestellt sind, wenden wir uns einer Frage zu, die sich im Zusammenhang mit dem Protokoll aus Abbildung 8.12 ganz offensichtlich aufdrängt: Existiert denn überhaupt irgendeine totale Einwegfunktion σ , die stark und assoziativ ist? Eine absolute Antwort auf diese Frage ist nicht bekannt. Schließlich ist ja nicht einmal bekannt, ob normale Einwegfunktionen existieren oder nicht, bei denen man auf all die netten zusätzlichen Eigenschaften, die von σ verlangt werden, verzichtet. Wir werden nun aber sehen, dass solch eine Funktion σ mit all den gewünschten Eigenschaften genau dann existiert, wenn eine normale Einwegfunktion existiert. Das heißt, man kann σ aus einer beliebigen Komplexitätstheoretischen Einwegfunktion konstruieren. Zunächst sei an die ersten beiden Aussagen von Satz 3.82 erinnert: Injektive Einwegfunktionen existieren genau dann, wenn $P \neq UP$ gilt. Wie in Behauptung 8.35 unten festgestellt wird, kann in analoger Weise die Existenz beliebiger Einwegfunktionen (die „many-to-one“ sein dürfen, also nicht unbedingt injektiv sein müssen) durch die Bedingung $P \neq NP$ charakterisiert werden. Im Rest dieses Abschnitts wird auf die explizite Angabe des Ausdrucks „many-to-one“ verzichtet, d.h., wenn von einer Einwegfunktion die Rede ist, ist stets eine Einwegfunktion gemeint, die many-to-one sein darf.

Behauptung 8.35. Einwegfunktionen existieren genau dann, wenn $P \neq NP$ gilt.

Nun wird der Typ einer Einwegfunktion charakterisiert, der für das Rivest–Sherman–Protokoll benötigt wird.

Satz 8.36. Totale, starke, kommutative und assoziative Einwegfunktionen existieren genau dann, wenn $P \neq NP$ gilt.

Beweis. Nach Behauptung 8.35 folgt aus der Existenz totaler, starker, kommutativer und assoziativer Einwegfunktionen unmittelbar $P \neq NP$. Es bleibt die Umkehrung zu zeigen. Sei also $P \neq NP$ angenommen. Sei L eine Menge in NP mit $L \not\subseteq P$, und sei M eine fixierte NPTM, die L akzeptiert. Beispielsweise könnte man unter der Annahme, dass $P \neq NP$ gilt, eine beliebige NP-vollständige Menge als L wählen. Gemäß Definition 5.23 in Abschnitt 5.2 ist ein Zeuge für „ $x \in L$ “ ein Wort $w \in \Sigma^*$, das einen akzeptierenden Pfad von M bei Eingabe x codiert. Für jedes $x \in L$ ist die Menge der Zeugen für „ $x \in L$ “ definiert durch

$$\text{Wit}_M(x) = \{w \in \Sigma^* \mid w \text{ ist ein Zeuge für } „x \in L“\}.$$

Zum Beispiel sind die Zeugen für die Elemente der NP-vollständigen Menge SOS in Beispiel 5.22 in Abschnitt 5.2 angegeben worden. Offenbar ist $\text{Wit}_M(x)$ genau dann leer, wenn $x \notin L$ gilt.

Als eine rein technische Voraussetzung nehmen wir an, dass für jedes $x \in L$ jeder Zeuge w für „ $x \in L$ “ die Länge $p(|x|)$ für ein echt wachsendes Polynom p hat und dass die Länge von w echt größer als die Länge von x ist, d.h., $|w| = p(|x|) > |x|$. Diese Annahme erlaubt es uns, Eingabewörter in L und ihre Zeugen auseinander zu halten, eine Eigenschaft, die bei unserer Konstruktion einer totalen, starken, kommutativen und assoziativen Einwegfunktion nützlich sein wird.

Die Konstruktion geht in zwei Stufen vor. Zunächst wird gezeigt, wie man eine nicht-totale, starke, kommutative und assoziative Einwegfunktion σ aus L konstruieren kann. Dann wird behauptet, dass σ zu einer totalen Funktion fortgesetzt werden kann, die all die anderen Eigenschaften von σ erbt. Der Beweis dieser Behauptung in der zweiten Stufe der Konstruktion wird dem Leser als Übung 8.13(e) überlassen. Bevor wir nun die formalen Details der ersten Stufe der Konstruktion präsentieren, in der σ von L ausgehend definiert wird, illustrieren wir die Idee in der folgenden kurzen Geschichte.

Story 8.37 *Im Polizeirevier einer kleinen Stadt hat Wachtmeisterin Sigma heute Dienst. Auf ihrem Schreibtisch liegt eine Liste L der üblichen Verdächtigen und es gibt viele Berichte über kürzlich begangene Verbrechen. Einige dieser Berichte enthalten die Beschreibung eines der üblichen Verdächtigen, etwa x , von Liste L . Wachtmeisterin Sigma heftet dem Bericht eine Aktenkopie dieser Beschreibung an, so dass dieser jetzt die Form $\langle x, x \rangle$ hat. Zu einigen Berichten gibt es die Aussage w eines Augenzeugen, der diesen Verdächtigen x am Tatort gesehen hat. Wachtmeisterin Sigma heftet w an x , so dass der Bericht nun die Form $\langle x, w \rangle$ hat. Es gibt auch viele andere Berichte, die weder die Beschreibung eines Verdächtigen noch die Aussage eines Zeugen enthalten.*

Eigentlich ist Wachtmeisterin Sigma für ihre Arbeit überqualifiziert. Vor zwei Jahren schloss sie die Polizeiausbildung mit Auszeichnung ab. Deshalb kann sie leicht die Beschreibung eines Verdächtigen von einer Zeugenaussage unterscheiden.⁶ Das heißt, sie kann sofort sagen, ob ein Bericht die Form $\langle x, x \rangle$ oder $\langle x, w \rangle$ hat. Noch besser ist, dass Wachtmeisterin Sigma leicht die Verlässlichkeit von Zeugen überprüfen kann. Bevor sie einen Bericht zu den Akten legt, verifiziert sie dessen Zeugenaussage mit Hilfe eines Lügendetektors.

Hin und wieder nimmt Wachtmeisterin Sigma zwei Berichte vom Tisch, sagen wir a und b . Sie hält a in ihrer linken Hand und b in ihrer rechten Hand und liest beide gründlich. Dann wählt sie manchmal einen der Berichte aus, entweder a oder b , reicht ihn ihrer Vorgesetzten, Kommissarin Omega, und wirft den anderen weg. Gelegentlich wirft sie auch beide weg. Wie entscheidet Wachtmeisterin Sigma aber, welchen Bericht sie weiterleitet und welchen sie in den Papierkorb wirft?

⁶ Einige ihrer Kollegen im Revier verwechseln allerdings immer Verdächtigenbeschreibungen mit Zeugenaussagen, was zum Teil die geringe Rate der Verbrechensaufklärung in dieser Stadt erklärt.

Hat Bericht a die Form $\langle x, w_1 \rangle$ und Bericht b die Form $\langle x, w_2 \rangle$, so äußern sich beide Zeugen zum selben Verdächtigen x . Um Zeit beim Lesen zu sparen, wählt Wachtmeisterin Sigma den Bericht mit der kürzeren Zeugenaussage aus, gibt diesen an Kommissarin Omega weiter und wirft den anderen weg. Die beiden Zeugenaussagen über x , w_1 und w_2 , können auch gleich lang – und sogar identisch – sein, und wenn das so ist, händigt Wachtmeisterin Sigma ihrer Vorgesetzten irgendeinen der beiden Berichte aus.

Hat einer der Berichte a und b die Form $\langle x, x \rangle$ und der andere die Form $\langle x, w \rangle$ für denselben Verdächtigen x und für die Aussage w eines Zeugen, der x am Tatort gesehen hat, so gibt Wachtmeisterin Sigma den Bericht $\langle x, x \rangle$ an Kommissarin Omega weiter und wirft $\langle x, w \rangle$ etwas gedankenlos in den Papierkorb.⁷

Sind die Berichte a und b jedoch nicht von der oben beschriebenen Form, so werden sie beide von der resoluten Wachtmeisterin sofort entsorgt.

Kehren wir nun zum formalen Beweis von Satz 8.36 zurück. Seien $a, b \in \Sigma^*$ beliebige zwei Eingabewörter für unsere Funktion σ . Definiere σ durch

$$a\sigma b = \begin{cases} \langle x, \min(w_1, w_2) \rangle & \text{falls } a = \langle x, w_1 \rangle \text{ und } b = \langle x, w_2 \rangle \\ & \quad \text{für ein } x \in \Sigma^* \text{ und } w_1, w_2 \in \text{Wit}_M(x) \\ \langle x, x \rangle & \text{falls } (a = \langle x, x \rangle \text{ und } b = \langle x, w \rangle) \\ & \quad \text{oder } (a = \langle x, w \rangle \text{ und } b = \langle x, x \rangle) \\ & \quad \text{für ein } x \in \Sigma^* \text{ und ein } w \in \text{Wit}_M(x) \\ \text{nicht definiert} & \text{sonst,} \end{cases} \quad (8.18)$$

wobei $\min(w_1, w_2)$ das lexikographisch kleinere der Wörter w_1 und w_2 bezeichnet. Es bleibt zu zeigen, dass σ die gewünschten Eigenschaften hat. Dass σ kommutativ, ehrlich, s-ehrlich, in Polynomialzeit berechenbar und nichtinvertierbar ist, ist eine Routinesache, siehe Übung 8.13(d). Insbesondere ist σ also eine Einwegfunktion.

Nun wird gezeigt, dass σ stark ist. Für einen Widerspruch sei angenommen, dass es einen Polynomialzeit-Invertierer i bezüglich der ersten Komponente des Arguments von σ gibt. Das heißt, für jedes Wort z im Wertebereich von σ und für jede erste Komponente $a \in \Sigma^*$ eines Arguments, für welche es eine zugehörige zweite Komponente $b \in \Sigma^*$ dieses Arguments mit $a\sigma b = z$ gibt, gilt $a\sigma i(\langle a, z \rangle) = z$. Der Inverter i kann dann folgendermaßen verwendet werden, um L in Polynomialzeit zu entscheiden:

Schritt 1: Um für ein gegebenes Eingabewort x zu entscheiden, ob x in L ist oder nicht, berechne das Wort $u = i(\langle \langle x, x \rangle, \langle x, x \rangle \rangle)$.

Schritt 2: Berechne die eindeutigen Wörter v und w , für die $\langle v, w \rangle = u$ gilt, d.h., v und w sind die Projektionen unserer Paarungsfunktion $\langle \cdot, \cdot \rangle$ an der Stelle u .

Schritt 3: Akzeptiere x genau dann, wenn $v = x$ und $w \in \text{Wit}_M(x)$ gilt.

Dieser Algorithmus läuft in Polynomialzeit und zeigt somit, dass L in P liegt, was unserer Annahme widerspricht, dass $L \notin P$ gilt. Also ist σ nicht bezüglich der ersten

⁷ Kein Wunder, dass in dieser Stadt so wenige Verbrechen aufgeklärt werden.

Komponente eines Arguments invertierbar. Eine analoge Argumentation zeigt, dass σ auch nicht bezüglich der zweiten Komponente eines Arguments invertierbar ist. Es folgt, dass σ stark nichtinvertierbar ist.

Schließlich zeigen wir noch, dass σ assoziativ ist. Seien $a, b, c \in \Sigma^*$ beliebige fest gewählte Wörter, die Komponenten von Argumenten für σ sein können. Betrachte die Projektionen unserer Paarungsfunktion an den Stellen a , b und c : $a = \langle a_1, a_2 \rangle$, $b = \langle b_1, b_2 \rangle$ und $c = \langle c_1, c_2 \rangle$. Sei $k \in \{0, 1, 2, 3\}$ die Zahl, die angibt, wie viele von a_2 , b_2 und c_2 Elemente von $\text{Wit}_M(a_1)$ sind. Ist beispielsweise $a_2 = b_2 \in \text{Wit}_M(a_1)$, aber $c_2 \notin \text{Wit}_M(a_1)$, dann ist $k = 2$. Gemäß Definition 8.34 ist zu zeigen, dass

$$(a\hat{\sigma}b)\hat{\sigma}c = a\hat{\sigma}(b\hat{\sigma}c) \quad (8.19)$$

gilt, wobei $\hat{\sigma}$ die Fortsetzung von σ aus dieser Definition ist. Zwei Fälle sind nun zu unterscheiden.

Fall 1: $a_1 = b_1 = c_1$ und $\{a_2, b_2, c_2\} \subseteq \{a_1\} \cup \text{Wit}_M(a_1)$. Die Intuition in diesem Fall ist, dass die Anzahl der Zeugen für „ $a_1 \in A$ “, die in den beiden Argumenten von σ vorkommen, folgendermaßen um eins verringert wird:

- Kommt in keinem der beiden Argumente von σ ein Zeuge für „ $a_1 \in A$ “ vor, so ist σ nicht definiert, also ist \perp der Wert von $\hat{\sigma}$.
- Kommt in genau einem Argument von σ ein Zeuge für „ $a_1 \in A$ “ vor, so hat σ – und demnach auch $\hat{\sigma}$ – den Wert $\langle a_1, a_1 \rangle$.
- Kommt in beiden Argumenten von σ ein Zeuge für „ $a_1 \in A$ “ vor, so ist $\langle a_1, w \rangle$ der Wert von $\hat{\sigma}$, wobei $w \in \{a_2, b_2, c_2\}$ der lexikographisch kleinere dieser beiden Zeugen ist.

Aus den oben genannten drei Teilaufgaben können wir das Folgende schließen:

- Ist $k \in \{0, 1\}$, so gilt $(a\hat{\sigma}b)\hat{\sigma}c = \perp = a\hat{\sigma}(b\hat{\sigma}c)$.
- Ist $k = 2$, so gilt $(a\hat{\sigma}b)\hat{\sigma}c = \langle a_1, a_1 \rangle = a\hat{\sigma}(b\hat{\sigma}c)$.
- Ist $k = 3$, so gilt $(a\hat{\sigma}b)\hat{\sigma}c = \langle a_1, \min(a_2, b_2, c_2) \rangle = a\hat{\sigma}(b\hat{\sigma}c)$, wobei wieder $\min(a_2, b_2, c_2)$ den lexikographisch kleinsten der Zeugen a_2 , b_2 und c_2 bezeichnet.

Fall 2: Entweder gilt $(a_1 \neq b_1 \text{ oder } a_1 \neq c_1 \text{ oder } b_1 \neq c_1)$ oder es gilt $(a_1 = b_1 = c_1 \text{ und } \{a_2, b_2, c_2\} \not\subseteq \{a_1\} \cup \text{Wit}_M(a_1))$. In jedem dieser beiden Teilaufgaben von Fall 2 gilt:

$$(a\hat{\sigma}b)\hat{\sigma}c = \perp = a\hat{\sigma}(b\hat{\sigma}c).$$

In jedem der obigen Fällen gilt (8.19). Folglich ist σ assoziativ. Um den Beweis von Satz 8.36 abzuschließen, bleibt noch zu zeigen, dass σ zu einer totalen Funktion fortgesetzt werden kann, ohne dass dabei eine der anderen Eigenschaften von σ zerstört wird, siehe Übung 8.13(e). \square

Selbst wenn die Ungleichheit $P \neq NP$ bekannt wäre (und totale, starke, kommutative und assoziative Einwegfunktionen somit nach Satz 8.36 existieren würden), so würde daraus natürlich noch nicht die Sicherheit des Rivest–Sherman–Protokolls aus Abbildung 8.12 folgen. In Übung 8.13(f) sollen mögliche Sicherheitsprobleme dieses Protokolls diskutiert werden.

8.7 Übungen und Probleme

Aufgabe 8.1 Abbildung 8.1 präsentiert das Diffie–Hellman-Protokoll, siehe auch Beispiel 8.1.

- (a) Verifiziere, dass 12 eine Primitivwurzel von 17 ist, dass also $\langle 12 \rangle = \mathbb{Z}_{17}^*$ gilt.
- (b) Verifiziere mit dem *Square-and-Multiply*-Algorithmus aus Abbildung 7.2, dass Bobs Geheimzahl in Beispiel 8.1 wirklich $\beta = 12^{15} \equiv 10 \pmod{17}$ ist.
- (c) Führe das Diffie–Hellman-Protokoll für $p = 17$ und $\gamma = 11$ aus, wobei Alice den geheimen Exponenten $a = 5$ und Bob den geheimen Exponenten $b = 7$ wählt.
- (d) Bestimme alle Primitivwurzeln der Primzahl 101 und führe das Diffie–Hellman-Protokoll für eine selbst gewählte Primitivwurzel γ von 101 und für selbst gewählte private Exponenten a und b aus.

Aufgabe 8.2 (a) Definiere das Diffie–Hellman-Problem aus Definition 8.2 als ein Entscheidungsproblem, nennen wir es *Diffie–Hellman*, und zeige, dass die beiden Versionen unter polynomialzeit-beschränkten Turing-Reduktionen äquivalent sind:

- $\text{Diffie–Hellman} \in \text{P}^{\text{diffie–hellman}}$ und
 - $\text{diffie–hellman} \in \text{FP}^{\text{Diffie–Hellman}}$.
- (b) Definiere das Problem des diskreten Logarithmus aus Definition 2.42 als ein Entscheidungsproblem, nennen wir es *DLog*, und zeige, dass die beiden Versionen unter polynomialzeit-beschränkten Turing-Reduktionen äquivalent sind: $\text{DLog} \in \text{P}^{\text{dlog}}$ und $\text{dlog} \in \text{FP}^{\text{DLog}}$.

Aufgabe 8.3 Abbildung 8.2 zeigt den Algorithmus von Shanks. Berechne die folgenden Werte mit diesem Algorithmus:

- (a) $\log_{27} 89 \pmod{101}$;
- (b) $\log_{569} 413 \pmod{809}$.

Aufgabe 8.4 Abbildung 8.3 zeigt das Public-Key-Kryptosystem von ElGamal.

- (a) Seien $p = 101$ die gewählte Primzahl und $\gamma = 27$ die von Alice und Bob gewählte Primitivwurzel von 101. Alice wählt den privaten Exponenten $a = 23$ und Bob wählt den privaten Exponenten $b = 87$. Verschlüssele die Nachricht $m = 89$ mit ElGamals Kryptosystem und zeige, dass die Entschlüsselung gemäß (8.5) den ursprünglichen Klartext ergibt.
- (b) Wiederhole die Aufgabe aus (a) mit den Parametern $p = 809$, $\gamma = 569$, $a = 227$, $b = 781$ und $m = 801$.

Aufgabe 8.5 Abbildung 8.4 zeigt das digitale Signatur-Schema von ElGamal.

- (a) Wähle die Parameter $p = 1367$ und $\gamma = 5$ wie in Beispiel 8.6. Angenommen, Bob wählt als seine privaten Exponenten $b = 711$ und $s = 117$. Überprüfe, dass $\text{ggT}(s, p - 1) = 1$ gilt. Was ist Bobs Signatur für die Nachricht $m = 828$? Verifiziere mit der Bedingung (8.7), dass diese Signatur gültig ist.

- (b) Führe das digitale Signatur-Schema von ElGamal mit anderen, selbst gewählten Parametern aus.

Aufgabe 8.6 In Abschnitt 8.2.3 findet man einen Key-only-Angriff auf das ElGamal-Signatur-Verfahren.

- (a) Wähle wie in Beispiel 8.13 die Parameter $p = 1367$ und $\gamma = 5$. Angenommen, Erich kennt Bobs private Exponenten $b = 711$ und $s = 117$ aus Übung 8.5(a) nicht. Er kennt jedoch den entsprechenden öffentlichen Wert β . Weiter sei angenommen, dass Erich die Werte $x = 67$ und $y = 99$ wählt, um den Key-only-Angriff aus Beispiel 8.13 auszuführen. Bestimme seine Signatur und die entsprechende Nachricht.
- (b) Führe den Key-only-Angriff aus (a) mit anderen, selbst gewählten Parametern aus.

Aufgabe 8.7 In Abschnitt 8.2.3 wird ein Known-Message-Angriff auf das ElGamal-Signatur-Verfahren präsentiert.

- (a) Verifiziere, dass die in (8.12) angegebenen Werte σ , ρ und m tatsächlich die ElGamal-Verifikationsbedingung (8.7) erfüllen: $\gamma^m \equiv \beta^\sigma \sigma^\rho \pmod{p}$.
- (b) Führe diesen Known-Message-Angriff auf das ElGamal-Signatur-Verfahren mit selbst gewählten Parametern aus.

Aufgabe 8.8 Führe einen Known-Message-Angriff aus, der einen totalen Bruch des ElGamal-Signatur-Schemas erreicht, falls dieselbe Wert s (siehe Abbildung 8.4) zweimal zum Signieren verschiedener Nachrichten, m_1 und m_2 , verwendet wird.

Hinweis: Sei γ die im Protokoll verwendete Primitivwurzel der Primzahl p , und sei β Bobs öffentlicher Schlüssel. Angenommen, (σ, ρ_1) ist Bobs Signatur für m_1 und (σ, ρ_2) ist Bobs Signatur für m_2 . Schreiben wir $\sigma = \gamma^s$, so ergibt sich:

$$\beta^\sigma \sigma^{\rho_1} \equiv \gamma^{m_1} \pmod{p} \quad \text{und} \quad \beta^\sigma \sigma^{\rho_2} \equiv \gamma^{m_2} \pmod{p}.$$

Bestimme den unbekannten Wert s unter Verwendung der oben genannten Kongruenzen. Ist s erst einmal bekannt, so kann damit – wie am Ende von Abschnitt 8.2.3 erklärt – Bobs anderer privater Exponent, b , ermittelt werden.

Aufgabe 8.9 Abbildung 8.5 in Abschnitt 8.3 präsentiert Rabins Kryptosystem.

- (a) Angenommen, Bob wählt die Primzahlen $p = 43$ und $q = 47$, die auch in Beispiel 8.16 verwendet wurden. Der Rabin-Modul ist also $n = 2021$. Bestimme die Rabin-Verschlüsselungen der folgenden Nachrichten: $m_1 = 234$, $m_2 = 1789$ und $m_3 = 1989$. Bestimme weiterhin für jeden der entsprechenden Schlüsseltexte c_i , $i \in \{1, 2, 3\}$, alle Klartexte, die von der Rabin-Verschlüsselungsfunktion auf c_i abgebildet werden.
- (b) Angenommen, du bist Bob. Wiederhole (a) mit anderen, selbst gewählten Parametern.
- (c) Angenommen, Erich kennt den Rabin-Modul $n = 13081$. Wende die Methode von Satz 8.18 an, um die Primfaktoren von n durch einen Chosen-Ciphertext-Angriff zu bestimmen; siehe auch Beispiel 8.21.

Aufgabe 8.10 Abbildung 8.8 in Beispiel 8.23 zeigt das Zero-Knowledge-Protokoll für GI.

- (a) Beweise, dass der in Abbildung 8.8 berechnete Isomorphismus σ tatsächlich zu $\text{Iso}(G_a, H)$ gehört, wie von Arthur verlangt.
- (b) Führe dieses Protokoll für die Graphen $G_0 = G$ und $G_1 = H$ und den Isomorphismus $\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 5 & 4 & 5 & 1 \end{pmatrix}$ aus $\text{Iso}(G, H)$ aus, wobei G und H die Graphen aus Beispiel 2.50 in Abschnitt 2.4 sind. Wähle anschließend selbst einen Isomorphismus $\mu \in \text{Iso}(G, H)$ und betrachte jede mögliche Kombination von Zufallsbits m und a aus $\{0, 1\}$ für dieses Protokoll. Wann gelingt die Authentifikation?
- (c) Angenommen, du bist Saruman und du kennst Merlins geheimen Isomorphismus π nicht. Bitte einen Freund, die Rolle von Arthur zu übernehmen, und führe dieses Protokoll gemeinsam mit ihm aus.
- (d) Wiederhole (b) mit zwei anderen, selbst gewählten Graphen mit je 10 Knoten.
- (e) Modifiziere das Protokoll aus Abbildung 8.8 so, dass Sarumans Wahrscheinlichkeit, erfolgreich zu betrügen, in einer Runde des Protokolls höchstens 2^{-10} ist.

Aufgabe 8.11 Abbildung 8.10 in Abschnitt 8.4 zeigt das von Fiat und A. Shamir vorgeschlagene Zero-Knowledge-Protokoll für QR.

- (a) Beweise, dass dieses Protokoll funktioniert.
- (b) Angenommen, das Berechnen von Quadratwurzeln modulo n und das Faktorisieren des Moduls n sind praktisch nicht machbar. Beweise unter dieser Annahme, dass sich ein Lauscher in diesem Protokoll mit einer Wahrscheinlichkeit von nur 2^{-k} , für ein festes k , als Merlin ausgeben kann. Wie viele unabhängige Runden des Basisprotokolls müssen dafür ausgeführt werden?
- (c) Beweise, dass dieses Protokoll die Zero-Knowledge-Eigenschaft hat.
- (d) Angenommen, Merlin hat die Primzahlen $p = 43$ und $q = 47$ gewählt, also ist $n = 2021$. Seine Geheimzahl ist $s = 97$; es gilt $\text{ggT}(2021, 97) = 1$. Führe das Fiat–Shamir-Protokoll gemäß Abbildung 8.10 für verschiedene, selbst gewählte Werte r und a aus.

Aufgabe 8.12 Betrachte das Merkle–Hellman-Kryptosystem aus Abschnitt 8.5.

- (a) Entwurf zum Beweis von Behauptung 8.31 einen deterministischen Algorithmus, der das Problem SOS für alle Instanzen $\langle \mathbf{s}, T \rangle$ mit einer superwachsenden Folge \mathbf{s} von Größen in Polynomialzeit löst. Wende deinen Algorithmus auf die Instanz $\langle \mathbf{s}, T \rangle$ mit $\mathbf{s} = (2, 3, 6, 12, 25, 51, 101, 203, 415)$ und $T = 486$ an und überprüfe, dass er die ursprüngliche Nachricht $\mathbf{m} = (1, 0, 1, 1, 0, 1, 0, 0, 1)$ wiederherstellt, wie in Beispiel 8.32 behauptet wurde.
- (b) Modifiziere das Merkle–Hellman-Kryptosystem folgendermaßen: Der öffentliche Schlüssel ist eine superwachsende Folge $\mathbf{s} = (s_1, s_2, \dots, s_n)$ von Größen, und die Verschlüsselung wird mittels der Verschlüsselungsfunktion

$$E_{\mathbf{s}}(\mathbf{m}) = \sum_{i=1}^n m_i s_i$$

ausgeführt, die von $\{0, 1\}^n$ in $\{0, 1, \dots, \sum_{i=1}^n s_i\}$ abbildet. Da E_s injektiv ist, kann der Algorithmus aus Behauptung 8.31 zur autorisierten Entschlüsselung verwendet werden. Was stimmt mit dieser Vereinfachung des eigentlichen Merkle-Hellman-Kryptosystems nicht?

- (c) Angenommen, Bob wählt die superwachsende Folge

$$\mathbf{s} = (1, 3, 7, 15, 29, 57, 117, 235, 475, 940),$$

die Primzahl $p = 1889$ und den Faktor $b = 666$. Verschlüssele die Nachricht $\mathbf{m} = (1, 0, 1, 1, 0, 0, 1, 1, 0, 1)$. Entschlüssele deinen Schlüsseltext zur Überprüfung, ob die ursprüngliche Nachricht wiederhergestellt werden kann.

Aufgabe 8.13 Betrachte das Schlüsseltausch-Protokoll von Rivest und Sherman in Abbildung 8.12.

- (a) Modifizierte dieses Protokoll so, dass ein Protokoll für digitale Signaturen daraus entsteht. **Hinweis:** Siehe Rabi und Sherman [RS97].
- (b) Konstruiere eine unehrliche, in Polynomialzeit berechenbare zweistellige Funktion, die s-ehrlich ist, und konstruiere eine ehrliche, in Polynomialzeit berechenbare zweistellige Funktion, die nicht s-ehrlich ist.

Hinweis: Siehe L. Hemaspaandra, Pasanen und Rothe [HPR01].

- (c) Angenommen, die totale, stark nichtinvertierbare und assoziative Einwegfunktion σ , auf der das Protokoll in Abbildung 8.12 beruht, ist zusätzlich noch kommutativ, d.h., für alle $x, y \in \Sigma^*$ gilt $x\sigma y = y\sigma x$. Modifizierte sowohl das Schlüsseltausch-Protokoll aus Abbildung 8.12 als auch das digitale Signatur-Protokoll aus (a) zu Protokollen mit $k \geq 2$ Parteien, nicht nur Alice und Bob.

Hinweis: Siehe Rabi und Sherman [RS97].

- (d) Zeige, dass die im Beweis von Satz 8.36 in (8.18) definierte Funktion σ kommutativ, ehrlich, s-ehrlich, in Polynomialzeit berechenbar und nichtinvertierbar ist. **Hinweis:** Siehe L. Hemaspaandra und Rothe [HR99].

- (e) Zeige, dass die in (8.18) definierte nicht-totale Funktion σ zu einer totalen Funktion σ_t fortgesetzt werden kann, ohne dass dabei irgendeine andere der Eigenschaften von σ zerstört wird.

Hinweis: Siehe L. Hemaspaandra und Rothe [HR99]. Betrachte speziell die folgende Konstruktion von $\sigma_t : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$. Sei \hat{x} ein festes Wort, das nicht in L liegt, und argumentiere, weshalb ein solches Wort existieren muss. Sei weiter $\hat{a} = \langle \hat{x}, 1\hat{x} \rangle$. Argumentiere, weshalb \hat{a} weder die Form $\langle x, x \rangle$ für irgendein $x \in \Sigma^*$ noch die Form $\langle x, w \rangle$ für irgendein $x \in \Sigma^*$ und irgendeinen Zeugen $w \in W_M(x)$ hat. Nach (8.18) ist σ weder für (\hat{a}, b) noch für (b, \hat{a}) für irgendein $b \in \Sigma^*$ definiert. Definiere die totale Fortsetzung σ_t so, dass sie auf D_σ mit σ übereinstimmt und alle Paare $(a, b) \notin D_\sigma$ auf das „Müll“-Element \hat{a} abbildet. Beweise, dass σ_t eine starke, kommutative und assoziative Einwegfunktion ist.

- (f) Diskutiere die Sicherheit des Rivest–Sherman-Protokolls aus Abbildung 8.12 unter allen erforderlichen Gesichtspunkten. Gehe dabei insbesondere auf die

Unterschiede zwischen dem Worst-Case- und dem Average-Case-Modell ein.

Hinweis: Siehe L. Hemaspaandra und Rothe [HR99].

- (g) Beweise, dass keine totale, assoziative Funktion injektiv sein kann.

Hinweis: Siehe Rabi und Sherman [RS97].

Problem 8.1 (Bit-Sicherheit des Diskreten Logarithmus)

Beweise Satz 8.12: Sei $\langle p, \gamma, \alpha, i \rangle$ eine Instanz des Bit-Problems des diskreten Logarithmus, und sei $p - 1 = r2^q$ für eine ungerade Zahl r . Dann gilt:

- (a) Für jedes $i \leq q$ kann $\text{dlogbit}(\langle p, \gamma, \alpha, i \rangle)$ in Polynomialzeit berechnet werden, und

- (b) $\log_\gamma \alpha \bmod p$ kann in $\text{FP}^{\text{dlogbit}(\langle p, \gamma, \alpha, q+1 \rangle)}$ berechnet werden.

Hinweis für (b): Betrachte den Fall $q = 1$, d.h., $p - 1 = 2r$ für eine ungerade Zahl r . Somit ist $p \equiv 3 \pmod{4}$. Zeige, dass dann jedes $\alpha \in \text{QR}_p$ die zwei Quadratwurzeln $\pm\alpha^{(p+1)/4}$ hat. Zeige außerdem, dass

$$\text{dlogbit}(\langle p, \gamma, \alpha, 1 \rangle) \neq \text{dlogbit}(\langle p, \gamma, p - \alpha, 1 \rangle),$$

aus $p \equiv 3 \pmod{4}$ folgt. Ist $\alpha \equiv \gamma^a \pmod{p}$ für einen unbekannten geraden Exponenten a , dann ergibt sich entweder

$$\alpha^{(p+1)/4} \equiv \gamma^{a/2} \pmod{p} \quad \text{oder} \tag{8.20}$$

$$-\alpha^{(p+1)/4} \equiv \gamma^{a/2} \pmod{p}. \tag{8.21}$$

Welche der Kongruenzen (8.20) oder (8.21) gilt, kann mit Satz 8.11 effizient getestet werden, vorausgesetzt, dass der Wert von $\text{dlogbit}(\langle p, \gamma, \alpha, 2 \rangle)$ bekannt ist, denn

$$\text{dlogbit}(\langle p, \gamma, \alpha, 2 \rangle) = \text{dlogbit}(\langle p, \gamma, \gamma^{a/2}, 1 \rangle).$$

Entwurf auf der Grundlage dieser Eigenschaft einen Polynomialzeit-Algorithmus, der die Orakelantwort $\text{dlogbit}(\langle p, \gamma, \alpha, 2 \rangle)$ benutzt, um die Binärdarstellung von $a = \log_\gamma \alpha \bmod p$ zu berechnen. Die Details dieses Beweises findet man in Stinson [Sti05].

Problem 8.2 (Brechen des Merkle–Hellman-Kryptosystems)

Zeige, wie man die Grundvariante des Merkle–Hellman-Kryptosystems aus Abbildung 8.11 in Abschnitt 8.5 brechen kann.

Hinweis: Siehe A. Shamir [Sha84] und auch A. Shamir und Zippel [SZ80]. Die wesentliche Idee dabei ist eine Anwendung des effizienten Algorithmus für *Integer Programming* mit einer festen Anzahl von Variablen, der auf H. Lenstra zurückgeht [Len83].

Problem 8.3 (Ein Protokoll ohne öffentliche Schlüssel)

Die Aufgabe in diesem Problem versteht man am besten, wenn man die folgende Geschichte aufmerksam liest.

Story 8.38 Wie es bei Kindern ihres Alters üblich ist, haben Paula und Ella sehr gern Geheimnisse miteinander.⁸ Eines Tages musste Paula jedoch ins Krankenhaus gehen, um eine ernste Lungenentzündung behandeln zu lassen. Die beiden sonst unzertrennlichen Schwestern waren deshalb sehr unglücklich. Zwar erholte sich Paula zum Glück sehr rasch von ihrer Krankheit, musste aber noch eine weitere Woche im Krankenhaus bleiben. Ella durfte ihre Schwester nicht besuchen, denn gesunden Kindern ist der Besuch der Intensivstation in diesem Krankenhaus nicht erlaubt. Nur ihre Eltern hatten eine Besuchserlaubnis. Dennoch waren Paula und Ella natürlich scharf darauf, miteinander zu kommunizieren und weiterhin Geheimnisse auszutauschen. Ihr Problem bestand darin, dass sie nun dafür die Hilfe ihrer Eltern benötigten, und unter Kindern ist es allgemein bekannt, dass Eltern niemals vertrauenswürdig sind, wenn es um das Bewahren eines Geheimnisses geht.

Schritt	Ella	Eltern	Paula
1	Ella kauft eine Schachtel S und zwei Vorhängeschlösser, x und y , bittet ihre Eltern, Paula im Krankenhaus y zu übergeben und behält x selbst		
2	verschließt ihre Botschaft m in der Schachtel S mit ihrem Vorhängeschloss x		
3		$S_x \Rightarrow$	
4			bringt ihr Vorhängeschloss y an der Schachtel S an
5		$\Leftarrow S_x^y$	
6	entfernt ihr Vorhängeschloss x		
7		$S^y \Rightarrow$	
8			entfernt ihr Vorhängeschloss y und liest die Botschaft m

Abb. 8.13. Paula und Ella benutzen ein Protokoll ohne öffentliche Schlüssel

Nun denken sie sich den folgenden raffinierten Plan aus, wie sie sich Geheimnisse mitteilen können, ohne dass ihre Eltern diese erfahren. Abbildung 8.13 zeigt das Protokoll, das Paula und Ella benutzen: Ella kauft von ihrem Taschengeld eine Schachtel S und zwei Vorhängeschlösser, x und y , bittet ihre Eltern, Paula im Krankenhaus das Schloss y zu übergeben, und behält x selbst. Dann schreibt sie die Botschaft m , die das Geheimnis enthält, das sie Paula mitteilen möchte.⁹ Sie legt m in die Schachtel S und verschließt diese mit ihrem Vorhängeschloss x . Die so verschlossene Schachtel wird mit S_x bezeichnet. Ihre Eltern nehmen S_x nun mit ins Krankenhaus und überreichen sie Paula, die sie aber nicht öffnet (sie kann sie ja gar nicht öffnen, weil sie

⁸ Leider wurden sie mir vorenthalten. Ich habe keine Ahnung, worum es dabei geht.

⁹ Wie gesagt habe ich leider keine Ahnung, was ihr Geheimnis sein könnte.

keinen Schlüssel für x hat), sondern stattdessen verschließt sie sie noch einmal mit ihrem eigenen Vorhängeschloss y . Die doppelt verschlossene Schachtel wird mit S_x^y bezeichnet, und die Eltern nehmen sie mit zurück nach Hause, wo sie sie Ella überreichen, die nun ihr Schloss x entfernt. Jetzt ist die Schachtel, S^y , immer noch mit Paulas Vorhängeschloss verschlossen. Bei ihrem nächsten Krankenbesuch nehmen die Eltern S^y mit und überreichen sie Paula, die sicherheitshalber wartet, bis ihre Eltern wieder gegangen sind, und dann erst ihr eigenes Schloss y entfernt, die nun unverschlossene Schachtel öffnet und – vor Neugier fast platzend – Ellas Botschaft zu lesen beginnt.

Hier ist die Aufgabe: Entwirf ein Protokoll, das so ähnlich wie das aus Abbildung 8.13 funktioniert. Anders als das von Paula und Ella sollte dieses neue Protokoll jedoch keine Schachteln und Vorhängeschlösser brauchen, sondern stattdessen geeignete mathematische Begriffe und Funktionen verwenden.

Hinweis: Zu jedem Zeitpunkt, zu dem die Eltern Zugriff auf die Schachtel hatten, war diese – traurig genug – verschlossen. Auch das neu zu entwerfende Protokoll sollte die Eigenschaft haben, dass immer dann, wenn Information übertragen wird, der Lauscher (unter vernünftigen komplexitätstheoretischen Annahmen) nicht in der Lage sein sollte, diese zu dechiffrieren. Die „Schachtel“ darf also nie „unverschlossen“ transportiert werden.

Auch ist die Feststellung interessant, dass Paula und Ella über keinen gemeinsamen geheimen Schlüssel verfügen müssen, um dieses Protokoll auszuführen. Aus diesem Grund wird dieses Protokoll, das auf eine unveröffentlichte Arbeit von A. Shamir zurückgeht, manchmal als das „No-Key-Protokoll“ bezeichnet. Paula und Ella reicht jeweils ihr eigener privater Schlüssel, denn sie müssen nur ihre eigenen Vorhängeschlösser auf- und zuschließen.

Eine letzte Bemerkung ist, dass die Reihenfolge des Abschließens und des Entfernen der beiden Vorhängeschlösser hier keine Rolle spielt: Das Schloss x kann von der Schachtel entfernt werden, auch wenn diese noch durch das Schloss y verschlossen ist, welches *nach x* angebracht worden war. Anders gesagt sind diese Operationen in gewissem Sinne *symmetrisch* oder *vertauschbar*. Die gesuchte Funktion, auf der das neue Protokoll beruhen sollte, muss dieses Merkmal ebenfalls aufweisen. Ein etwas konkreterer Hinweis ist, dass die modulare Potenzfunktion, die im Diffie-Hellman-Protokoll aus Abbildung 8.1 verwendet wird, in den Exponenten sozusagen ebenfalls „symmetrisch“ oder „vertauschbar“ ist: $\gamma^{ab} \equiv \gamma^{ba} \pmod{p}$.

Ach ja ... und wenn du zufällig irgendwie das in Story 8.38 erwähnte Geheimnis von Ella und Paula erfährst, dann teile es mir bitte mit. Ich bin ihr Vater.

8.8 Zusammenfassung und bibliographische Notizen

Allgemeine Bemerkungen: Wie in Abschnitt 7.6 erwähnt wurde, legten Diffie und Hellman die Grundlagen der Public-Key-Kryptographie. Ihre bahnbrechende Arbeit [DH76] ist ein Meilenstein in der Geschichte der Kryptographie, nicht nur, weil

sie als Erste eine Lösung des Schlüsseltausch-Problems fanden, das in der symmetrischen Kryptographie eine ganz zentrale Rolle spielt, sondern auch, weil sie als Erste die Idee der Public-Key-Kryptographie vorschlugen. In Abschnitt 7.6 wurde erwähnt, dass ähnliche Erkenntnisse unabhängig und zum Teil bereits früher im nicht-öffentlichen (d.h. im nachrichtendienstlichen) Bereich gewonnen wurden.

Die meisten der in diesem Kapitel präsentierten Public-Key-Kryptosysteme und kryptographischen Protokolle können auch in anderen Lehrbüchern der Kryptographie gefunden werden; siehe z.B. Stinson [Sti05], Buchmann [Buc01a, Buc01b], Salomaa [Sal96] und Goldreich [Gol01].

Spezielle Bemerkungen: Das in Abbildung 8.1 präsentierte Schlüsseltausch-Protokoll von Diffie und Hellman findet man in [DH76]. ElGamals Kryptosystem aus Abbildung 8.3 und das verwandte digitale Signatur-Schema aus Abbildung 8.4 wurden in [ElG85] vorgestellt. Der „Baby-Step Giant-Step“-Algorithmus aus Abbildung 8.2, mit dem diskrete Logarithmen berechnet werden, geht auf Shanks [Sha54] zurück. Für mehr Hintergrund und Details zum Problem des diskreten Logarithmus wird der Leser auf die Übersichtsarbeiten von Odlyzko und LaMacchia [Odl85, LO91, Odl00] verwiesen.

Schnorr [Sch90] schlug eine Modifizierung des ElGamal-Signatur-Verfahrens vor, das schließlich zum *Digital Signature Standard* der Vereinigten Staaten führte, siehe [Nat91, Nat92]. Seien p und q Primzahlen mit $p \equiv 1 \pmod{q}$; typischerweise wird p durch ein Binärwort mit 1024 Bits und q durch ein Binärwort mit 160 Bits dargestellt. In Schnorrs Modifizierung des ElGamal-Signatur-Verfahrens signiert Bob eine Nachricht der Länge $\log q$ mit einer digitalen Unterschrift der Länge etwa $2 \log q$. Schnorrs Idee bestand im Wesentlichen darin, die Berechnung dadurch zu beschleunigen, dass in einer Untergruppe von \mathbb{Z}_p^* gearbeitet wird, die die Größe q hat und für die das Berechnen diskreter Logarithmen als schwer gilt.

Rabins Kryptosystem wurde in [Rab79] präsentiert. Williams [Wil80] und Kurosawa, Ito und Takeuchi [KIT88] schlugen Public-Key-Kryptosysteme vor, die wie Rabins System in dem Sinn beweisbar sicher sind, dass sie zu brechen so schwer ist wie das Faktorisieren. Anders als Rabins System jedoch ist die Entschlüsselung in ihren Systemen nicht mehrdeutig.

Die Begriffe des interaktiven Beweissystems und des Zero-Knowledge-Protokolls wurden von Goldwasser, Micali und Rackoff [GMR89] eingeführt. Der verwandte Begriff des Arthur-Merlin-Spiels geht auf Babai und Moran [Bab85, BM88] zurück. Zachos und Heller [Zac88, ZH86] und Boppana, Hästad und Zachos [BHZ87] untersuchten diese Begriffe ebenfalls schon sehr früh.

Interaktive Beweissysteme, Zero-Knowledge, Arthur-Merlin-Spiele und verwandte Begriffe werden in vielen Forschungsarbeiten und Lehrbüchern untersucht, sowohl in kryptographischer als auch in komplexitätstheoretischer Hinsicht. Ein gefeiertes komplexitätstheoretisches Resultat ist beispielsweise A. Shamir [Sha92] zu verdanken: Interaktive Beweissysteme haben genau dieselbe Berechnungskraft wie Turingmaschinen, die in polynomialem Raum arbeiten. Hinsichtlich der kryptographischen Anwendungen dieser Begriffe ist als eine der tiefgründigsten und umfassendsten Quellen das vierte Kapitel in Goldreichs Buch [Gol01] zu nennen. Diese wichtigen

Begriffe werden auch in einer Vielzahl von Übersichtsarbeiten und weiteren Buchkapiteln präsentiert, etwa in denen von Balcázar, Díaz und Gabarró [BDG90], Beutelspacher et al. [Beu94, BSW01, Beu02], Bovet und Crescenzi [BC93], Buchmann [Buc01a, Buc01b], Du und Ko [DK00], Feigenbaum [Fei92], Goldreich [Gol88], Goldwasser [Gol89], Köbler, Schöning und Torán [KST93], Papadimitriou [Pap95], Rothe [Rot02, Rot07b], Stinson [Sti95], Schöning [Sch95b], Wechsung [Wec00] und Wegener [Weg03].

Der spezielle Begriff des Zero-Knowledge, der in Definition 8.24 eingeführt wurde, heißt „*honest-verifier perfect zero-knowledge*“, was bedeutet, dass (a) der Verifizierer ehrlich ist und (b) die Wahrscheinlichkeitsverteilungen im Original- und simulierten Protokoll perfekt identisch sind. Wie unmittelbar nach dieser Definition erwähnt wurde, ist die Annahme (a) für die meisten kryptographischen Anwendungen womöglich nicht realistisch genug, da ein unehrlicher Verifizierer das Protokoll zu seinem Vorteil abändern könnte. Um ein solches Verhalten zu vermeiden, kann man die Definition des Zero-Knowledge dadurch abändern, dass man für jeden potenziellen Verifizierer A verlangt, dass ein Simulator S existiert, so dass das von ihnen gemeinsam erzeugte simulierte Protokoll nicht vom Originalprotokoll unterscheidbar ist. Wenn die Zufallsbits öffentlich sind, können *honest-verifier* Zero-Knowledge-Protokolle stets in Protokolle umgeformt werden, die die Zero-Knowledge-Eigenschaft sogar in der Gegenwart unehrlicher Verifizierer besitzen.

Es wurde auch erwähnt, dass die Annahme (b) womöglich zu strikt ist und dass man mit schwächeren Anforderungen als denen des perfekten Zero-Knowledge ebenso gut auskommen könnte. Zu solchen schwächeren Varianten von Zero-Knowledge gehören das „*statistical zero-knowledge*“ (auch bezeichnet als „*almost-perfect zero-knowledge*“) und das „*computational zero-knowledge*“. Das erstere Modell verlangt, dass die im Original- und simulierten Protokoll übertragene Information durch geeignete statistische Tests ununterscheidbar sein soll. Das letztere Modell verlangt lediglich, dass die im Original- und simulierten Protokoll übertragene Information „berechnungsununterscheidbar“ sein soll, d.h., für jede randomisierte polynomialzeit-beschränkte Turingmaschine soll die Wahrscheinlichkeit dafür, dass diese Unterschiede in den entsprechenden Wahrscheinlichkeitsverteilungen entdeckt, vernachlässigbar klein sein.

Das Zero-Knowledge-Protokoll für GI aus Beispiel 8.23 geht auf Goldreich, Micali und Wigderson [GMW91] zurück. Sie bewiesen auch eines der wichtigsten Resultate über Zero-Knowledge: Jedes Problem in NP hat ein „berechnungsununterscheidbares“ (also ein „*computational*“) Zero-Knowledge-Protokoll unter der plausiblen Annahme, dass es kryptographisch sichere Bit-Commitment-Schemata gibt. Eine der Schlüsselideen in ihrem Beweis ist der Entwurf eines berechnungsununterscheidbaren Zero-Knowledge-Protokolls für ein gut bekanntes NP-vollständiges Problem, nämlich 3-Colorability, siehe Satz 3.56. Im Gegensatz dazu fanden Brassard und Crépeau [BC89] gute Indizien dafür, dass solch eine starke Aussage im Modell des perfekten Zero-Knowledge aus Definition 8.24 vermutlich nicht gilt.

Das Zero-Knowledge-Protokoll für QR aus Abbildung 8.10 wurde von Fiat und A. Shamir [FS86] vorgeschlagen. Das Fiat–Shamir–Protokoll ist besonders gut für die Authentifikation in großen Computer-Netzwerken geeignet. Es ist ein Public-Key-

Protokoll, aber dabei effizienter als die meisten anderen Public-Key-Protokolle. Es kann auf einer Chip-Karte implementiert werden. Und es hat die Zero-Knowledge-Eigenschaft. Diese Vorteile haben zu einer rasanten Verbreitung dieses Protokolls in praktischen Anwendungen geführt. Beispielsweise ist es im „Videocrypt“-System [CH91] integriert, das beim Bezahlfernsehen genutzt wird. Feige, Fiat und A. Shamir [FFS88] verbesserten das ursprüngliche Fiat-Shamir-Zero-Knowledge-Identifikationsschema zu einem Zero-Knowledge-Protokoll, in dem nicht nur die geheimen Quadratwurzeln modulo n nicht enthüllt werden, sondern auch nicht die Information, ob es eine Quadratwurzel modulo n überhaupt gibt.

Die Grundvariante des Merkle–Hellman-Kryptosystems aus Abbildung 8.11 ist Merkle und Hellman [MH78] zu verdanken. Wie bereits im Hinweis zur Lösung von Problem 8.2 erwähnt wurde, gelang es A. Shamir [Sha84] in den frühen 1980ern, ihr System zu brechen, siehe auch A. Shamir und Zippel [SZ80]. Die iterierte Variante des Merkle–Hellman-Systems wurde von Brickell [Bri85] gebrochen, siehe auch Adleman [Adl83] und Brickell, Lagarias und Odlyzko [BLO83, Bri83, Lag83].

Eine andere Variante eines auf einem Rucksack-Problem basierenden Public-Key-Kryptosystems, vorgeschlagen von Chor und Rivest [CR88], widerstand etwa ein Jahrzehnt lang allen Versuchen, sie zu brechen. Zum Beispiel war der Angriff von Schnorr und Hörner [SH95] nur teilweise erfolgreich; sie brachen das Chor-Rivest-Kryptosystem in den Dimensionen 103 und 151. Ihr Angriff und verschiedene andere Angriffe auf Kryptosysteme, deren Sicherheit auf dem Rucksack-Problem beruht, verwenden Gitterreduktionsalgorithmen, die den LLL-Algorithmus von Lenstra, Lenstra und Lovász [LLL82] verbessern. Gitter-basierte Techniken sind beim Brechen von solchen Systemen besonders nützlich, die auf Instanzen des Rucksack- oder, spezieller, des Subset-of-Sums-Problems mit einer *geringen Dichte* beruhen. Im Gegensatz dazu basiert das Chor–Rivest-Kryptosystem auf Rucksack-Instanzen hoher Dichte. Schließlich wurde dieses System dennoch von Vaudenay [Vau01] gebrochen, der algebraische Methoden anwandte. Unter den auf dem Rucksack-Problem oder auf Gitter-Problemen beruhenden Kryptosystemen sind derzeit das NTRU-Kryptosystem, das von Hoffstein, Pipher und Silverman [HPS98] entwickelt wurde, und diesem verwandte Protokolle [HPS01, HHGP⁺03] immer noch ungebrochen, siehe auch Coppersmith und A. Shamir [CS97].

Kellerer, Pferschy und Pisinger haben ein tiefgründiges, sehr umfangreiches Werk über Rucksack-Probleme geschrieben [KPP04], wobei sie sich auf die algorithmischen Eigenschaften und die Berechnungskomplexität dieser Probleme konzentrieren. Für mehr Hintergrund und Details über Kryptosysteme, die auf Subset-of-Sums- und Rucksack-Problemen beruhen, wird der Leser auf Kapitel 3 des Buches von Salomaa [Sal96] verwiesen. Insbesondere wird dort A. Shamirs kryptoanalytischer Angriff auf die Grundvariante des Merkle–Hellman-Kryptosystems erläutert, siehe auch den Übersichtsartikel von Brickell und Odlyzko [BO92]. Lagarias und Odlyzko [LO85] gehörten zu den Ersten, die Subset-of-Sums-Probleme mit Gitterreduktionstechniken lösten. Für mehr Hintergrund und Details über die Komplexität von Gitter-Problemen im Allgemeinen und insbesondere hinsichtlich ihrer Anwendungen in der Kryptologie wird der interessierte Leser auf das Buch von Micciancio

und Goldwasser [MG02] und auf die Übersichtsartikel von Cai [Cai99], Kumar und Sivakumar [KS01] sowie Nguyen und Stern [NS01] verwiesen.

Andere Public-Key-Kryptosysteme, deren Sicherheit auf NP-vollständigen Problemen beruht, wurden z.B. von Shamir [Sha83], Impagliazzo und Naor [IN96] und Ajtai und Dwork [AD97] vorgeschlagen. Even, Selman und Yacobi [ESY84, EY80] studierten ebenfalls Kryptosysteme, die zu brechen NP-hart ist. Diesbezüglich entwickelten sie eine Theorie von „*promise problems*“ und konzentrierten sich auf deren Anwendungen in der Public-Key-Kryptographie; siehe auch Grollmann und Selman [GS88].

Das Schlüsseltausch-Protokoll in Abbildung 8.12 wird in [RS97] Rivest und Sherman zugeschrieben. Rabi und Sherman schlugen das verwandte digitale Signatur-Schema vor [RS97], siehe Übung 8.13(a). Beide Protokolle verwenden eine Funktion σ , die lediglich durch ihre Eigenschaften spezifiziert ist: σ soll eine totale, stark nichtinvertierbare, kommutative und assoziative Einwegfunktion sein. Es ist nicht bekannt, ob solche Funktionen überhaupt existieren. Rabi und Sherman [RS97] bewiesen, dass kommutative und assoziative Einwegfunktionen genau dann existieren, wenn $P \neq NP$ gilt. Allerdings sind die von ihnen konstruierten Funktionen weder total¹⁰ noch stark nichtinvertierbar, noch nicht einmal, wenn man von der Annahme $P \neq NP$ ausgeht. Sie ließen die Frage offen, ob $P \neq NP$ auch eine hinreichende Bedingung für die Existenz von Funktionen ist, die alle für die Rivest–Sherman- und Rabi–Sherman-Protokolle erforderlichen Eigenschaften besitzen.

Diese Frage wurde von L. Hemaspaandra und Rothe [HR99] beantwortet, und zwar als Satz 8.36: Totale, stark nichtinvertierbare, kommutative und assoziative Einwegfunktionen existieren im Worst-Case-Modell unter der (unbewiesenen, doch plausiblen) Annahme, dass $P \neq NP$ gilt. Einwegfunktionen mit diesen Eigenschaften sind seither in verschiedenen Zusammenhängen gründlich untersucht worden; siehe z.B. die Arbeit von L. Hemaspaandra, Pasanen und Rothe [HPR01], Homer [Hom04], L. Hemaspaandra, Rothe und A. Saxena [HRS08, HRS05] und den Übersichtsartikel von Beygelzimer et al. [BHHR99]. Es ist allerdings wichtig, zu betonen, dass Satz 8.36 weit davon entfernt ist, die Sicherheit der Rivest–Sherman- und Rabi–Sherman-Protokolle zu beweisen. Das heißt, diesen Protokollen fehlt ein Beweis der Sicherheit, selbst wenn die Ungleichheit $P \neq NP$ bekannt wäre, d.h., selbst wenn eine Funktion σ mit den gewünschten Eigenschaften existierte.

Die Sicherheitsprobleme der Rivest–Sherman- und Rabi–Sherman-Protokolle werden in [HR99] genauer diskutiert, siehe auch Übung 8.13(f). Kurz gesagt, die starke Nichtinvertierbarkeit schließt lediglich den offensichtlichen direkten Angriff aus, der in Abschnitt 8.6 erwähnt wurde, aber sie verhindert nicht andere mögliche Angriffsarten. Außerdem werden die Begriffe der Nichtinvertierbarkeit und der starken Nichtinvertierbarkeit im Worst-Case-Modell definiert, welches in der ange-

¹⁰ Rabi und Sherman [RS97] schlugen eine Konstruktion vor, von der sie behaupteten, sie könne aus jeder gegebenen nicht-totalen (schwach) assoziativen Einwegfunktion eine totale und assoziative Einwegfunktion machen. Jedoch ist der Beweis ihrer Behauptung fehlerhaft: Wenn ihre Behauptung stimmen würde, dann würde der unwahrscheinliche Kollaps $UP = NP$ unmittelbar folgen, wie in [HR99] gezeigt wurde.

wandten Kryptographie ungeeignet ist. Für kryptographische Anwendungen müsste man Funktionen konstruieren, die unter plausiblen Annahmen sogar im Average-Case-Modell nichtinvertierbar und stark nichtinvertierbar sind. In der Komplexitätstheorie gibt es interessante Resultate, die hoffnungsvolle mögliche Kandidaten für dieses Ziel benennen. So bewies etwa Ajtai [Ajt96], dass bestimmte Varianten des NP-vollständigen Shortest-Lattice-Vector-Problems im Worst-Case-Modell und im Average-Case-Modell etwa gleich schwer sind. Die Sicherheit des von Ajtai und Dwork [AD97] entworfenen Public-Key-Kryptosystems beruht auf dieser Worst-Case/Average-Case-Äquivalenz.

Keines der beiden in diesem Kapitel präsentierten Schlüsseltausch-Protokolle hat bisher einen Beweis der Sicherheit, weder das Diffie–Hellman-Protokoll aus Abbildung 8.1 noch das Rivest–Sherman-Protokoll aus Abbildung 8.12. Ein großer Unterschied zwischen diesen beiden Protokollen ist, dass das Rivest–Sherman-Protokoll auf einer unspezifischen Funktion σ beruht, die nur über ihre Eigenschaften beschrieben ist, wohingegen das Diffie–Hellman-Protokoll eine konkrete, spezifische Funktion als seinen Grundbaustein benutzt. Seine Sicherheit beruht auf der (unbewiesenen, doch plausiblen) Annahme, dass die Berechnung diskreter Logarithmen eine schwere Aufgabe ist. Jedoch ist nicht bekannt, ob Diffie–Hellman zu brechen ebenso schwer ist wie das Berechnen diskreter Logarithmen. Ein gewisser Fortschritt ist von Maurer und Wolf [MW99, MW00] gemacht worden, die Bedingungen formulierten, unter denen die Härte des Brechens von Diffie–Hellman in Beziehung zu der des Berechnens diskreter Logarithmen gesetzt werden kann. Wieder beruhen ihre Resultate auf unbewiesenen, doch plausiblen Annahmen: Diffie–Hellman zu brechen und diskrete Logarithmen zu berechnen sind in einer zyklischen Gruppe polynomialzeit-äquivalente Aufgaben, wenn man voraussetzt, dass $v(p)$ polynomell in $\log p$ ist, wobei $v(p)$ das Minimum der jeweils größten Primfaktoren von d bezeichnet, genommen über alle Zahlen d im Intervall $[p - 2\sqrt{p} + 1, p + 2\sqrt{p} + 1]$.

Der Begriff der komplexitätstheoretischen Einwegfunktion (also der im Worst-Case-Modell definierten Einwegfunktion) wurde von Grollmann und Selman [GS88] eingeführt; siehe auch Berman [Ber77], Brassard, Fortune und Hopcroft [BFH78, Bra79] sowie Ko [Ko85]. Komplexitätstheoretische Einwegfunktionen verschiedener Typen und verwandte Begriffe sind seither intensiv untersucht worden; siehe z.B. [AR88, Wat88, HH91, Sel92, RS93, HRW97b, RS97, HR99, BHHR99, HR00, HPR01, RH02, FFNR03, HT03b, Hom04, HRS05, HRS08]. Beispielsweise sind Einwegfunktionen im Worst-Case-Modell eng mit der Isomorphievermutung verwandt, siehe die Vermutungen 3.73, 3.79 und 3.80 in Abschnitt 3.6.2.

In einer anderen Forschungslinie ist die Existenz verschiedener Typen von Einwegfunktionen im Worst-Case-Modell hinsichtlich der Separationen geeigneter Komplexitätsklassen charakterisiert worden. Beispielsweise wurde die Frage, ob es Einwegpermutationen (also totale, injektive und surjektive Einwegfunktionen) gibt oder nicht, von Grollmann und Selman [GS88] aufgeworfen, von L. Hemaspaandra und Rothe [HR00, RH02] weiter untersucht und schließlich von Homan und Thakur [HT03b] gelöst. Letztere zeigten, dass Einwegpermutationen genau dann existieren, wenn $P \neq UP \cap coUP$ gilt. Fenner, Fortnow, Naik und J. Rogers [FFNR03] bewiesen, dass partielle, *many-to-one* und surjektive Einwegfunktionen genau dann

existieren, wenn $P \neq NP$ gilt; verwandte Resultate wurden unabhängig von L. Hemaspaandra, Rothe und Wechsung [HRW97a] erzielt.

Zwei Anmerkungen zum Schluss sind, dass Story 8.37 von einer in [BHHR99] erzählten Geschichte inspiriert ist und dass Story 8.38 auf einer wahren Geschichte beruht.

Tabellenverzeichnis

2.1	Testlauf des Euklidischen Algorithmus	13
2.2	Testlauf des erweiterten Euklidischen Algorithmus	15
2.3	Die ersten zwanzig Fibonacci-Zahlen	16
2.4	Die Fibonacci-Zahlen vermehren sich wie die Kaninchen und umgekehrt	17
2.5	Überführungsfunktionen des DEA und des NEA aus Abbildung 2.4 .	25
2.6	Die Überführungsfunktion δ des LBA M für $L = \{a^n b^n c^n \mid n \geq 1\}$..	31
2.7	Interpretation der Zustände von M	31
2.8	Wahrheitstafeln für einige boolesche Operationen	34
2.9	Äquivalenzen zwischen booleschen Formeln	36
3.1	Einige typische Komplexitätsklassen	70
3.2	Vergleich einiger Polynomial- und Exponentialfunktionen	71
3.3	Was passiert, wenn die Computer schneller werden?	71
3.4	Variablen in der Cook-Reduktion	102
4.1	Verschlüsselung durch die Verschiebungschiffre mit Schlüssel $k = 17$	151
4.2	Verschlüsselung durch die affine Chiffre mit Schlüssel $k = (5, 7)$...	152
4.3	Häufigkeiten von Buchstaben in langen, typischen englischen Texten	153
4.4	Häufigkeiten der Buchstaben im Schlüsseltext aus Beispiel 4.7 ..	154
4.5	Raten in der Häufigkeitsanalyse	155
4.6	Vigenère-Quadrat	156
4.7	Verschlüsselung durch die Vigenère-Chiffre mit Schlüssel ELLA ..	156
4.8	Beispiel einer Verschlüsselung mit der Hill-Chiffre	159
4.9	Kryptoanalyse eines polyalphabetischen Systems mit Periode 7	160
4.10	Kasiskis Methode: ein durch die Vigenère-Chiffre erzeugter Schlüsseltext	161
4.11	Kasiskis Methode: zweite Spalte des neu angeordneten Schlüsseltexts	162
4.12	Kasiskis Methode: entschlüsselter Vigenère-Schlüsseltext	163

4.13 Known-Plaintext-Angriff auf die Hill-Chiffre	164
4.14 Wahrheitstafel der Operation <i>Exklusives-Oder</i>	167
4.15 Blockverschlüsselung im CFB-Modus	169
4.16 Blockverschlüsselung im OFB-Modus	170
4.17 Ein mit der affinen Chiffre erzeugter Schlüsseltext für Übung 4.2(a) .	184
4.18 Zwei aus dem gleichen Klartext entstandene Schlüsseltexte für Übung 4.3	184
4.19 Ein mittels der Hill-Chiffre erzeugter Schlüsseltext für Übung 4.4(d)	185
4.20 Ein mittels der Vigenère-Chiffre erzeugter Schlüsseltext für Übung 4.5	186
4.21 Beaufort-Quadrat	187
 5.1 Beispiel einer \leq_{tt}^P -Reduktion von IN-Odd auf IS	230
 6.1 Laufzeiten ausgewählter Algorithmen für das Erfüllbarkeitsproblem .	295
6.2 Eine ZPP-Berechnung	307
6.3 Definition des Prädikats C im Beweis von Satz 6.22	319
 7.1 Beispiel einer RSA-Verschlüsselung	354
7.2 Sieb des Eratosthenes	357
7.3 Primzahltest unter Verwendung des Kleinen Fermat	358
7.4 Fermat-Zeugen und Fermat-Lügner für $n = 143$	359
7.5 Einige MR-Zeugen und MR-Lügner für die Carmichael-Zahl 561 ..	365
7.6 Berechnung von $a^{(n-1)/2} \bmod n$ beim Solovay–Strassen-Test . .	372
7.7 Berechnung von $x^i \bmod n$ in Pollards $(p - 1)$ -Algorithmus	378
7.8 Berechnung von $\sigma(x)$ beim quadratischen Sieb	380
7.9 Bestimmung der \mathfrak{B} -glatten Werte $\sigma(x)$ durch Siebe mit p	383
7.10 Laufzeiten ausgewählter Faktorisierungsalgorithmen	384
7.11 Faktorisierung einiger Zahlen RSA- d	385
7.12 Berechnung der Kettenbruchentwicklung von $101/37$	391
7.13 Berechnung der Konvergenten in Wieners Angriff	392
 8.1 Berechnung der Primitivwurzeln von 17	406
8.2 Berechnung von $\alpha = 12^{10} \bmod 17$ beim Diffie–Hellman-Protokoll .	407
8.3 Berechnung der Listen \mathcal{L}_1 und \mathcal{L}_2 für Shanks’ Algorithmus	411
8.4 Verifikation der Signatur in ElGamals digitalem Signatur-Protokoll .	416
8.5 Instanzen des Bit-Problems des diskreten Logarithmus in Beispiel 8.10	418

Abbildungsverzeichnis

1.1	Ein typisches kryptographisches Szenario	1
2.1	Euklidischer Algorithmus	12
2.2	Erweiterter Euklidischer Algorithmus	14
2.3	Syntaxbaum für eine Ableitung in Beispiel 2.8	22
2.4	Ein deterministischer und ein nichtdeterministischer endlicher Automat	24
2.5	Eine Turingmaschine	27
2.6	Drei Graphen: G ist isomorph zu H , aber nicht zu F	51
3.1	2-SAT ist NL-vollständig	97
3.2	Cook-Reduktion	101
3.3	$3\text{-SAT} \leq_m^p \text{IS}$	107
3.4	$3\text{-SAT} \leq_m^p \text{3-Colorability}$	109
3.5	$\text{3-Colorability} \leq_m^p \text{DNP}$	111
3.6	$3\text{-SAT} \leq_m^p \text{3-DM}$	114
3.7	Die Tantrix TM -Teile	134
3.8	Sämtliche 56 Tantrix TM -Teile mit vier Farben	134
3.9	Ein Tantrix TM -Puzzle und seine Lösung	135
3.10	Sämtliche 14 Tantrix TM -Teile mit drei Farben	135
3.11	Sämtliche 8 Tantrix TM -Teile mit zwei Farben	136
4.1	Ein lineares Rückkopplungsschieberegister	172
5.1	Boolesche Hierarchie über NP (Hasse-Diagramm)	201
5.2	Boolesche Hierarchie über NP (Venn-Diagramm)	202
5.3	Exact-5-DNP ist DP-vollständig	205
5.4	Baumartige Struktur S_i in der Guruswami–Khanna-Reduktion	211
5.5	Grundgitter	211
5.6	Verbindungsmuster zwischen den Grundgittern	212
5.7	Verbindung der „Blätter“ wegen <i>derselben</i> Zeile	213

5.8	Verbindung der „Blätter“ wegen <i>verschiedener</i> Zeilen	213
5.9	Präfixsuche nach dem kleinsten Graphisomorphismus	218
5.10	Polynomialzeit-Hierarchie (Hasse-Diagramm)	221
5.11	Polynomialzeit-Hierarchie (Venn-Diagramm)	222
5.12	„Easy-Hard“-Technik: Präfixsuche nach dem kleinsten harten Wort .	248
5.13	Ein akzeptierender alternierender Teilbaum für eine ATM	251
5.14	ATMs als ein paralleles Berechnungsmodell	252
5.15	$\text{ATIME}(t) \subseteq \text{DSPACE}(t)$: Initialisierung der DTM N	254
5.16	$\text{ATIME}(t) \subseteq \text{DSPACE}(t)$: Simulation der ATM M	254
5.17	$\text{ATIME}(t) \subseteq \text{DSPACE}(t)$: Auswertung der ATM M	255
5.18	$\text{DTIME}(2^{\text{Lin}(s)}) \subseteq \text{ASPACE}(s)$: Kopfbewegung der DTM M	259
5.19	$\text{DTIME}(2^{\text{Lin}(s)}) \subseteq \text{ASPACE}(s)$: Berechnungsbaum der ATM N	260
5.20	Die Low- und die High-Hierarchie in NP	264
6.1	Backtracking-Algorithmus für 3-SAT	297
6.2	Algorithmus RANDOM-SAT	299
6.3	Übergangsgraph eines stochastischen Automaten für RANDOM-SAT	300
6.4	Illustration der ersten Aussage von Lemma 6.20	317
6.5	Algorithmus LERC	334
6.6	NP-Maschine N für die Orakelmenge A	337
6.7	FP^{SPP} -Algorithmus zur Berechnung eines starken Generators	338
6.8	Probabilistische, Arthur-Merlin- und Zählklassen und die PH	347
7.1	RSA-Protokoll	351
7.2	Der <i>Square-and-Multiply</i> -Algorithmus	353
7.3	Digitale Signaturen mit RSA	355
7.4	Probedivision zur Lösung des Primzahl-Problems	357
7.5	Fermat-Test	360
7.6	Miller–Rabin-Test	363
7.7	Solovay–Strassen-Test	370
7.8	Pollards $(p - 1)$ -Faktorisierungsalgorithmus	377
8.1	Schlüsseltausch-Protokoll von Diffie und Hellman	405
8.2	Shanks’ „Baby-Step Giant-Step“-Algorithmus	410
8.3	ElGamals Public-Key-Kryptosystem	413
8.4	ElGamals digitales Signatur-Schema	415
8.5	Rabins Public-Key-Kryptosystem	425
8.6	Faktorisieren eines Rabin-Moduls	429
8.7	Arthurs Labyrinth	432
8.8	Zero-Knowledge-Protokoll für Graphisomorphie	434
8.9	Simulation des Zero-Knowledge-Protokolls für Graphisomorphie .	438
8.10	Zero-Knowledge-Identifikationsschema von Fiat und Shamir . . .	439
8.11	Public-Key-Kryptosystem von Merkle und Hellman	441
8.12	Rivest–Sherman-Protokoll für den Schlüsseltausch	443
8.13	Paula und Ella benutzen ein Protokoll ohne öffentliche Schlüssel .	455

Literaturverzeichnis

- [ACG⁺03] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, M. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation*. Springer-Verlag, second edition, 2003.
- [AD97] M. Ajtai and C. Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 284–293. ACM Press, 1997.
- [Adl83] L. Adleman. On breaking the iterated Merkle-Hellman public key cryptosystem. In *Proceedings of the 15th ACM Symposium on Theory of Computing*, pages 402–412. ACM Press, 1983.
- [AH87] L. Adleman and M. Huang. Recognizing primes in random polynomial time. In *Proceedings of the 19th ACM Symposium on Theory of Computing*, pages 462–469. ACM Press, May 1987.
- [AH92a] L. Adleman and M. Huang. *Primality Testing and Abelian Varieties over Finite Fields*. Springer-Verlag Lecture Notes in Mathematics #1512, Berlin, Heidelberg, New York, 1992.
- [AH92b] E. Allender and L. Hemachandra. Lower bounds for the low hierarchy. *Journal of the ACM*, 39(1):234–251, 1992.
- [AHH⁺93] V. Arvind, Y. Han, L. Hemachandra, J. Köbler, A. Lozano, M. Mundhenk, M. Ogiwara, U. Schöning, R. Silvestri, and T. Thierauf. Reductions to sets of low information content. In K. Ambos-Spies, S. Homer, and U. Schöning, editors, *Complexity Theory*, pages 1–45. Cambridge University Press, 1993.
- [AHS93] K. Ambos-Spies, S. Homer, and U. Schöning, editors. *Complexity Theory: Current Research*. Cambridge University Press, 1993.
- [Ajt96] M. Ajtai. Generating hard instances of lattice problems. In *Proceedings of the 28th ACM Symposium on Theory of Computing*, pages 99–108. ACM Press, 1996.
- [AK02a] V. Arvind and J. Köbler. New lowness results for ZPP(NP) and other complexity classes. *Journal of Computer and System Sciences*, 65(2):257–277, 2002.
- [AK02b] V. Arvind and P. Kurur. Graph isomorphism is in SPP. In *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science*, pages 743–750. IEEE Computer Society Press, November 2002.
- [AKS02] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. Unpublished manuscript, August 2002.

- [AKS04] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004.
- [All85] E. Allender. Invertible functions, 1985. PhD thesis, Georgia Institute of Technology.
- [All86] E. Allender. The complexity of sparse sets in P. In *Proceedings of the 1st Structure in Complexity Theory Conference*, pages 1–11. Springer-Verlag *Lecture Notes in Computer Science* #223, June 1986.
- [All88] E. Allender. Isomorphisms and 1-L reductions. *Journal of Computer and System Sciences*, 36(6):336–350, 1988.
- [All91] E. Allender. Limitations of the upward separation technique. *Mathematical Systems Theory*, 24(1):53–67, 1991.
- [ALM⁺98] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and intractability of approximation problems. *Journal of the ACM*, 45(3):501–555, May 1998.
- [AM77] L. Adleman and K. Manders. Reducibility, randomness, and intractability. In *Proceedings of the 9th ACM Symposium on Theory of Computing*, pages 151–153. ACM Press, 1977.
- [Ant00] H. Anton. *Elementary Linear Algebra*. John Wiley and Sons, New York, eighth edition, 2000.
- [AR88] E. Allender and R. Rubinstein. P-printable sets. *SIAM Journal on Computing*, 17(6):1193–1202, 1988.
- [Aro94] S. Arora. *Probabilistic Checking of Proofs and Hardness of Approximation Problems*. PhD thesis, University of California at Berkeley, November 1994. Revised version available as Princeton University Technical Report CS-TR-476-94.
- [Arr63] K. Arrow. *Social Choice and Individual Values*. John Wiley and Sons, 1951 (revised edition 1963).
- [AS92] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. In *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science*, pages 2–11, 1992.
- [AS98] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70–122, January 1998. Preliminary version appears as [AS92].
- [AT99] V. Arvind and J. Torán. Sparse sets, approximable sets, and parallel queries to NP. *Information Processing Letters*, 69:181–188, February 1999.
- [AT01] M. Agrawal and T. Thierauf. The formula isomorphism problem. *SIAM Journal on Computing*, 30(3):990–1009, June 2001.
- [Bab85] L. Babai. Trading group theory for randomness. In *Proceedings of the 17th ACM Symposium on Theory of Computing*, pages 421–429. ACM Press, April 1985.
- [Bal85] J. Balcazár. Simplicity, relativizations and nondeterminism. *SIAM Journal on Computing*, 14(1):148–157, 1985.
- [Bal90] J. Balcazár. Self-reducibility. *Journal of Computer and System Sciences*, 41(3):367–388, 1990.
- [Bau00a] F. Bauer. *Decrypted Secrets: Methods and Maxims of Cryptology*. Springer-Verlag, second edition, 2000.
- [Bau00b] F. Bauer. *Entzifferte Geheimnisse: Methoden und Maximen der Kryptologie*. Springer-Verlag, third edition, 2000. In German.
- [BBJ⁺89] A. Bertoni, D. Bruschi, D. Joseph, M. Sitharam, and P. Young. Generalized boolean hierarchies and boolean hierarchies over RP. In *Proceedings of the 7th Conference on Fundamentals of Computation Theory*, pages 35–46. Springer-Verlag *Lecture Notes in Computer Science* #380, August 1989.

- [BBS86a] J. Balcázar, R. Book, and U. Schöning. The polynomial-time hierarchy and sparse oracles. *Journal of the ACM*, 33(3):603–617, 1986.
- [BBS86b] J. Balcázar, R. Book, and U. Schöning. Sparse sets, lowness and highness. *SIAM Journal on Computing*, 15(3):739–746, 1986.
- [BC89] G. Brassard and C. Crépeau. Sorting out zero-knowledge. In *Advances in Cryptology – EUROCRYPT ’89*, pages 181–191. Springer-Verlag *Lecture Notes in Computer Science #434*, April 1989.
- [BC93] D. Bovet and P. Crescenzi. *Introduction to the Theory of Complexity*. Prentice Hall, 1993.
- [BCKT94] N. Bshouty, R. Cleve, S. Kannan, and C. Tamon. Oracles and queries that are sufficient for exact learning. In *Proceedings of the 7th ACM Conference on Computational Learning Theory*, pages 130–139. ACM Press, 1994.
- [BCO93] R. Beigel, R. Chang, and M. Ogiwara. A relationship between difference hierarchies and relativized polynomial hierarchies. *Mathematical Systems Theory*, 26(3):293–310, 1993.
- [BCS92] D. Bovet, P. Crescenzi, and R. Silvestri. A uniform approach to define complexity classes. *Theoretical Computer Science*, 104(2):263–283, 1992.
- [BD00] D. Boneh and G. Durfee. Cryptanalysis of RSA with private key d less than $N^{0.292}$. *IEEE Transactions on Information Theory*, IT-46, 2000.
- [BDG90] J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity II*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1990.
- [BDG95] J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, second edition, 1995.
- [Bei91a] R. Beigel. Bounded queries to SAT and the boolean hierarchy. *Theoretical Computer Science*, 84(2):199–223, 1991.
- [Bei91b] R. Beigel. Relativized counting classes: Relations among thresholds, parity, and mods. *Journal of Computer and System Sciences*, 42(1):76–96, 1991.
- [BEM⁺07] D. Bruß, G. Erdélyi, T. Meyer, T. Riege, and J. Rothe. Quantum cryptography: A survey. *ACM Computing Surveys*, 39(2):article 6, 27 pp., 2007.
- [Ber77] L. Berman. *Polynomial Reducibilities and Complete Sets*. PhD thesis, Cornell University, Ithaca, NY, 1977.
- [Ber78] P. Berman. Relationship between density and deterministic complexity of NP-complete languages. In *Proceedings of the 5th International Colloquium on Automata, Languages, and Programming*, pages 63–71. Springer-Verlag *Lecture Notes in Computer Science #62*, 1978.
- [Ber97] A. Berthiaume. Quantum computation. In L. Hemaspaandra and A. Selman, editors, *Complexity Theory Retrospective II*, pages 23–51. Springer-Verlag, 1997.
- [Ber04] D. Bernstein. Distinguishing prime numbers from composite numbers: The state of the art in 2004. Manuscript at <http://cr.yp.to/primetests.html>, 2004.
- [Beu94] A. Beutelspacher. *Cryptology*. Spectrum series. Mathematical Association of America, 1994.
- [Beu02] A. Beutelspacher. *Kryptologie*. Vieweg, 6th edition, 2002. In German.
- [BF98] H. Buhrman and L. Fortnow. Two queries. In *Proceedings of the 13th Annual IEEE Conference on Computational Complexity*, pages 13–19. IEEE Computer Society Press, May 1998.
- [BFH78] G. Brassard, S. Fortune, and J. Hopcroft. A note on cryptography and $\text{NP} \cap \text{coNP} = \text{P}$. Technical Report TR-338, Department of Computer Science, Cornell University, Ithaca, NY, April 1978.
- [BG70] R. Book and S. Greibach. Quasi-realtime languages. *Mathematical Systems Theory*, 4:97–111, 1970.

- [BG82] A. Blass and Y. Gurevich. On the unique satisfiability problem. *Information and Control*, 55(1–3):80–88, 1982.
- [BG92] R. Beigel and J. Gill. Counting classes: Thresholds, parity, mods, and fewness. *Theoretical Computer Science*, 103(1):3–23, 1992.
- [BGS75] T. Baker, J. Gill, and R. Solovay. Relativizations of the P=?NP question. *SIAM Journal on Computing*, 4(4):431–442, 1975.
- [BH77] L. Berman and J. Hartmanis. On isomorphisms and density of NP and other complete sets. *SIAM Journal on Computing*, 6(2):305–322, 1977.
- [BH88] S. Buss and L. Hay. On truth-table reducibility to SAT and the difference hierarchy over NP. In *Proceedings of the 3rd Structure in Complexity Theory Conference*, pages 224–233. IEEE Computer Society Press, June 1988.
- [BH91] S. Buss and L. Hay. On truth-table reducibility to SAT. *Information and Computation*, 91(1):86–102, March 1991.
- [BH04] M. Bauland and E. Hemaspaandra. Isomorphic implication. Technical Report cs.CC/0412062, ACM Computing Research Repository, December 2004. Revised, April 2005.
- [BHHR99] A. Beygelzimer, L. Hemaspaandra, C. Homan, and J. Rothe. One-way functions in worst-case cryptography: Algebraic and security properties are on the house. *SIGACT News*, 30(4):25–40, December 1999.
- [BHL95] H. Buhrman, E. Hemaspaandra, and L. Longpré. SPARSE reduces conjunctively to TALLY. *SIAM Journal on Computing*, 24(3):673–681, June 1995.
- [BHR00] B. Borchert, L. Hemaspaandra, and J. Rothe. Restrictive acceptance suffices for equivalence problems. *London Mathematical Society Journal of Computation and Mathematics*, 3:86–95, March 2000.
- [BHRV02] E. Böhler, E. Hemaspaandra, S. Reith, and H. Vollmer. Equivalence and isomorphism of boolean constraint satisfaction. In *Proceedings of the 16th Annual Conference of the EACSL (CSL 2002)*, pages 412–426. Springer-Verlag *Lecture Notes in Computer Science* #2471, 2002.
- [BHRV04] E. Böhler, E. Hemaspaandra, S. Reith, and H. Vollmer. The complexity of boolean constraint isomorphism. In *Proceedings of the 21st Annual Symposium on Theoretical Aspects of Computer Science*, pages 164–175. Springer-Verlag *Lecture Notes in Computer Science* #2996, 2004.
- [BHW91] R. Beigel, L. Hemachandra, and G. Wechsung. Probabilistic polynomial time is closed under parity reductions. *Information Processing Letters*, 37(2):91–94, 1991.
- [BHZ87] R. Boppana, J. Hästad, and S. Zachos. Does co-NP have short interactive proofs? *Information Processing Letters*, 25(2):127–132, 1987.
- [BJY90] D. Bruschi, D. Joseph, and P. Young. Strong separations for the boolean hierarchy over RP. *International Journal of Foundations of Computer Science*, 1(3):201–218, 1990.
- [BK04] T. Brueggemann and W. Kern. An improved deterministic local search algorithm for 3-SAT. *Theoretical Computer Science*, 329(1-3):303–313, 2004.
- [Bla58] D. Black. *The Theory of Committees and Elections*. Cambridge University Press, 1958.
- [Ble98] D. Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In *Advances in Cryptology – CRYPTO '98*, pages 1–12. Springer-Verlag *Lecture Notes in Computer Science* #1462, August 1998.
- [BLO83] E. Brickell, J. Lagarias, and A. Odlyzko. Evaluation of the Adleman attack on multiply iterated knapsack cryptosystems. In *Advances in Cryptology – CRYPTO '83*, pages 39–42, New York, 1983. Plenum Press.

- [BLS99] A. Brandstädt, V. Le, and J. Spinrad. *Graph Classes: A Survey*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1999.
- [Blu67] M. Blum. A machine-independent theory of the complexity of recursive functions. *Journal of the ACM*, 14(2):322–336, April 1967.
- [BM88] L. Babai and S. Moran. Arthur-Merlin games: A randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36(2):254–276, 1988.
- [BM04] J. Blömer and A. May. A generalized Wiener attack on RSA. In *7th International Workshop on Practice and Theory in Public-Key Cryptography*, pages 1–13. Springer-Verlag Lecture Notes in Computer Science #2947, 2004.
- [BO92] E. Brickell and A. Odlyzko. Cryptanalysis, a survey of recent results. In G. Simmons, editor, *Contemporary Cryptology: The Science of Information Integrity*, pages 501–540. IEEE Computer Society Press, 1992.
- [Bon85] M. Bonuccelli. Dominating sets and dominating number of circular arc graphs. *Discrete Applied Mathematics*, 12:203–213, 1985.
- [Bon99] D. Boneh. Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS*, 46(2):203–213, February 1999.
- [Boo74] R. Book. Tally languages and complexity classes. *Information and Control*, 26(2):186–193, 1974.
- [Bor89] J. Borges. The library of babel. In J. Borges and A. Kerrigan, editors, *Ficciones*, pages 79–88. Grove Press, 1989.
- [BORW88] R. Book, P. Orponen, D. Russo, and O. Watanabe. Lowness properties of sets in the exponential-time hierarchy. *SIAM Journal on Computing*, 17(3):504–516, 1988.
- [BR88] J. Balcázar and D. Russo. Immunity and simplicity in relativizations of probabilistic complexity classes. *R.A.I.R.O. Theoretical Informatics and Applications*, 22(2):227–244, 1988.
- [BR93] B. Borchert and D. Ranjan. The subfunction relations are Σ_2^P -complete. Technical Report MPI-I-93-121, Max-Planck Institut Saarbrücken, Saarbrücken, Germany, 1993.
- [BR07] D. Baumeister and J. Rothe. Satisfiability parsimoniously reduces to the TantrixTM rotation puzzle problem. In *Proceedings of the 5th Conference on Machines, Computations and Universality*, pages 134–145. Springer-Verlag Lecture Notes in Computer Science #4664, September 2007.
- [BR08] D. Baumeister and J. Rothe. The three-color and two-color TantrixTM rotation puzzle problems are NP-complete via parsimonious reductions. In *Proceedings of the 2nd International Conference on Language and Automata Theory and Applications*. Springer-Verlag Lecture Notes in Computer Science, March 2008. To appear. Full version available as Technical Report cs.CC/0711.1827, ACM Computing Research Repository (CoRR), November 2007.
- [Bra79] G. Brassard. A note on the complexity of cryptography. *IEEE Transactions on Information Theory*, 25(2):232–233, 1979.
- [Bri83] E. Brickell. Solving low density knapsacks. In *Advances in Cryptology – CRYPTO '83*, pages 25–37, New York, 1983. Plenum Press.
- [Bri85] E. Brickell. Breaking iterated knapsacks. In *Advances in Cryptology – CRYPTO '84*, pages 342–358. Springer-Verlag Lecture Notes in Computer Science #196, 1985.

- [BRS91] R. Beigel, N. Reingold, and D. Spielman. PP is closed under intersection. In *Proceedings of the 23rd ACM Symposium on Theory of Computing*, pages 1–9. ACM Press, May 1991.
- [BRS98] B. Borchert, D. Ranjan, and F. Stephan. On the computational complexity of some classical equivalence relations on boolean functions. *Mathematical Systems Theory*, 31(6):679–693, 1998.
- [Bru92] D. Bruschi. Strong separations of the polynomial hierarchy with oracles: Constructive separations by immune and simple sets. *Theoretical Computer Science*, 102(2):215–252, 1992.
- [BST93a] H. Buhrman, E. Spaan, and L. Torenvliet. Bounded reductions. In K. Ambos-Spies, S. Homer, and U. Schöning, editors, *Complexity Theory*, pages 83–99. Cambridge University Press, 1993.
- [BST93b] H. Buhrmann, E. Spaan, and L. Torenvliet. The relative power of logspace and polynomial time reductions. *Computational Complexity*, 3(3):231–244, 1993.
- [BSW01] A. Beutelspacher, J. Schwenk, and K. Wolfenstetter. *Moderne Verfahren der Kryptographie*. Vieweg, 4th edition, 2001. In German.
- [BT96] H. Buhrman and L. Torenvliet. P-selective self-reducible sets: A new characterization of P. *Journal of Computer and System Sciences*, 53(2):210–217, 1996.
- [BTT89a] J. Bartholdi III, C. Tovey, and M. Trick. The computational difficulty of manipulating an election. *Social Choice and Welfare*, 6(3):227–241, 1989.
- [BTT89b] J. Bartholdi III, C. Tovey, and M. Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6(2):157–165, 1989.
- [BTT92] J. Bartholdi III, C. Tovey, and M. Trick. How hard is it to control an election? *Mathematical Comput. Modelling*, 16(8/9):27–40, 1992.
- [Buc01a] J. Buchmann. *Einführung in die Kryptographie*. Springer-Verlag, second edition, 2001. In German.
- [Buc01b] J. Buchmann. *Introduction to Cryptography*. Undergraduate Texts in Mathematics. Springer-Verlag, 2001.
- [BvHT93] H. Buhrman, P. van Helden, and L. Torenvliet. P-selective self-reducible sets: A new characterization of P. In *Proceedings of the 8th Structure in Complexity Theory Conference*, pages 44–51. IEEE Computer Society Press, May 1993.
- [Cai99] J. Cai. Some recent progress on the complexity of lattice problems. In *Proceedings of the 14th Annual IEEE Conference on Computational Complexity*, pages 158–179. IEEE Computer Society Press, May 1999.
- [Cai01] J. Cai. $S_2^P \subseteq ZPP^{NP}$. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, pages 620–629. IEEE Computer Society Press, October 2001.
- [Can96] R. Canetti. More on BPP and the polynomial-time hierarchy. *Information Processing Letters*, 57(5):237–241, 1996.
- [CCHO03] J. Cai, V. Chakaravarthy, L. Hemaspaandra, and M. Ogihara. Competing provers yield improved Karp–Lipton collapse results. In *Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science*, pages 535–546. Springer-Verlag *Lecture Notes in Computer Science #2607*, 2003.
- [CGH⁺88] J. Cai, T. Gundersmann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung. The boolean hierarchy I: Structural properties. *SIAM Journal on Computing*, 17(6):1232–1252, 1988.
- [CGH⁺89] J. Cai, T. Gundersmann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung. The boolean hierarchy II: Applications. *SIAM Journal on Computing*, 18(1):95–111, 1989.

- [CH86] J. Cai and L. Hemachandra. The boolean hierarchy: Hardware over NP. In *Proceedings of the 1st Structure in Complexity Theory Conference*, pages 105–124. Springer-Verlag *Lecture Notes in Computer Science* #223, June 1986.
- [CH91] M. Cohen and J. Hashkes. A system for controlling access to broadcast transmissions. European Patent Application 0 428252 A2, May 1991.
- [Cha91] R. Chang. *On the Structure of NP Computations under Boolean Operators*. PhD thesis, Cornell University, Ithaca, NY, 1991.
- [CHV92] J. Cai, L. Hemachandra, and J. Vyskoč. Promise problems and access to unambiguous computation. In *Proceedings of the 17th Symposium on Mathematical Foundations of Computer Science*, pages 162–171. Springer-Verlag *Lecture Notes in Computer Science* #629, August 1992.
- [CHV93] J. Cai, L. Hemachandra, and J. Vyskoč. Promises and fault-tolerant database access. In K. Ambos-Spies, S. Homer, and U. Schöning, editors, *Complexity Theory*, pages 101–146. Cambridge University Press, 1993.
- [CK96] R. Chang and J. Kadin. The boolean hierarchy and the polynomial hierarchy: A closer connection. *SIAM Journal on Computing*, 25(2):340–354, April 1996.
- [CKR95] R. Chang, J. Kadin, and P. Rohatgi. On unique satisfiability and the threshold behavior of randomized reductions. *Journal of Computer and System Sciences*, 50(3):359–373, 1995.
- [CKS81] A. Chandra, D. Kozen, and L. Stockmeyer. Alternation. *Journal of the ACM*, 26(1), 1981.
- [CLRS01] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, second edition, 2001.
- [CLS03] V. Conitzer, J. Lang, and T. Sandholm. How many candidates are needed to make elections hard to manipulate? In *Proceedings of the 9th Conference on Theoretical Aspects of Rationality and Knowledge*, pages 201–214. ACM Press, 2003.
- [CM87] J. Cai and G. Meyer. Graph minimal uncolorability is D^P -complete. *SIAM Journal on Computing*, 16(2):259–277, April 1987.
- [Cob64] A. Cobham. The intrinsic computational difficulty of functions. In *Proceedings of the 1964 International Congress for Logic Methodology and Philosophy of Science*, pages 24–30. North Holland, 1964.
- [Con85] J.-A.-N. de Caritat, Marquis de Condorcet. *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix*. 1785. Facsimile reprint of original published in Paris, 1972, by the Imprimerie Royale. English translation appears in I. McLean and A. Urken, *Classics of Social Choice*, University of Michigan Press, 1995, pages 91–112.
- [Coo71] S. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd ACM Symposium on Theory of Computing*, pages 151–158. ACM Press, 1971.
- [Coo74] S. Cook. An observation on time-storage trade off. *Journal of Computer and System Sciences*, 9:308–316, 1974.
- [Cop97] D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, 10(4):233–260, 1997.
- [CPS99] J. Cai, A. Pavan, and D. Sivakumar. On the hardness of permanent. In *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science*, pages 90–99. Springer-Verlag *Lecture Notes in Computer Science* #1563, 1999.

- [CR88] B. Chor and R. Rivest. A knapsack-type public key cryptosystem based on arithmetic in finite fields. *IEEE Transactions on Information Theory*, IT-45(5):901–909, 1988.
- [CS92] J. Castro and C. Seara. Characterizations of some complexity classes between Θ_2^P and Δ_2^P . In *Proceedings of the 9th Annual Symposium on Theoretical Aspects of Computer Science*, pages 305–317. Springer-Verlag *Lecture Notes in Computer Science* #577, February 1992.
- [CS97] D. Coppersmith and A. Shamir. Lattice attacks on NTRU. In *Advances in Cryptology – EUROCRYPT ’97*, pages 52–61. Springer-Verlag *Lecture Notes in Computer Science* #1233, 1997.
- [CS99] J. Cai and D. Sivakumar. Sparse hard sets for P: Resolution of a conjecture of Hartmanis. *Journal of Computer and System Sciences*, 58(2):280–296, April 1999.
- [CS00] J. Cai and D. Sivakumar. Resolution of Hartmanis’ conjecture for NL-hard sparse sets. *Theoretical Computer Science*, 240(2):257–269, 2000.
- [CS02] V. Conitzer and T. Sandholm. Complexity of manipulating elections with few candidates. In *Proceedings of the 18th National Conference on Artificial Intelligence*, pages 314–319. AAAI Press, 2002.
- [CT95] Z. Chen and S. Toda. The complexity of selecting maximal solutions. *Information and Computation*, 119:231–239, 1995.
- [CT04] J. Cai and R. Threlfall. A note on quadratic residuosity and UP. *Information Processing Letters*, 92(3):127–131, 2004.
- [CW79] J. Carter and M. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18:143–154, 1979.
- [CW04] J. Cai and O. Watanabe. Relativized collapsing between BPP and PH under stringent oracle access. *Information Processing Letters*, 90(3):147–154, 2004.
- [Del07] T. Delbrouck. Vom Fachmann für Kenner: Wie wahr. *Titanic*, 12:36, 2007.
- [DGH⁺02] E. Dantsin, A. Goerdt, E. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, and U. Schöning. A deterministic $(2 - 2/(k+1))^n$ algorithm for k-SAT based on local search. *Theoretical Computer Science*, 289(1):69–83, October 2002.
- [DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.
- [Die04] M. Dietzfelbinger. *Primality Testing in Polynomial Time: From Randomized Algorithms to “PRIMES is in P”*. Tutorial. Springer-Verlag *Lecture Notes in Computer Science* #3000, Berlin, Heidelberg, New York, 2004.
- [Dix81] J. Dixon. Asymptotically fast factorization of integers. *Mathematics of Computation*, 36:255–260, 1981.
- [DK00] D. Du and K. Ko. *Theory of Computational Complexity*. John Wiley and Sons, 2000.
- [DKNS01] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proceedings of the 10th International World Wide Web Conference*, pages 613–622. ACM Press, 2001.
- [Dod76] C. Dodgson. A method of taking votes on more than two issues. Pamphlet printed by the Clarendon Press, Oxford, and headed “not yet published” (see the discussions in [MU95, Bla58], both of which reprint this paper), 1876.
- [DR01] J. Daemen and V. Rijmen. *The Design of Rijndael*. Springer-Verlag, 2001.
- [Edm65] J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

- [EG93] T. Eiter and G. Gottlob. Propositional circumscription and extended closed-world reasoning are Π_2^P -complete. *Theoretical Computer Science*, 114(2):231–245, 1993. Addendum appears in the same journal, 118(2):315, 1993.
- [EG97] T. Eiter and G. Gottlob. The complexity class Θ_2^P : Recent results and applications. In *Proceedings of the 11th Conference on Fundamentals of Computation Theory*, pages 1–18. Springer-Verlag *Lecture Notes in Computer Science* #1279, September 1997.
- [EHRS07] G. Erdélyi, L. Hemaspaandra, J. Rothe, and H. Spakowski. On approximating optimal weighted lobbying, and frequency of correctness versus average-case polynomial time. In *Proceedings of the 16th International Symposium on Fundamentals of Computation Theory*, pages 300–311. Springer-Verlag *Lecture Notes in Computer Science* #4639, August 2007.
- [EHTY92] D. Eppstein, L. Hemachandra, J. Tisdall, and B. Yener. Simultaneous strong separations of probabilistic and unambiguous complexity classes. *Mathematical Systems Theory*, 25(1):23–36, 1992.
- [ElG85] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469–472, 1985.
- [ENR08a] G. Erdélyi, M. Nowak, and J. Rothe. Sincere-strategy preference-based approval voting broadly resists control. In *Proceedings of the 33rd International Symposium on Mathematical Foundations of Computer Science*, pages 311–322. Springer-Verlag *Lecture Notes in Computer Science* #5162, August 2008.
- [ENR08b] G. Erdélyi, M. Nowak, and J. Rothe. Sincere-strategy preference-based approval voting fully resists constructive control and broadly resists destructive control. Technical Report cs.GT/0806.0535, ACM Computing Research Repository (CoRR), June 2008.
- [Esp01] W. Espelage. *Bewegungsminimierung in der Förderband-Flow-Shop-Verarbeitung*. PhD thesis, Heinrich-Heine-Universität Düsseldorf, Düsseldorf, Germany, 2001. In German.
- [ESY84] S. Even, A. Selman, and Y. Yacobi. The complexity of promise problems with applications to public-key cryptography. *Information and Control*, 61(2):159–173, 1984.
- [EW00] W. Espelage and E. Wanke. Movement optimization in flow shop processing with buffers. *Mathematical Methods of Operations Research*, 51(3):495–513, 2000.
- [EW01] W. Espelage and E. Wanke. A 3-approximation algorithmus for movement minimization in conveyor flow shop processing. In *Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science*, pages 363–374. Springer-Verlag *Lecture Notes in Computer Science* #2136, 2001.
- [EW03] W. Espelage and E. Wanke. Movement minimization for unit distances in conveyor flow shop processing. *Mathematical Methods of Operations Research*, 57(2):172–206, 2003.
- [EY80] S. Even and Y. Yacobi. Cryptocomplexity and NP-completeness. In *Proceedings of the 7th International Colloquium on Automata, Languages, and Programming*, pages 195–207. Springer-Verlag *Lecture Notes in Computer Science*, 1980.
- [Far84] M. Farber. Domination, independent domination, and duality in strongly chordal graphs. *Discrete Applied Mathematics*, 7:115–130, 1984.
- [Fei92] J. Feigenbaum. Overview of interactive proof systems and zero-knowledge. In G. Simmons, editor, *Contemporary Cryptology: The Science of Information Integrity*, pages 423–439. IEEE Computer Society Press, 1992.
- [Fel68] W. Feller. *Introduction to Probability Theory and its Applications*, volume 1. John Wiley and Sons, 1968.

- [FFK92] S. Fenner, L. Fortnow, and S. Kurtz. An oracle relative to which the isomorphism conjecture holds. In *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science*, pages 30–39. IEEE Computer Society Press, October 1992.
- [FFK94] S. Fenner, L. Fortnow, and S. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48(1):116–148, 1994.
- [FFNR96] S. Fenner, L. Fortnow, A. Naik, and J. Rogers. Inverting onto functions. In *Proceedings of the 11th Annual IEEE Conference on Computational Complexity*, pages 213–222. IEEE Computer Society Press, May 1996.
- [FFNR03] S. Fenner, L. Fortnow, A. Naik, and J. Rogers. Inverting onto functions. *Information and Computation*, 186(1):90–103, 2003.
- [FFS88] U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, 1988.
- [FHH06] P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. The complexity of bribery in elections. In *Proceedings of the 21st National Conference on Artificial Intelligence*, pages 641–646. AAAI Press, July 2006.
- [FHHR] P. Faliszewski, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. A richer understanding of the complexity of election systems. In S. Ravi and S. Shukla, editors, *Fundamental Problems in Computing: Essays in Honor of Professor Daniel J. Rosenkrantz*. Springer. To appear. Available as Technical Report cs.GT/0609112, ACM Computing Research Repository (CoRR), September 2006.
- [FHHR07] P. Faliszewski, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Llull and Copeland voting broadly resist bribery and control. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, pages 724–730. AAAI Press, July 2007.
- [FHHR08] P. Faliszewski, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Copeland voting fully resists constructive control. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management*, pages 165–176. Springer-Verlag *Lecture Notes in Computer Science* #5034, June 2008.
- [FHKS02] U. Feige, M. Halldórsson, G. Kortsarz, and A. Srinivasan. Approximating the domatic number. *SIAM Journal on Computing*, 32(1):172–195, 2002.
- [Fis77] P. Fishburn. Condorcet social choice functions. *SIAM Journal on Applied Mathematics*, 33(3):469–489, 1977.
- [FK92] M. Fellows and N. Koblitz. Self-witnessing polynomial-time complexity and prime factorization. *Designs, Codes and Cryptography*, 2(3):231–235, 1992.
- [For79] S. Fortune. A note on sparse complete sets. *SIAM Journal on Computing*, 8(3):431–433, 1979.
- [For97] L. Fortnow. Counting complexity. In L. Hemaspaandra and A. Selman, editors, *Complexity Theory Retrospective II*, pages 81–107. Springer-Verlag, 1997.
- [FR96] L. Fortnow and N. Reingold. PP is closed under truth-table reductions. *Information and Computation*, 124(1):1–6, 1996.
- [FS86] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology – CRYPTO ’86*, pages 186–194. Springer-Verlag *Lecture Notes in Computer Science* #263, 1986.
- [FSS84] M. Furst, J. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.
- [FY96] L. Fortnow and T. Yamakami. Generic separations. *Journal of Computer and System Sciences*, 52(1):191–197, 1996.
- [Gai39] H. Gaines. *Cryptoanalysis*. Dover Publications, New York, 1939.

- [Gas02] W. Gasarch. The P =? NP poll. *SIGACT News*, 33(2):34–47, 2002.
- [GH92] K. Ganesan and S. Homer. Complete problems and strong polynomial reducibilities. *SIAM Journal on Computing*, 21(4):733–742, August 1992.
- [GH96] J. Goldsmith and S. Homer. Scalability and the isomorphism problem. *Information Processing Letters*, 57(3):137–143, 1996.
- [GHL99] A. Gál, S. Halevi, R. Lipton, and E. Petrank. Computing from partial solutions. In *Proceedings of the 14th Annual IEEE Conference on Computational Complexity*, pages 34–45. IEEE Computer Society Press, May 1999.
- [Gib75] J. Gibbs. On the equilibrium of heterogeneous substances. *Transactions of the Connecticut Academy of Arts and Sciences*, 3:108–248, 1875. Also in *The American Journal of Science and Arts*, vol. 16, no. 96, 1978. Reprinted in *The Scientific Papers of J. Willard Gibbs, Vol. I: Thermodynamics*, Longman, 1906; Dover, 1961, pp. 55–353.
- [Gil77] J. Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing*, 6(4):675–695, 1977.
- [GJ79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.
- [GJ86] J. Goldsmith and D. Joseph. Three results on the polynomial isomorphism of complete sets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pages 390–397, 1986.
- [GJS76] M. Garey, D. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.
- [GJY87] J. Goldsmith, D. Joseph, and P. Young. Self-reducible, P-selective, near-testable, and P-cheatable sets: The effect of internal structure on the complexity of a set. In *Proceedings of the 2nd Structure in Complexity Theory Conference*, pages 50–59, 1987.
- [GK00] V. Guruswami and S. Khanna. On the hardness of 4-coloring a 3-colorable graph. In *Proceedings of the 15th Annual IEEE Conference on Computational Complexity*, pages 188–197. IEEE Computer Society Press, May 2000.
- [GK04] V. Guruswami and S. Khanna. On the hardness of 4-coloring a 3-colorable graph. *SIAM Journal on Discrete Mathematics*, 18(1):30–40, 2004.
- [GKP89] R. Graham, D. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Wesley, 1989.
- [GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, February 1989.
- [GMW91] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):691–729, July 1991.
- [GNW90] T. Gundersmann, N. Nasser, and G. Wechsung. A survey on counting classes. In *Proceedings of the 5th Structure in Complexity Theory Conference*, pages 140–153. IEEE Computer Society Press, July 1990.
- [Gol77] L. Goldschlager. The monotone and planar circuit value problems are log space complete for P. *SIGACT News*, 9(2):25–29, 1977.
- [Gol80] M. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, 1980.
- [Gol88] O. Goldreich. Randomness, interactive proofs, and zero-knowledge—A survey. In R. Herken, editor, *The Universal Turing Machine: A Half-Century Survey*, pages 377–405. Oxford University Press, Oxford, 1988.

- [Gol89] S. Goldwasser. Interactive proof systems. In J. Hartmanis, editor, *Computational Complexity Theory*, pages 108–128. AMS Short Course Lecture Notes: Introductory Survey Lectures, Proceedings of Symposia in Applied Mathematics, Volume 38, American Mathematical Society, 1989.
- [Gol97a] O. Goldreich. Notes on Levin’s theory of average-case complexity. Technical Report TR97-058, Electronic Colloquium on Computational Complexity, November 1997.
- [Gol97b] O. Goldreich. A taxonomy of proof systems. In L. Hemaspaandra and A. Selman, editors, *Complexity Theory Retrospective II*, pages 109–134. Springer-Verlag, 1997.
- [Gol99] O. Goldreich. *Modern cryptography, probabilistic proofs, and pseudorandomness*, volume 17 of *Algorithms and Combinatorics*. Springer-Verlag, 1999.
- [Gol01] O. Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2001.
- [GOR98] J. Goldsmith, M. Ogihara, and J. Rothe. Tally NP sets and easy census functions. In *Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science*, pages 483–492. Springer-Verlag *Lecture Notes in Computer Science #1450*, August 1998.
- [GOR00] J. Goldsmith, M. Ogihara, and J. Rothe. Tally NP sets and easy census functions. *Information and Computation*, 158(1):29–52, April 2000.
- [GP86] L. Goldschlager and I. Parberry. On the construction of parallel computers from various bases of boolean functions. *Theoretical Computer Science*, 43(1):43–58, 1986.
- [Gre91] F. Green. An oracle separating $\oplus P$ from PP^{PH} . *Information Processing Letters*, 37(3):149–153, 1991.
- [GRTZ02] N. Gisin, G. Ribordy, W. Tittel, and H. Zbinden. Quantum cryptography. *Reviews of Modern Physics*, 74:145–195, 2002.
- [GRW02] A. Große, J. Rothe, and G. Wechsung. Computing complete graph isomorphisms and hamiltonian cycles from partial ones. *Theory of Computing Systems*, 35(1):81–93, February 2002.
- [GRW06] A. Große, J. Rothe, and G. Wechsung. On computing the smallest four-coloring of planar graphs and non-self-reducible sets in P. *Information Processing Letters*, 99(6):215–221, 2006.
- [GS88] J. Grollmann and A. Selman. Complexity measures for public-key cryptosystems. *SIAM Journal on Computing*, 17(2):309–335, 1988.
- [GS89] S. Goldwasser and M. Sipser. Private coins versus public coins in interactive proof systems. In S. Micali, editor, *Randomness and Computation*, volume 5 of *Advances in Computing Research*, pages 73–90. JAI Press, Greenwich, 1989. A preliminary version appeared in *Proc. 18th Ann. ACM Symp. on Theory of Computing*, 1986, pp. 59–68.
- [Gup91] S. Gupta. The power of witness reduction. In *Proceedings of the 6th Structure in Complexity Theory Conference*, pages 43–59. IEEE Computer Society Press, June/July 1991.
- [Gup93] S. Gupta. On bounded-probability operators and C=P. *Information Processing Letters*, 48:93–98, 1993.
- [GW87] T. Gundermann and G. Wechsung. Counting classes with finite acceptance types. *Computers and Artificial Intelligence*, 6(5):395–409, 1987.
- [Hač79] L. Hačijan. A polynomial algorithm in linear programming. *Soviet Math. Dokl.*, 20:191–194, 1979.
- [Har69] F. Harary. *Graph Theory*. Addison-Wesley, 1969.

- [Har74] F. Harary. A survey of the reconstruction conjecture. In *Graphs and Combinatorics*, pages 18–28. Springer-Verlag *Lecture Notes in Mathematics* #406, 1974.
- [Har78] J. Hartmanis. On log-tape isomorphisms of complete sets. *Theoretical Computer Science*, 7(3):273–286, 1978.
- [Har83a] J. Hartmanis. Generalized Kolmogorov complexity and the structure of feasible computations. In *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science*, pages 439–445. IEEE Computer Society Press, 1983.
- [Har83b] J. Hartmanis. On sparse sets in NP–P. *Information Processing Letters*, 16(2):55–60, 1983.
- [Har89] J. Hartmanis. Gödel, von Neumann, and the P =? NP problem. *Bulletin of the EATCS*, 38:101–107, 1989.
- [Hås88] J. Håstad. Solving simultaneous modular equations of low degree. *SIAM Journal on Computing*, 17(2):336–341, April 1988. Special issue on cryptography.
- [Hau14] F. Hausdorff. *Grundzüge der Mengenlehre*. Walter de Gruyter and Co., 1914.
- [Hem87] L. Hemachandra. The strong exponential hierarchy collapses. In *Proceedings of the 19th ACM Symposium on Theory of Computing*, pages 110–122. ACM Press, May 1987.
- [Hem89] L. Hemachandra. The strong exponential hierarchy collapses. *Journal of Computer and System Sciences*, 39(3):299–322, 1989.
- [Hem93] L. Hemaspaandra. Lowness: A yardstick for NP-P. *SIGACT News*, 24(Spring)(2):10–14, 1993.
- [Hem98] H. Hempel. *Boolean Hierarchies – On Collapse Properties and Query Order*. PhD thesis, Friedrich-Schiller-Universität Jena, Jena, Germany, October 1998.
- [Her90] U. Hertrampf. Relations among MOD-classes. *Theoretical Computer Science*, 74(3):325–328, 1990.
- [HH88] J. Hartmanis and L. Hemachandra. Complexity classes without machines: On complete languages for UP. *Theoretical Computer Science*, 58(1–3):129–142, 1988.
- [HH91] J. Hartmanis and L. Hemachandra. One-way functions and the nonisomorphism of NP-complete sets. *Theoretical Computer Science*, 81(1):155–163, 1991.
- [HH00] E. Hemaspaandra and L. Hemaspaandra. Computational politics: Electoral systems. In *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science*, pages 64–83. Springer-Verlag *Lecture Notes in Computer Science* #1893, 2000.
- [HH04] M. Holzer and W. Holzer. TantrixTM rotation puzzles are intractable. *Discrete Applied Mathematics*, 144(3):345–358, 2004.
- [HH07] E. Hemaspaandra and L. Hemaspaandra. Dichotomy for voting systems. *Journal of Computer and System Sciences*, 73(1):73–83, 2007.
- [HHGP⁺03] J. Hoffstein, N. Howgrave-Graham, J. Pipher, J. Silverman, and W. Whyte. NTRUSIGN: Digital signatures using the NTRU lattice. In *Topics in Cryptology – CT-RSA 2003, The Cryptographers’ Track at the RSA Conference*, pages 122–140. Springer-Verlag *Lecture Notes in Computer Science* #2612, 2003.
- [HHH98a] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel. Query order in the polynomial hierarchy. *Journal of Universal Computer Science*, 4(6):574–588, June 1998.
- [HHH98b] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel. What’s up with downward collapse: Using the easy-hard technique to link boolean and polynomial hierarchy collapses. *SIGACT News*, 29(3):10–22, 1998.

- [HHH99] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel. A downward collapse within the polynomial hierarchy. *SIAM Journal on Computing*, 28(2):383–393, 1999.
- [HHH01] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel. Using the no-search easy-hard technique for downward collapse. Technical Report TR-752, University of Rochester, Department of Computer Science, Rochester, NY, June 2001. Earlier versions or parts of this paper appeared in the Proceedings of the 6th Italian Conference on Theoretical Computer Science (ICTCS’98) and of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS’99).
- [HHR97a] E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Exact analysis of Dodgson elections: Lewis Carroll’s 1876 voting system is complete for parallel access to NP. *Journal of the ACM*, 44(6):806–825, November 1997.
- [HHR97b] E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Raising NP lower bounds to parallel NP lower bounds. *SIGACT News*, 28(2):2–13, June 1997.
- [HHR05] E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Anyone but him: The complexity of precluding an alternative. In *Proceedings of the 20th National Conference on Artificial Intelligence*, pages 95–101. AAAI Press, 2005.
- [HHR07a] E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Anyone but him: The complexity of precluding an alternative. *Artificial Intelligence*, 171(5–6):255–285, 2007.
- [HHR07b] E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Hybrid elections broaden complexity-theoretic resistance to control. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 1308–1314. AAAI Press, January 2007.
- [HHRT04] E. Hemaspaandra, L. Hemaspaandra, S. Radziszowski, and R. Tripathi. Complexity results in graph reconstruction. In *Proceedings of the 29th International Symposium on Mathematical Foundations of Computer Science*, pages 287–297. Springer-Verlag Lecture Notes in Computer Science #3153, 2004.
- [HHT97] Y. Han, L. Hemaspaandra, and T. Thierauf. Threshold computation and cryptographic security. *SIAM Journal on Computing*, 26(1):59–78, February 1997.
- [HHW99] L. Hemaspaandra, H. Hempel, and G. Wechsung. Query order. *SIAM Journal on Computing*, 28(2):637–651, 1999.
- [HIS85] J. Hartmanis, N. Immerman, and V. Sewelson. Sparse sets in NP–P: EXPTIME versus NEXPTIME. *Information and Control*, 65(2/3):159–181, 1985.
- [HJ95] L. Hemaspaandra and S. Jha. Defying upward and downward separation. *Information and Computation*, 121:1–13, 1995.
- [HJRW97] L. Hemaspaandra, Z. Jiang, J. Rothe, and O. Watanabe. Polynomial-time multi-selectivity. *Journal of Universal Computer Science*, 3(3):197–229, March 1997.
- [HJRW98] L. Hemaspaandra, Z. Jiang, J. Rothe, and O. Watanabe. Boolean operations, joins, and the extended low hierarchy. *Theoretical Computer Science*, 205(1–2):317–327, September 1998.
- [HK73] J. Hopcroft and R. Karp. An $n^{5/2}$ algorithm for maximum matching in bipartite graphs. *SIAM Journal on Computing*, 2:225–231, 1973.
- [HL94] S. Homer and L. Longpré. On reductions of NP sets to sparse sets. *Journal of Computer and System Sciences*, 48(2):324–336, April 1994.
- [HLS65] J. Hartmanis, P. Lewis, and R. Stearns. Classification of computations by time and memory requirements. In *Proc. IFIP Congress 65*, pages 31–35, Washington, D.C., 1965. International Federation for Information Processing, Spartan Books.

- [HM80] J. Hartmanis and S. Mahaney. An essay about research on sparse NP complete sets. In *Proceedings of the 9th Symposium on Mathematical Foundations of Computer Science*, pages 40–57. Springer-Verlag *Lecture Notes in Computer Science* #88, September 1980.
- [HM83] S. Homer and W. Maass. Oracle dependent properties of the lattice of NP sets. *Theoretical Computer Science*, 24(3):279–289, 1983.
- [HMU01] J. Hopcroft, R. Motwani, and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, second edition, 2001.
- [HNOS96] E. Hemaspaandra, A. Naik, M. Ogiwara, and A. Selman. P-selective sets and reducing search to decision vs. self-reducibility. *Journal of Computer and System Sciences*, 53(2):194–209, 1996.
- [HO93] L. Hemachandra and M. Ogiwara. Is $\#P$ closed under subtraction? In G. Rosenberg and A. Salomaa, editors, *Current Trends in Theoretical Computer Science: Essays and Tutorials*, pages 523–536. World Scientific Press, 1993.
- [HO02] L. Hemaspaandra and M. Ogiwara. *The Complexity Theory Companion*. EAT-CS Texts in Theoretical Computer Science. Springer-Verlag, Berlin, Heidelberg, New York, 2002.
- [Hof82] C. Hoffman. *Group-Theoretic Algorithms and Graph Isomorphism*. Lecture Notes in Computer Science #136. Springer-Verlag, 1982.
- [Hom04] C. Homan. Tight lower bounds on the ambiguity of strong, total, associative, one-way functions. *Journal of Computer and System Sciences*, 68(3):657–674, 2004.
- [HOW92] L. Hemachandra, M. Ogiwara, and O. Watanabe. How hard are sparse sets? In *Proceedings of the 7th Structure in Complexity Theory Conference*, pages 222–238. IEEE Computer Society Press, June 1992.
- [HPR01] L. Hemaspaandra, K. Pasanen, and J. Rothe. If $P \neq NP$ then some strongly noninvertible functions are invertible. In *Proceedings of the 13th International Symposium on Fundamentals of Computation Theory*, pages 162–171. Springer-Verlag *Lecture Notes in Computer Science* #2138, August 2001.
- [HPS98] J. Hoffstein, J. Pipher, and J. Silverman. NTRU: A ring-based public key cryptosystem. In *Proceedings of the 3rd International Symposium on Algorithmic Number Theory*, pages 267–288, 1998.
- [HPS01] J. Hoffstein, J. Pipher, and J. Silverman. NSS: An NTRU lattice-based signature scheme. In *Advances in Cryptology – EUROCRYPT ’01*, pages 211–228. Springer-Verlag *Lecture Notes in Computer Science* #2045, 2001.
- [HR95] L. Hemaspaandra and J. Rothe. Intersection suffices for boolean hierarchy equivalence. In *Proceedings of the 1st Annual International Computing and Combinatorics Conference*, pages 430–435. Springer-Verlag *Lecture Notes in Computer Science* #959, August 1995.
- [HR97a] E. Hemaspaandra and J. Rothe. Recognizing when greed can approximate maximum independent sets is complete for parallel access to NP. Technical Report Math/Inf/97/14, Friedrich-Schiller-Universität Jena, Jena, Germany, May 1997.
- [HR97b] L. Hemaspaandra and J. Rothe. Unambiguous computation: Boolean hierarchies and sparse Turing-complete sets. *SIAM Journal on Computing*, 26(3):634–653, June 1997.
- [HR98] E. Hemaspaandra and J. Rothe. Recognizing when greed can approximate maximum independent sets is complete for parallel access to NP. *Information Processing Letters*, 65(3):151–156, February 1998.

- [HR99] L. Hemaspaandra and J. Rothe. Creating strong, total, commutative, associative one-way functions from any one-way function in complexity theory. *Journal of Computer and System Sciences*, 58(3):648–659, 1999.
- [HR00] L. Hemaspaandra and J. Rothe. Characterizing the existence of one-way permutations. *Theoretical Computer Science*, 244(1–2):257–261, August 2000.
- [HRS05] L. Hemaspaandra, J. Rothe, and A. Saxena. Enforcing and defying associativity, commutativity, totality, and strong noninvertibility for one-way functions in complexity theory. In *Proceedings of the 9th Italian Conference on Theoretical Computer Science*, pages 265–279. Springer-Verlag *Lecture Notes in Computer Science* #3701, October 2005.
- [HRS06] E. Hemaspaandra, J. Rothe, and H. Spakowski. Recognizing when heuristics can approximate minimum vertex covers is complete for parallel access to NP. *R.A.I.R.O. Theoretical Informatics and Applications*, 40(1):75–91, 2006.
- [HRS08] L. Hemaspaandra, J. Rothe, and A. Saxena. Enforcing and defying associativity, commutativity, totality, and strong noninvertibility for one-way functions in complexity theory. *Theoretical Computer Science*, 401(1–3):27–35, 2008.
- [HRSZ98] L. Hemaspaandra, K. Rajasethupathy, P. Sethupathy, and M. Zimand. Power balance and apportionment algorithms for the United States Congress. *The ACM Journal of Experimental Algorithms*, 3(1):article 1, 16 pp., 1998.
- [HRW97a] L. Hemaspaandra, J. Rothe, and G. Wechsung. Easy sets and hard certificate schemes. *Acta Informatica*, 34(11):859–879, November 1997.
- [HRW97b] L. Hemaspaandra, J. Rothe, and G. Wechsung. On sets with easy certificates and the existence of one-way permutations. In *Proceedings of the 3rd Italian Conference on Algorithms and Complexity*, pages 264–275. Springer-Verlag *Lecture Notes in Computer Science* #1203, March 1997.
- [HS65] J. Hartmanis and R. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.
- [HS66] F. Hennie and R. Stearns. Two-tape simulation of multitape Turing machines. *Journal of the ACM*, 13:533–546, 1966.
- [HS89] S. Homer and A. Selman. Oracles for structural properties: the isomorphism problem and public-key cryptography. In *Proceedings of the 4th Structure in Complexity Theory Conference*, pages 3–14. IEEE Computer Society Press, June 1989.
- [HS97] L. Hemaspaandra and A. Selman, editors. *Complexity Theory Retrospective II*. Springer-Verlag, 1997.
- [HS01] S. Homer and A. Selman. *Computability and Complexity Theory*. Texts in Computer Science. Springer-Verlag, 2001.
- [HSV05] E. Hemaspaandra, H. Spakowski, and J. Vogel. The complexity of Kemeny elections. *Theoretical Computer Science*, 349(3):382–391, December 2005.
- [HT98] P. Hegernes and J. Telle. Partitioning graphs into generalized dominating sets. *Nordic Journal of Computing*, 5(2):128–142, 1998.
- [HT02] C. Homann and M. Thakur. One-way permutations and self-witnessing languages. In R. Baeza-Yates, U. Montanari, and N. Santoro, editors, *Foundations of Information Technology in the Era of Network and Mobile Computing*, pages 243–254. Kluwer Academic Publishers, August 2002. Proceedings of the 17th IFIP World Computer Congress/2nd IFIP International Conference on Theoretical Computer Science.
- [HT03a] L. Hemaspaandra and L. Torenvliet. *Theory of Semi-Feasible Algorithms*. EATCS Monographs in Theoretical Computer Science. Springer-Verlag, 2003.

- [HT03b] C. Homan and M. Thakur. One-way permutations and self-witnessing languages. *Journal of Computer and System Sciences*, 67(3):608–622, 2003.
- [HU79] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
- [HW79] G. Hardy and E. Wright. *An Introduction to the Theory of Numbers*. Clarendon Press, Oxford, 5th edition, 1979.
- [HW97] E. Hemaspaandra and G. Wechsung. The minimization problem for boolean formulas. In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, pages 575–584. IEEE Computer Society Press, October 1997.
- [HW02] E. Hemaspaandra and G. Wechsung. The minimization problem for boolean formulas. *SIAM Journal on Computing*, 31(6):1948–1958, 2002.
- [HY84] J. Hartmanis and Y. Yesha. Computation times of NP sets of different densities. *Theoretical Computer Science*, 34(1–2):17–32, 1984.
- [HZ96] L. Hemaspaandra and M. Zimand. Strong self-reducibility precludes strong immunity. *Mathematical Systems Theory*, 29(5):535–548, 1996.
- [Imm88] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17:935–938, 1988.
- [IN96] R. Impagliazzo and M. Naor. Efficient cryptographic schemes provably as secure as subset sum. *Journal of Cryptology*, 9(4):199–216, 1996.
- [IT03] K. Iwama and S. Tamaki. Improved upper bounds for 3-SAT. Technical Report TR03-053, Electronic Colloquium on Computational Complexity, July 2003. 3 pages.
- [Jac74] N. Jacobson. *Basic Algebra I*. W. H. Freeman, 1974.
- [JL76] N. Jones and W. Laaser. Problems complete for deterministic polynomial time. *Theoretical Computer Science*, 3(1):105–117, October 1976.
- [JLL76] N. Jones, Y. Lien, and W. Laaser. New problems complete for nondeterministic log space. *Mathematical Systems Theory*, 10(1):1–17, 1976.
- [Joh81] D. Johnson. The NP-completeness column: An ongoing guide. *Journal of Algorithms*, 2(4):393–405, December 1981. First column in a series of columns on NP-completeness appearing in the same journal.
- [Jon75] N. Jones. Space-bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences*, 11:68–75, 1975.
- [JY85] D. Joseph and P. Young. Some remarks on witness functions for nonpolynomial and noncomplete sets in NP. *Theoretical Computer Science*, 39(2–3):225–237, August 1985.
- [JY90] D. Joseph and P. Young. Self-reducibility: Effects of internal structure on computational complexity. In A. Selman, editor, *Complexity Theory Retrospective*, pages 82–107. Springer-Verlag, 1990.
- [Kad88] J. Kadin. The polynomial time hierarchy collapses if the boolean hierarchy collapses. *SIAM Journal on Computing*, 17(6):1263–1282, 1988. Erratum appears in the same journal, 20(2):404, 1991.
- [Kad89] J. Kadin. $P^{NP[\log n]}$ and sparse Turing-complete sets for NP. *Journal of Computer and System Sciences*, 39(3):282–298, 1989.
- [Kah67] D. Kahn. *The Codebreakers: The Story of Secret Writing*. MacMillan Publishing Company, 1967.
- [Kar72] R. Karp. Reducibilities among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103, 1972.
- [KH94] D. Kratsch and L. Hemaspaandra. On the complexity of graph reconstruction. *Mathematical Systems Theory*, 27(3):257–273, 1994.

- [KIT88] K. Kurosawa, T. Ito, and M. Takeuchi. Preliminary comments on the MIT public-key cryptosystem. *Cryptologia*, 12(4):225–233, 1988.
- [KL80] R. Karp and R. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the 12th ACM Symposium on Theory of Computing*, pages 302–309. ACM Press, April 1980. An extended version has also appeared as: Turing machines that take advice, *L'Enseignement Mathématique*, 2nd series 28, 1982, pages 191–209.
- [KLD86] K. Ko, T. Long, and D. Du. On one-way functions and polynomial-time isomorphisms. *Theoretical Computer Science*, 47:263–276, 1986.
- [Kle52] S. Kleene. *Introduction to Metamathematics*. D. van Nostrand Company, Inc., New York and Toronto, 1952.
- [KLS00] S. Khanna, N. Linial, and S. Safra. On the hardness of approximating the chromatic number. *Combinatorica*, 20(3):393–415, 2000.
- [KMR87] S. Kurtz, S. Mahaney, and J. Royer. Progress on collapsing degrees. In *Proceedings of the 2nd Structure in Complexity Theory Conference*, pages 126–131. IEEE Computer Society Press, June 1987.
- [KMR88] S. Kurtz, S. Mahaney, and J. Royer. Collapsing degrees. *Journal of Computer and System Sciences*, 37:247–268, 1988.
- [KMR89] S. Kurtz, S. Mahaney, and J. Royer. The isomorphism conjecture fails relative to a random oracle. In *Proceedings of the 21st ACM Symposium on Theory of Computing*, pages 157–166. ACM Press, May 1989.
- [KMR90] S. Kurtz, S. Mahaney, and J. Royer. The structure of complete degrees. In A. Selman, editor, *Complexity Theory Retrospective*, pages 108–146. Springer-Verlag, 1990.
- [Knu98] D. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2 of *Computer Science and Information*. Addison-Wesley, third edition, 1998.
- [Ko82] K. Ko. Some observations on the probabilistic algorithms and NP-hard problems. *Information Processing Letters*, 14(1):39–43, 1982.
- [Ko85] K. Ko. On some natural complete operators. *Theoretical Computer Science*, 37(1):1–30, 1985.
- [Ko90] K. Ko. A note on separating the relativized polynomial time hierarchy by immune sets. *R.A.I.R.O. Theoretical Informatics and Applications*, 24(3):229–240, 1990.
- [Köb89] J. Köbler. *Strukturelle Komplexität von Anzahlproblemen*. PhD thesis, University of Stuttgart, Stuttgart, Germany, 1989. In German.
- [Köb94] J. Köbler. Locating P/poly optimally in the extended low hierarchy. *Theoretical Computer Science*, 134:263–285, 1994.
- [Köb95] J. Köbler. On the structure of low sets. In *Proceedings of the 10th Structure in Complexity Theory Conference*, pages 246–261. IEEE Computer Society Press, 1995.
- [Kob97] N. Koblitz. *Algebraic Aspects of Cryptography*, volume 3 of *Algorithms and Computation in Mathematics*. Springer-Verlag, 1997.
- [KPP04] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer-Verlag, Berlin, Heidelberg, New York, 2004.
- [KR95] B. Kaliski Jr. and M. Robshaw. The secure use of RSA. *CryptoBytes*, 1(3):7–13, 1995.
- [Kra22] M. Kraïtchik. *Théorie des Nombres*. Gauthier-Villars, Paris, 1922.
- [Kre88] M. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36:490–509, 1988.

- [KRS88] B. Kaliski Jr., R. Rivest, and A. Sherman. Is the data encryption standard a group? (Results of cycling experiments on DES). *Journal of Cryptology*, 1(1):3–36, 1988.
- [KS85] K. Ko and U. Schöning. On circuit-size complexity and the low hierarchy in NP. *SIAM Journal on Computing*, 14(1):41–51, 1985.
- [KS94] H. Kaplan and R. Shamir. The domatic number problem on some perfect graph families. *Information Processing Letters*, 49(1):51–56, January 1994.
- [KS01] R. Kumar and D. Sivakumar. Complexity of SVP—A reader’s digest. *SIGACT News*, 32(3):40–52, June 2001.
- [KS04] N. Kayal and N. Saxena. On the ring isomorphism & automorphism problems. Technical Report TR04-109, Electronic Colloquium on Computational Complexity, October 2004.
- [KST89] J. Köbler, U. Schöning, and J. Torán. On counting and approximation. *Acta Informatica*, 26(4):363–379, 1989.
- [KST92] J. Köbler, U. Schöning, and J. Torán. Graph isomorphism is low for PP. *Computational Complexity*, 2:301–330, 1992.
- [KST93] J. Köbler, U. Schöning, and J. Torán. *The Graph Isomorphism Problem: Its Structural Complexity*. Birkhäuser, 1993.
- [KSTT92] J. Köbler, U. Schöning, S. Toda, and J. Torán. Turing machines with few accepting computations and low sets for PP. *Journal of Computer and System Sciences*, 44(2):272–286, 1992.
- [KSW87] J. Köbler, U. Schöning, and K. Wagner. The difference and truth-table hierarchies for NP. *R.A.I.R.O. Informatique théorique et Applications*, 21:419–435, 1987.
- [Kur64] S. Kuroda. Classes of languages and linear-bounded automata. *Information and Control*, 7(2):207–223, June 1964.
- [KV91] S. Khuller and V. Vazirani. Planar graph coloring is not self-reducible, assuming P \neq NP. *Theoretical Computer Science*, 88(1):183–189, 1991.
- [KW98] J. Köbler and O. Watanabe. New collapse consequences of NP having small circuits. *SIAM Journal on Computing*, 28(1):311–324, 1998.
- [Lad75] R. Ladner. On the structure of polynomial time reducibility. *Journal of the ACM*, 22(1):155–171, 1975.
- [Lag83] J. Lagarias. Knapsack public key cryptosystems and diophantine approximation. In *Advances in Cryptology – CRYPTO ’83*, pages 3–23, New York, 1983. Plenum Press.
- [Lau83] C. Lautemann. BPP and the polynomial hierarchy. *Information Processing Letters*, 17(4):215–217, 1983.
- [Len83] H. Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, 1983.
- [Len87] H. Lenstra Jr. Factoring integers with elliptic curves. *Annals of Mathematics*, 126:649–673, 1987.
- [Lev73] L. Levin. Universal sorting problems. *Problemy Peredaci Informacii*, 9:115–116, 1973. In Russian. English translation in *Problems of Information Transmission*, 9:265–266, 1973.
- [Lis86] G. Lischke. Oracle constructions to prove all possible relationships between relativizations of P, NP, NEL, EP, and NEP. *Zeitsch. f. math. Logik und Grundlagen d. Math.*, 32:257–270, 1986.
- [Lis99] G. Lischke. Towards the actual relationship between NP and exponential time. *Mathematical Logic Quarterly*, 45(1):31–49, 1999. A preliminary version has appeared as: Impossibilities and possibilities of weak separation between NP and

- exponential time. In *Proceedings of the 5th Structure in Complexity Theory Conference*, pages 245–253. IEEE Computer Society Press, 1990.
- [LL76] R. Ladner and N. Lynch. Relativization of questions about log space computability. *Mathematical Systems Theory*, 10(1):19–32, 1976.
- [LL93] A. Lenstra and H. Lenstra, Jr. *The Development of the Number Field Sieve*, volume 1554 of *Lecture Notes in Mathematics*. Springer-Verlag, 1993.
- [LLL82] A. Lenstra, H. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:513–534, 1982.
- [LLS75] R. Ladner, N. Lynch, and A. Selman. A comparison of polynomial time reducibilities. *Theoretical Computer Science*, 1(2):103–124, 1975.
- [LO85] J. Lagarias and A. Odlyzko. Solving low-density subset sum problems. *Journal of the ACM*, 32(1):229–246, 1985.
- [LO91] B. LaMacchia and A. Odlyzko. Computation of discrete logarithms in prime fields. *Designs, Codes and Cryptography*, 1(1):47–62, 1991.
- [Lon82a] T. Long. A note on sparse oracles for NP. *Journal of Computer and System Sciences*, 24:224–232, 1982.
- [Lon82b] T. Long. Strong nondeterministic polynomial-time reducibilities. *Theoretical Computer Science*, 21:1–25, 1982.
- [Lon85] T. Long. On restricting the size of oracles compared with restricting access to oracles. *SIAM Journal on Computing*, 14(3):585–597, 1985. Erratum appears in the same journal, 17(3):628, 1988.
- [LOT03] M. Liśkiewicz, M. Ogiwara, and S. Toda. The complexity of counting self-avoiding walks in subgraphs of two-dimensional grids and hypercubes. *Theoretical Computer Science*, 304(1–3):129–156, July 2003.
- [LR94] K.-J. Lange and P. Rossmanith. Unambiguous polynomial hierarchies and exponential size. In *Proceedings of the 9th Structure in Complexity Theory Conference*, pages 106–115. IEEE Computer Society Press, June/July 1994.
- [LS86] T. Long and A. Selman. Relativizing complexity classes with sparse oracles. *Journal of the ACM*, 33(3):618–627, 1986.
- [LSH65] P. Lewis, R. Stearns, and J. Hartmanis. Memory bounds for recognition of context-free and context-sensitive languages. In *Proceedings of the 6th IEEE Symposium on Switching Circuit Theory and Logical Design*, pages 191–202, 1965.
- [Lub96] M. Luby. *Pseudorandomness and Cryptographic Applications*. Princeton Computer Science Notes. Princeton University Press, Princeton, New Jersey, 1996.
- [Mah82] S. Mahaney. Sparse complete sets for NP: Solution of a conjecture of Berman and Hartmanis. *Journal of Computer and System Sciences*, 25(2):130–143, 1982.
- [Mah86] S. Mahaney. Sparse sets and reducibilities. In R. Book, editor, *Studies in Complexity Theory*, pages 63–118. John Wiley and Sons, 1986.
- [Mah89] S. Mahaney. The isomorphism conjecture and sparse sets. In J. Hartmanis, editor, *Computational Complexity Theory*, pages 18–46. American Mathematical Society, 1989. Proceedings of Symposia in Applied Mathematics #38.
- [Mat79] R. Mathon. A note on the graph isomorphism counting problem. *Information Processing Letters*, 8(3):131–132, 1979.
- [May04] A. May. Computing the RSA secret key is deterministic polynomial time equivalent to factoring. In *Advances in Cryptology – CRYPTO ’04*, pages 213–219. Springer-Verlag Lecture Notes in Computer Science #3152, 2004.
- [McC81] W. McColl. Planar crossovers. *IEEE Transactions on Computers*, C-30(3):223–225, 1981.

- [MG02] D. Micciancio and S. Goldwasser. *Complexity of Lattice Problems: A Cryptographic Perspective*, volume 671 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, Boston, Massachusetts, March 2002.
- [MH78] R. Merkle and M. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory*, IT-24(5):525–530, 1978.
- [Mil76] G. Miller. Riemann’s hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13:300–317, 1976.
- [Moo92] J. Moore. Protocol failures in cryptosystems. In G. Simmons, editor, *Contemporary Cryptology: The Science of Information Integrity*, pages 541–558. IEEE Computer Society Press, 1992.
- [MP79] A. Meyer and M. Paterson. With what frequency are apparently intractable problems difficult? Technical Report MIT/LCS/TM-126, MIT Laboratory for Computer Science, Cambridge, MA, 1979.
- [MS72] A. Meyer and L. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Proceedings of the 13th IEEE Symposium on Switching and Automata Theory*, pages 125–129, 1972.
- [MS85] B. Monien and E. Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics*, 10:287–295, 1985.
- [MU95] I. McLean and A. Urken. *Classics of Social Choice*. University of Michigan Press, Ann Arbor, Michigan, 1995.
- [Mül93] H. Müller. A note on balanced immunity. *Mathematical Systems Theory*, 26(2):157–167, 1993.
- [MW99] U. Maurer and S. Wolf. The relationship between breaking the Diffie-Hellman protocol and computing discrete logarithms. *SIAM Journal on Computing*, 28(5):1689–1721, 1999.
- [MW00] U. Maurer and S. Wolf. The Diffie–Hellman protocol. *Designs, Codes and Cryptography*, 19(2/3):147–171, 2000.
- [MY85] S. Mahaney and P. Young. Reductions among polynomial isomorphism types. *Theoretical Computer Science*, 39:207–224, 1985.
- [Nat91] National Institute of Standards and Technology (NIST). Digital signature standard (DSS). *Federal Register*, 56(169), August 1991.
- [Nat92] National Institute of Standards and Technology (NIST). The Digital Signature Standard, proposed by NIST. *Communications of the Association for Computing Machinery*, 35(7):36–40, July 1992.
- [NR98] R. Niedermeier and P. Rossmanith. Unambiguous computations and locally definable acceptance types. *Theoretical Computer Science*, 194:137–161, 1998.
- [NS01] P. Nguyen and J. Stern. The two faces of lattices in cryptology. In *Proceedings of the International Conference on Cryptography and Lattices*, pages 146–180. Springer-Verlag *Lecture Notes in Computer Science* #2146, 2001.
- [Odi89] P. Odifreddi. *Classical Recursion Theory*. North-Holland, 1989.
- [Odl85] A. Odlyzko. *Discrete Logarithms in Finite Fields and Their Cryptographic Significance*. Springer-Verlag *Lecture Notes in Computer Science* #209, Berlin, Heidelberg, New York, 1985.
- [Odl00] A. Odlyzko. Discrete logarithms: The past and the future. *Designs, Codes and Cryptography*, 19(2/3):129–145, 2000.
- [Ogi95] M. Ogiwara. Sparse hard sets for P yield space-efficient algorithms. In *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science*, pages 354–361. IEEE Computer Society Press, 1995.

- [OH93] M. Ogiwara and L. Hemachandra. A complexity theory for feasible closure properties. *Journal of Computer and System Sciences*, 46(3):295–325, 1993.
- [OT01] M. Ogiwara and S. Toda. The complexity of computing the number of self-avoiding walks in two-dimensional grid graphs and in hypercube graphs. In *Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science*, pages 585–597. Springer-Verlag *Lecture Notes in Computer Science* #2136, 2001.
- [OW91] M. Ogiwara and O. Watanabe. On polynomial-time bounded truth-table reducibility of NP sets to sparse sets. *SIAM Journal on Computing*, 20(3):471–483, June 1991.
- [OW02] T. Ottmann and P. Widmayer. *Algorithmen und Datenstrukturen*. Spektrum Akademischer Verlag, Heidelberg, Berlin, fourth edition, 2002. In German.
- [Pap84] C. Papadimitriou. On the complexity of unique solutions. *Journal of the ACM*, 31(2):392–400, 1984.
- [Pap95] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 2nd edition, 1995. Reprinted with corrections.
- [Pol74] J. Pollard. Theorems on factorization and primality testing. *Proc. Cambridge Philos. Soc.*, 76:521–528, 1974.
- [Pom85] C. Pomerance. The quadratic sieve factoring algorithm. In *Advances in Cryptology – EUROCRYPT ’84*, pages 169–182. Springer-Verlag *Lecture Notes in Computer Science* #209, 1985.
- [PPSZ98] R. Paturi, P. Pudlák, M. Saks, and F. Zane. An improved exponential-time algorithm for k -SAT. In *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science*, pages 628–637. IEEE Computer Society Press, November 1998.
- [Pra75] V. Pratt. Every prime has a succinct certificate. *SIAM Journal on Computing*, 4(3):214–220, 1975.
- [PS82] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.
- [PW88] C. Papadimitriou and D. Wolfe. The complexity of facets resolved. *Journal of Computer and System Sciences*, 37(1):2–13, 1988.
- [PY84] C. Papadimitriou and M. Yannakakis. The complexity of facets (and some facets of complexity). *Journal of Computer and System Sciences*, 28(2):244–259, 1984.
- [PZ83] C. Papadimitriou and S. Zachos. Two remarks on the power of counting. In *Proceedings of the 6th GI Conference on Theoretical Computer Science*, pages 269–276. Springer-Verlag *Lecture Notes in Computer Science* #145, 1983.
- [Rab60] M. Rabin. Degree of difficulty of computing a function and a partial ordering of recursive sets. Technical Report Technical Report 2, The Hebrew University, Jerusalem, Israel, 1960.
- [Rab79] M. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical Report LCS/TR-212, Massachusetts Institute of Technology, Laboratory for Computer Science, 1979.
- [Rab80] M. Rabin. Probabilistic algorithms for testing primality. *Journal of Number Theory*, 12:128–138, 1980.
- [Rac82] C. Rackoff. Relativized questions involving probabilistic algorithms. *Journal of the ACM*, 29(1):261–268, 1982.
- [Raz87] A. Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mat. Zametki*, 41(4):598–607, 1987. In Russian. English Translation in *Mathematical Notes of the Academy of Sciences of the USSR*, 41(4):333–338, 1987.

- [Rei90] R. Reischuk. *Einführung in die Komplexitätstheorie*. Teubner, Stuttgart, 1990. In German.
- [RH02] J. Rothe and L. Hemaspaandra. On characterizing the existence of partial one-way permutations. *Information Processing Letters*, 82(3):165–171, May 2002.
- [Rog67] H. Rogers, Jr. *The Theory of Recursive Functions and Effective Computability*. McGraw-Hill, 1967.
- [Rog97] J. Rogers. The Isomorphism Conjecture holds and one-way functions exist relative to an oracle. *Journal of Computer and System Sciences*, 54(3):412–423, June 1997.
- [Roh95] P. Rohatgi. Saving queries with randomness. *Journal of Computer and System Sciences*, 50(3):476–492, 1995.
- [Rol05] D. Rolf. Improved bound for the PPSZ/Schöning-algorithm for 3-SAT. Technical Report TR05-159, Electronic Colloquium on Computational Complexity, 2005.
- [Ros67] A. Rosenberg. Real-time definable languages. *Journal of the ACM*, 14:645–662, 1967.
- [Ros99] K. Rosen. *Elementary Number Theory and its Applications*. Addison-Wesley, 1999.
- [Rot95] J. Rothe. A promise class at least as hard as the polynomial hierarchy. *Journal of Computing and Information*, 1(1):92–107, April 1995. Special Issue: *Proceedings of the 6th International Conference on Computing and Information*, CD-ROM ISSN 1201-8511, Trent University Press.
- [Rot99] J. Rothe. Immunity and simplicity for exact counting and other counting classes. *R.A.I.R.O. Theoretical Informatics and Applications*, 33(2):159–176, March/April 1999.
- [Rot02] J. Rothe. Some facets of complexity theory and cryptography: A five-lecture tutorial. *ACM Computing Surveys*, 34(4):504–549, December 2002.
- [Rot03] J. Rothe. Exact complexity of Exact-Four-Colorability. *Information Processing Letters*, 87(1):7–12, July 2003.
- [Rot04] J. Rothe. Erlkönig. In H. Hempel, editor, *Wechsung in Jena. Ein Sammelband mit Erinnerungen an das Wirken von Gerd Wechsung an der alma mater jenensis*, page 114, Jena, Germany, February 2004. Appeared also in *Informatik Spektrum*, 27(1):80, Springer-Verlag, February 2004.
- [Rot07a] J. Rothe. Complexity theory. In A. Iványi, editor, *Algorithms of Informatics I*, pages 364–392. Mondat Kiadó, 2007. Chapter 8. English translation of *Informatikai Algoritmusok I*.
- [Rot07b] J. Rothe. Cryptology. In A. Iványi, editor, *Algorithms of Informatics I*, pages 332–363. Mondat Kiadó, 2007. Chapter 7. English translation of *Informatikai Algoritmusok I*.
- [RR06a] T. Riege and J. Rothe. Completeness in the boolean hierarchy: Exact-Four-Colorability, minimal graph uncolorability, and exact domatic number problems – a survey. *Journal of Universal Computer Science*, 12(5):551–578, 2006.
- [RR06b] T. Riege and J. Rothe. Complexity of the exact domatic number problem and of the exact conveyor flow shop problem. *Theory of Computing Systems*, 39(5):635–668, September 2006.
- [RR06c] T. Riege and J. Rothe. Improving deterministic and randomized exponential-time algorithms for the satisfiability, the colorability, and the domatic number problem. *Journal of Universal Computer Science*, 12(6):725–745, 2006.
- [RRW94] R. Rao, J. Rothe, and O. Watanabe. Upward separation for FewP and related classes. *Information Processing Letters*, 52(4):175–180, April 1994. Corrigendum appears in the same journal, 74(1–2):89, 2000.

- [RS59] M. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:114–125, 1959.
- [RS93] M. Rabi and A. Sherman. Associative one-way functions: A new paradigm for secret-key agreement and digital signatures. Technical Report CS-TR-3183/UMIACS-TR-93-124, Department of Computer Science, University of Maryland, College Park, Maryland, 1993.
- [RS97] M. Rabi and A. Sherman. An observation on associative one-way functions in complexity theory. *Information Processing Letters*, 64(5):239–244, 1997.
- [RS98] A. Russell and R. Sundaram. Symmetric alternation captures BPP. *Computational Complexity*, 7(2):152–162, 1998.
- [RSA78] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signature and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
- [RSV02] J. Rothe, H. Spakowski, and J. Vogel. Exact complexity of Exact-Four-Colorability and of the winner problem for Young elections. In R. Baeza-Yates, U. Montanari, and N. Santoro, editors, *Foundations of Information Technology in the Era of Network and Mobile Computing*, pages 310–322. Kluwer Academic Publishers, August 2002. Proceedings of the 17th IFIP World Computer Congress/2nd IFIP International Conference on Theoretical Computer Science.
- [RSV03] J. Rothe, H. Spakowski, and J. Vogel. Exact complexity of the winner problem for Young elections. *Theory of Computing Systems*, 36(4):375–386, June 2003.
- [Rue86] R. Rueppel. *Analysis and Design of Stream Ciphers*. Springer-Verlag, Berlin, Heidelberg, New York, 1986.
- [RW01] S. Reith and K. Wagner. On boolean lowness and boolean highness. *Theoretical Computer Science*, 261(2):305–321, June 2001.
- [Saa95] D. Saari. *Basic Geometry of Voting*. Springer-Verlag, Berlin, Heidelberg, New York, 1995.
- [Saa01] D. Saari. *Chaotic Elections! A Mathematician Looks at Voting*. American Mathematical Society, 2001.
- [Sal73] A. Salomaa. *Formal Languages*. Academic Press, 1973.
- [Sal96] A. Salomaa. *Public-Key Cryptography*, volume 23 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, second edition, 1996.
- [Sav70] W. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.
- [Sav73] W. Savitch. Maze recognizing automata and nondeterministic tape complexity. *Journal of Computer and System Sciences*, 7(4):389–403, August 1973.
- [SB84] U. Schöning and R. Book. Immunity, relativization, and nondeterminism. *SIAM Journal on Computing*, 13(2):329–337, 1984.
- [Sch76] C. Schnorr. Optimal algorithms for self-reducible problems. In S. Michaelson and R. Milner, editors, *Proceedings of the 3rd International Colloquium on Automata, Languages, and Programming*, pages 322–337, University of Edinburgh, July 1976. Edinburgh University Press.
- [Sch79] C. Schnorr. On self-transformable combinatorial problems, 1979. Presented at *IEEE Symposium on Information Theory*, Udine, and *Symposium über Mathematische Optimierung*, Oberwolfach.
- [Sch81] U. Schöning. A note on complete sets for the polynomial hierarchy. *SIGACT News*, 13:30–34, 1981.
- [Sch82] U. Schöning. A uniform approach to obtain diagonal sets in complexity classes. *Theoretical Computer Science*, 18:95–103, 1982.

- [Sch83] U. Schöning. A low and a high hierarchy within NP. *Journal of Computer and System Sciences*, 27:14–28, 1983.
- [Sch87] U. Schöning. Probabilistic complexity classes and lowness. In *Proceedings of the 2nd Structure in Complexity Theory Conference*, pages 2–8. IEEE Computer Society Press, June 1987.
- [Sch88] U. Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37(3):312–323, 1988.
- [Sch89] U. Schöning. Probabilistic complexity classes and lowness. *Journal of Computer and System Sciences*, 39(1):84–100, 1989.
- [Sch90] C. Schnorr. Efficient identification and signature schemes for smart cards. In *Advances in Cryptology – CRYPTO ’89*, pages 239–251. Springer-Verlag *Lecture Notes in Computer Science* #435, February 1990.
- [Sch95a] U. Schöning. *Logik für Informatiker*. Spektrum Akademischer Verlag, Heidelberg, Berlin, 1995. In German.
- [Sch95b] U. Schöning. *Perlen der Theoretischen Informatik*. BI Wissenschaftsverlag, Mannheim, 1995. In German.
- [Sch96] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley and Sons, New York, second edition, 1996.
- [Sch99] U. Schöning. A probabilistic algorithm for k -SAT and constraint satisfaction problems. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, pages 410–414. IEEE Computer Society Press, October 1999.
- [Sch01] U. Schöning. *Algorithmik*. Spektrum Akademischer Verlag, Heidelberg, Berlin, 2001. In German.
- [Sch02a] U. Schöning. *Ideen der Informatik*. Oldenbourg Verlag, München, Wien, 2002. In German.
- [Sch02b] U. Schöning. A probabilistic algorithm for k -SAT based on limited local search and restart. *Algorithmica*, 32(4):615–623, 2002.
- [Sch05] U. Schöning. Algorithmics in exponential time. In *Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science*, pages 36–43. Springer-Verlag *Lecture Notes in Computer Science* #3404, 2005.
- [Sel78] A. Selman. Polynomial time enumeration reducibility. *SIAM Journal on Computing*, 7(4):440–457, 1978.
- [Sel79] A. Selman. P-selective sets, tally languages, and the behavior of polynomial time reducibilities on NP. *Mathematical Systems Theory*, 13:55–65, 1979.
- [Sel82a] A. Selman. Analogues of semirecursive sets and effective reducibilities to the study of NP complexity. *Information and Control*, 52:36–51, 1982.
- [Sel82b] A. Selman. Reductions on NP and P-selective sets. *Theoretical Computer Science*, 19:287–304, 1982.
- [Sel88a] A. Selman. Natural self-reducible sets. *SIAM Journal on Computing*, 17(5):989–996, 1988.
- [Sel88b] A. Selman. Promise problems complete for complexity classes. *Information and Computation*, 78:87–98, 1988.
- [Sel90] A. Selman, editor. *Complexity Theory Retrospective*. Springer-Verlag, 1990.
- [Sel92] A. Selman. A survey of one-way functions in complexity theory. *Mathematical Systems Theory*, 25(3):203–221, 1992.
- [Sel94] A. Selman. A taxonomy of complexity classes of functions. *Journal of Computer and System Sciences*, 48(2):357–381, 1994.
- [SH95] C. Schnorr and H. Hörner. Attacking the Chor–Rivest cryptosystem by improved lattice reduction. In *Advances in Cryptology – EUROCRYPT ’89*, pages 1–12. Springer-Verlag *Lecture Notes in Computer Science* #921, May 1995.

- [Sha49] C. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28(4):657–715, 1949.
- [Sha54] D. Shanks. A logarithm algorithm. *Math. Tables and Other Aids to Computation*, 8:60–64, 1954.
- [Sha83] A. Shamir. Embedding cryptographic trapdoors in arbitrary knapsack systems. *Information Processing Letters*, 17(2):77–79, 1983.
- [Sha84] A. Shamir. A polynomial-time algorithm for breaking the basic Merkle-Hellman cryptosystem. *IEEE Transactions on Information Theory*, IT-30(5):699–704, 1984.
- [Sha92] A. Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, 1992.
- [Sha95] A. Shamir. RSA for paranoids. *CryptoBytes*, 1(3):1–4, 1995.
- [SHL65] R. Stearns, J. Hartmanis, and P. Lewis. Hierarchies of memory limited computations. In *Proceedings of the 6th IEEE Symposium on Switching Circuit Theory and Logical Design*, pages 179–190, 1965.
- [Sho67] J. Shoenfield. *Mathematical Logic*. Addison-Wesley, 1967.
- [Sim75] J. Simon. *On Some Central Problems in Computational Complexity*. PhD thesis, Cornell University, Ithaca, NY, January 1975. Available as Cornell Department of Computer Science Technical Report TR75-224.
- [Sim79] G. Simmons. Symmetric and asymmetric encryption. *ACM Computing Surveys*, 11(4):305–330, 1979.
- [Sin99] S. Singh. *The Code Book. The Science of Secrecy from Ancient Egypt to Quantum Cryptography*. Fourth Estate, London, 1999.
- [Sip80] M. Sipser. Halting space-bounded computations. *Theoretical Computer Science*, 10(3):335–338, March 1980.
- [Sip83] M. Sipser. A complexity theoretic approach to randomness. In *Proceedings of the 15th ACM Symposium on Theory of Computing*, pages 330–335. ACM Press, 1983.
- [Sip92] M. Sipser. The history and status of the P versus NP question. In *Proceedings of the 24th ACM Symposium on Theory of Computing*, pages 603–618. ACM Press, 1992.
- [SL94] M. Sheu and T. Long. The extended low hierarchy is an infinite hierarchy. *SIAM Journal on Computing*, 23(3):488–509, 1994.
- [Smo87] R. Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of the 19th ACM Symposium on Theory of Computing*, pages 77–82. ACM Press, May 1987.
- [SN77] G. Simmons and M. Norris. Preliminary comments on the MIT public-key cryptosystem. *Cryptologia*, 1(4):406–414, 1977.
- [Soa77] R. Soare. Computational complexity, speedability, and levelable sets. *Journal of Symbolic Logic*, 42:545–563, 1977.
- [Soa87] R. Soare. *Recursively Enumerable Sets and Degrees: A Study of Computable Functions and Computably Generated Sets*. Perspectives in Mathematical Logic. Springer-Verlag, 1987.
- [SS77] R. Solovay and V. Strassen. A fast Monte Carlo test for primality. *SIAM Journal on Computing*, 6:84–85, 1977. Erratum appears in the same journal, 7(1):118, 1978.
- [Ste90] R. Stearns. Juris Hartmanis: The beginnings of computational complexity. In A. Selman, editor, *Complexity Theory Retrospective*, pages 1–18. Springer-Verlag, 1990.
- [Sti95] D. Stinson. *Cryptography: Theory and Practice*. CRC Press, Boca Raton, 1995.

- [Sti05] D. Stinson. *Cryptography: Theory and Practice*. CRC Press, Boca Raton, third edition, 2005.
- [Sto73] L. Stockmeyer. Planar 3-colorability is NP-complete. *SIGACT News*, 5(3):19–25, 1973.
- [Sto77] L. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1977.
- [SU02a] M. Schaefer and C. Umans. Completeness in the polynomial-time hierarchy: Part I: A compendium. *SIGACT News*, 33(3):32–49, September 2002.
- [SU02b] M. Schaefer and C. Umans. Completeness in the polynomial-time hierarchy: Part II. *SIGACT News*, 33(4):22–36, December 2002.
- [Sud95] M. Sudan. *Efficient Checking of Polynomials and Proofs and the Hardness of Approximation Problems*. Springer-Verlag Lecture Notes in Computer Science #1001, 1995. ACM Distinguished Thesis. Based on the author’s Ph.D. thesis, UC Berkeley, 1992.
- [SV00] H. Spakowski and J. Vogel. Θ_2^P -completeness: A classical approach for new results. In *Proceedings of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 348–360. Springer-Verlag Lecture Notes in Computer Science #1974, December 2000.
- [SZ80] A. Shamir and R. Zippel. On the security of the Merkle-Hellman cryptographic scheme. *IEEE Transactions on Information Theory*, IT-26(3):339–340, 1980.
- [Sze88] R. Szelepcsenyi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26:279–284, 1988.
- [Tar93] J. Tarui. Probabilistic polynomials, AC^0 functions and the polynomial-time hierarchy. *Theoretical Computer Science*, 113:167–183, 1993.
- [Thi00] T. Thierauf. *The Computational Complexity of Equivalence and Isomorphism Problems*. Springer-Verlag Lecture Notes in Computer Science #1852, New York, NY, USA, 2000.
- [TO92] S. Toda and M. Ogiwara. Counting classes are at least as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 21(2):316–328, 1992.
- [Tod91] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.
- [Tor91] J. Torán. Complexity classes defined by counting quantifiers. *Journal of the ACM*, 38(3):753–774, 1991.
- [Tur36] A. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, ser. 2, 42:230–265, 1936. Correction, *ibid*, vol. 43, pp. 544–546, 1937.
- [TvEB89] L. Torenvliet and P. van Emde Boas. Simplicity, immunity, relativizations and nondeterminism. *Information and Computation*, 80(1):1–17, 1989.
- [Uma01] C. Umans. The minimum equivalent DNF problem and shortest implicants. *Journal of Computer and System Sciences*, 63(4):597–611, 2001.
- [Val76] L. Valiant. The relative complexity of checking and evaluating. *Information Processing Letters*, 5(1):20–23, 1976.
- [Val79a] L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.
- [Val79b] L. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.
- [Vau01] S. Vaudenay. Cryptanalysis of the Chor–Rivest cryptosystem. *Journal of Cryptology*, 14(2):87–100, 2001.
- [Vaz03] V. Vazirani. *Approximation Algorithms*. Springer-Verlag, second edition, 2003.

- [Ver92] N. Vereshchagin. On the power of PP. In *Proceedings of the 7th Structure in Complexity Theory Conference*, pages 138–143. IEEE Computer Society Press, June 1992.
- [Vol99] H. Vollmer. *Introduction to Circuit Theory*. EATCS Texts in Theoretical Computer Science. Springer-Verlag, 1999.
- [VV86] L. Valiant and V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986.
- [Wag86] K. Wagner. The complexity of combinatorial problems with succinct input representations. *Acta Informatica*, 23:325–356, 1986.
- [Wag87a] K. Wagner. More complicated questions about maxima and minima, and some closures of NP. *Theoretical Computer Science*, 51:53–80, 1987.
- [Wag87b] K. Wagner. Number-of-query hierarchies. Institut für Mathematik 158, Universität Augsburg, Augsburg, Germany, October 1987.
- [Wag89] K. Wagner. Number-of-query hierarchies. Institut für Informatik 4, Universität Würzburg, Würzburg, Germany, February 1989.
- [Wag90] K. Wagner. Bounded query classes. *SIAM Journal on Computing*, 19(5):833–846, 1990.
- [Wan97] J. Wang. Average-case computational complexity theory. In L. Hemaspaandra and A. Selman, editors, *Complexity Theory Retrospective II*, pages 295–328. Springer-Verlag, 1997.
- [Wat88] O. Watanabe. On hardness of one-way functions. *Information Processing Letters*, 27(3):151–157, 1988.
- [Wat91] O. Watanabe. On the P-isomorphism conjecture. *Theoretical Computer Science*, 83(2):337–343, June 1991.
- [Wec85] G. Wechsung. On the boolean closure of NP. In *Proceedings of the 5th Conference on Fundamentals of Computation Theory*, pages 485–493. Springer-Verlag Lecture Notes in Computer Science #199, 1985. (An unpublished precursor of this paper was coauthored by K. Wagner).
- [Wec00] G. Wechsung. *Vorlesungen zur Komplexitätstheorie*, volume 32 of *Teubner-Texte zur Informatik*. Teubner, Stuttgart, 2000. In German.
- [Weg87] I. Wegener. *The Complexity of Boolean Functions*. Wiley Teubner Series in Computer Science. John Wiley and Sons, New York, 1987.
- [Weg03] I. Wegener. *Komplexitätstheorie. Grenzen der Effizienz von Algorithmen*. Springer-Verlag, Berlin, Heidelberg, New York, 2003. In German.
- [Wel93] D. Welsh. *Complexity: Knots, Colourings and Counting*. Cambridge University Press, 1993.
- [Wel98] D. Welsh. *Codes and Cryptography*. Oxford science publications. Clarendon Press, Oxford, 6th edition, 1998. Reprinted with corrections.
- [Wie90] M. Wiener. Cryptanalysis of short RSA secret exponents. *IEEE Transactions on Information Theory*, IT-36(3):553–558, 1990.
- [Wil80] H. Williams. A modification of the RSA public-key encryption procedure. *IEEE Transactions on Information Theory*, IT-26(6):726–729, 1980.
- [Woe03] G. Woeginger. Exact algorithms for NP-hard problems. In M. Jünger, G. Reinelt, and G. Rinaldi, editors, *Combinatorial Optimization: “Eureka, you shrink!”*, pages 185–207. Springer-Verlag Lecture Notes in Computer Science #2570, 2003.
- [Wra77] C. Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3:23–33, 1977.

- [WW86] K. Wagner and G. Wechsung. *Computational Complexity*. D. Reidel Publishing Company, 1986. Distributors for the U.S.A. and Canada: Kluwer Academic Publishers.
- [Yap83] C. Yap. Some consequences of non-uniform conditions on uniform classes. *Theoretical Computer Science*, 26:287–300, 1983.
- [You77] H. Young. Extending Condorcet’s rule. *Journal of Economic Theory*, 16(2):335–353, 1977.
- [You83] P. Young. Some structural properties of polynomial reducibilities and sets in NP. In *Proceedings of the 15th ACM Symposium on Theory of Computing*, pages 392–401. ACM Press, April 1983.
- [You90] P. Young. Juris Hartmanis: Fundamental contributions to isomorphism problems. In A. Selman, editor, *Complexity Theory Retrospective*, pages 28–58. Springer-Verlag, 1990.
- [You92] P. Young. How reductions to sparse sets collapse the polynomial-time hierarchy: A primer. *SIGACT News*, 1992. Part I (#3, pages 107–117), Part II (#4, pages 83–94), and Corrigendum to Part I (#4, page 94).
- [Zac82] S. Zachos. Robustness of probabilistic complexity classes under definitional perturbations. *Information and Computation*, 54(3):143–154, 1982.
- [Zac88] S. Zachos. Probabilistic quantifiers and games. *Journal of Computer and System Sciences*, 36:433–451, 1988.
- [ZH86] S. Zachos and H. Heller. A decisive characterization of BPP. *Information and Control*, 69:125–135, 1986.
- [Zim04] M. Zimand. *Computational Complexity: A Quantitative Perspective*, volume 196 of *North-Holland Mathematics Studies*. Elsevier, 2004.

Sach- und Autorenverzeichnis

\square	28	\forall^p	215, 216 , 221, 224
$\lfloor \cdot \rfloor$	14 , 74	\bowtie	214 , 273
$\lceil \cdot \rceil$	14 , 75	\equiv	36 , 42 , 57
$ \cdot $	19	\cong	50
$\ \cdot \ $	20	\cong_p	132
*	20	\circ	43 , 68 , 307
+	20	\leq	44
-	20	\leq_{ae} , $<_{ae}$, \geq_{ae} , $>_{ae}$	68
\cup	20	\leq_{io} , $<_{io}$, \geq_{io} , $>_{io}$	68
\cap	20	\preceq , \prec , \succeq , \succ	69
\vdash	21	\preceq_{io} , \prec_{io} , \succeq_{io} , \succ_{io}	69
\vdash^*	21	\leq_m^{\log}	91 , 93, 94, 96, 99, 100, 139, 141, 219, 260, 261, 274
\neg	34 , 35	\leq_m^{\log} -Härte	91
\vee	34 , 35, 198 , 271	\leq_m^{\log} -Vollständigkeit	91 , 94, 96, 139, 141, 260, 261
\wedge	34 , 35, 198 , 271	\leq_m^p	89 , 131, 210, 219, 220, 266–269, 274, 276, 288, 308, 312, 330, 333, 339, 348
\Rightarrow	34 , 35	\leq_m^p -Härte	89
\iff	34 , 35	\leq_m^p -Vollständigkeit	89 , 124, 127, 240, 266–269, 276, 288, 308, 312, 330, 333
\oplus	167 , 291	\leq_T^{NP}	219 , 274, 284
Δ	269 , 289	\leq_T^{NP} -Härte	246
\vee	siehe \exists		
\wedge	siehe \forall		
\exists	37		
\exists^p	215, 216 , 221, 224		
\exists^+	315 , 316–333, 339		
\forall	37		

$\leq_{\text{ST}}^{\text{NP}}$ **262**, 266, 276, 290
 $\leq_{\text{ST}}^{\text{NP}}$ -Härte **262**
 $\leq_{\text{ST}}^{\text{NP}}$ -Vollständigkeit **262**
 $\leq_{\text{pos-T}}^{\text{P}}$ 219, **274**, 284
 $\leq_{\text{T}}^{\text{P}}$ 143, **218**, 220, 267, 274, 276,
 284, 287, 329, 345
 $\leq_{\text{T}}^{\text{P}}$ -Härte 143, **219**, 274, 287
 $\leq_{\text{T}}^{\text{P}}$ -Vollständigkeit **219**, 263, 288
 $\leq_{\text{tt}}^{\text{P}}$ **228**, 230, 284, 288
 $\leq_{\text{tt}}^{\text{P}}$ -Härte **229**
 $\leq_{\text{tt}}^{\text{P}}$ -Vollständigkeit **229**
 $(\dot{-})$ **47**, 370
#P 142, 294, **326**, 339–341, 345

- Abschlusseigenschaften von, *siehe* Funktionenklasse,
- Abschlusseigenschaften einer

#P-Vollständigkeit **143**
#P₁ **142**, 143
#P₁-Vollständigkeit 143
#RA 400
⊕P 143, **327**, 329, 330, 334, 337,
 339, 343, 345–347
⊕P-low 329, 345
⊕P^{SPP} 329
2-Colorability **108**, 131
2^{Lin} **68**
2^{Lin(·)} **68**
2^{Pol} **68**
2^{Pol(·)} **68**
2-SAT 64, **96**, 100, 139
2-TRP 135
3-Colorability 6, 64, **108**, 131,
 196, 210, 282
3-DM **112**, 113, 132, 216
3-SAT 64, **96**, 104, 138, 210, 282,
 294–297, 343
3-TRP 135
4-SAT 295, 343
4-TRP 134
5-SAT 295, 343
6-SAT 295, 343

A

$\alpha(\cdot)$ **228**

ablehnende Berechnung
siehe Turingmaschine, Berechnung einer, ablehnende

- Anzahl von —en *siehe* rej_M

Absorption **36**
acc_(·) **304**
adjungierte Matrix **157**
Adjunkte *siehe* adjungierte Matrix
Adleman, L. 290, 349, 350, 399,
 400, 459

- *siehe auch* RSA-Kryptosystem
- *siehe auch* RSA-Signatur

Advanced Encryption Standard
siehe AES
AES 193
affine Chiffre
siehe Chiffre, Block-, affine
affin-lineare Chiffre *siehe* Chiffre,
 Block-, affin-lineare
AGAP **261**
Agrawal, M. 121, 133, 284, 349,
 374, 375, 385, 399
Ajtai, M. 460, 461
akzeptierende Berechnung
siehe Turingmaschine, Berechnung einer, akzeptierende

- Anzahl von —en *siehe* acc_(·)

AL **260**
Alberti, B. 191
al Choresmi, M. 11
Algebra **43–47**, 59
Algorithmik **11–19**, 59
algorithmisches Gerät 62
Algorithmus 11

- Las-Vegas-, *siehe auch* ZPP 8,
 293, 302, **307**, 399, 427, 428
- Laufzeit eines 70, 71
- Monte-Carlo- 8, 293, 302, **305**,
 362
- *no-biased*, *siehe auch* RP **305**,
 307, 360, 362, 368, 373, 400
- *yes-biased*, *siehe auch* coRP
305, 307, 360
- randomisierter 8, 9, 52, 125,
 271, **293–348**, 427, 428

- *siehe* Baby-Step Giant-Step,
 siehe auch SHANKS
- *siehe* BACKTRACKING-SAT
- *siehe* ERWEITERTER-EUKLID
- *siehe* EUKLID
- *siehe* FERMAT
- *siehe* LERC
- *siehe* LLL
- *siehe* MILLER-RABIN
- *siehe* POLLARD
- *siehe* RANDOM-FACTOR
- *siehe* RANDOM-SAT
- *siehe* SHANKS
- *siehe* SOLOVAY-STRASSEN
- *siehe* SQUARE-AND-MULTIPLY
- *siehe* TRIAL-DIVISION
- *siehe auch* Turingmaschine
- Alice 1
- Al-Khowarizmi, A. *siehe* al Chores-mi, M.
- Allen, W. 162, 185
- Allender, E. 137, 141, 142, 291
- allgemein rekursive Funktion *siehe* Funktion, allgemein rekursive
- ALOGTIME **253**
- Alphabet, *siehe auch* Σ **19**
 - Menge der Wörter über einem,
 siehe auch Σ^* **19**
- Alternation 249–261
- Alternative *siehe* Kandidat
- alternierende logarithmische Zeit,
 siehe auch ALOGTIME 61, **253**
- alternierende Polynomialzeit,
 siehe auch AP 7, **257**
- alternierender logarithmischer Raum,
 siehe auch AL 7, **260**
- alternierende Turingmaschine *siehe* ATM; *siehe* Turingmaschine, alternierende
- AM, *siehe auch* Arthur-Merlin-Spiele **322**, 324, 325, 328, 332, 339, 347
- $AM^{AM \cap coAM}$ 325
- AMA, *siehe auch* Arthur-Merlin-Spiele **322**, 324, 339
- Ambos-Spies, K. VIII, 10
- AMH, *siehe auch* Arthur-Merlin-Spiele **322**, 436
- Angriff
 - aktiver 148, 408
 - Chosen-Ciphertext- **146**, 355, 394, 401, 430, 451
 - Chosen-Message- *siehe* Angriff, Chosen-Plaintext
 - Chosen-Plaintext- **146**, 162, 355, 394, 420, 421, 430
 - Ciphertext-only- **146**, 151, 152, 177, 190
 - Key-only- **146**, 420
 - Known-Message-
 siehe Angriff, Known-Plaintext
 - Known-Plaintext- **146**, 163, 164, 176, 184, 188, 420, 421
 - Man-in-the-Middle- 148, 408, 431
 - passiver 148
 - Personifikations- 148
 - Substitutions- 148
 - *siehe auch* kryptoanalytischer Angriff
- Anton, H. 59
- AP **257**
- Approximierbarkeit 9, 139, 282, 286
- Äquivalenz, *siehe auch* \iff **34**, 35
- Äquivalenzrelation **132**
- Arithmetik modulo einer Zahl
 siehe $\mathbb{Z}_{(\cdot)}$, Arithmetik in
- Arora, S. 282
- Arrow, K. 234
- Arthur *siehe* Arthur-Merlin-Spiele
- Arthur-Merlin-Spiele 8, 282, 294, **314–325**, 332, 339, 344, 347, 403, **430–439**, 457
- Arthur-Merlin-Hierarchie *siehe* AMH
 - *siehe auch* Arthur-Merlin-Spiele
- Arthurs Labyrinth 432
- Arvind, V. 140, 288, 335, 346
- ASPACE(\cdot) **251**, 257–260, 276

- Assing, S. VIII
 Assoziativität 36, 43, 445
 – schwache 445, 460
 asymmetrische Kryptographie *siehe*
 Kryptographie, asymmetrische;
 siehe auch Kryptographie,
 Public-Key-
 asymmetrisches Kryptosystem *siehe*
 Kryptosystem, asymmetrisches;
 siehe auch Kryptosystem,
 Public-Key-
 ATIME(\cdot) 251, 253–257, 276
 ATM 61, 62, 67, 195, 249, 251, 252
 Aurich, V. VIII
 Ausiello, G. 9, 282
 Aut(\cdot) 50
 Authentikation 2, 9, 147, 403, 408,
 430–439, 459
 Authentikationscode 148
 Authentikationsproblem 148
 – *siehe* Nachrichtenauthentikation
 – *siehe* Nachrichtenintegrität
 – *siehe* Nutzerauthentikation
 Authentikationsprotokoll 430–439
 – *siehe* Challenge-and-Response-
 Protokoll
 – *siehe* Zero-Knowledge-Protokoll
 auto 335, 338, 341
 Automorphismus eines Graphen,
 siehe auch Aut(\cdot) 50
 Axt *siehe* Werkzeuge, Axt
- B**
 Babai, L. 8, 344, 457
 Babbage, C. 160, 172, 191
 Baby-Klonen 4, 113
 Baby-Step Giant-Step *siehe* SHANKS
 BACKTRACKING-SAT 297, 338
 Baker, T. 292
 balancierte Immunität 291, 292
 Balcázar, J. 10, 136, 140, 291, 344,
 345, 458
 Bartholdi III, J. 285, 286
 Baselt, D. VII
 Bauer, F. 2, 190, 191
- Bauland, M. 284
 Baumeister, D. VII, VIII, 136
 Bayes, T. 53
 – *siehe auch* Satz, von Bayes
 BC(\mathcal{C}) 199, 200, 272
 BC(NP) 199, 200, 244, 272
 Beatles 185
 Beaufort, F. 185
 Beaufort-Chiffre
 siehe Chiffre, Beaufort-
 Beaufort-Quadrat 187
 Beckwith, J. VIII
 Beigel, R. 246, 281, 286, 289, 342,
 344, 345
 berechenbare Funktion
 – *siehe* Funktion, allgemein rekur-
 sive;
 – *siehe* Funktion, partiell rekursive
 Berechenbarkeit 30
 Berechenbarkeitstheorie *siehe*
 Theorie, Berechenbarkeits-
 Berechnung *siehe* Turingmaschine,
 Berechnung einer
 – effiziente 6, 70, 137
 – ineffiziente 6, 70
 – *intractable*
 siehe Berechnung, ineffiziente
 – machbare
 siehe Berechnung, effiziente
 – parallele 252, 257
 – Schwellwert- 303, 343
- Berechnungsmodell
 siehe algorithmisches Gerät;
 siehe Turingmaschine
 Berechnungsparadigma
 siehe Turingmaschine,
 Akzeptierungsmodus einer
 Berman, L. 7, 122–125, 140, 142,
 143, 287, 461
 Berman–Hartmanis-Isomorphie-
 vermutung *siehe* Isomorphe-
 vermutung
 Bernstein, D. 399, 400
 Bernstein, F. 122

- *siehe auch* Satz, von Cantor und Bernstein
- Berthiaume, A. 10
- Bertoni, A. 281
- Beutelspacher, A. 190, 344, 458
- Beweiser 344
 - *siehe auch* Beweissystem, interaktives
 - *siehe auch* Zero-Knowledge-Protokoll
- Beweissystem
 - interaktives 3, 8, 282, 284, 288
 - *siehe auch* Arthur-Merlin-Spiele
 - *siehe auch* PCP
- Beweisverifikation 282
 - *siehe auch* PCP
- Beygelzimer, A. VII, 191, 460
- \mathfrak{B} -glatt **381**, 383
- $\text{BH}(\mathcal{C})$ **199**, 272
- $\text{BH}_k(\mathcal{C})$ **199**, 272, 289
- $\text{BH}(\text{NP})$ **198**, 200, 202, 244, 272, 287
- $\text{BH}_2(\text{NP})$ **197**, 198, 201, 202, 204, 209, 214, 273, 277, 278
- $\text{BH}_k(\text{NP})$ **198**, 200, 201, 204, 214, 240, 246, 272, 276, 281, 289
- $\text{BH}_k(\text{NP})$ -hart 204, 281
- $\text{BH}_k(\text{NP})$ -vollständig 204, 214, 240, 272, 276
- BHT **245**
- Bi-Immunität 291
- $\text{bin}(\cdot)$ **20**
- $\text{Bin}(\cdot)$ **81**
- Binärsuche 217, 228, 243
- Blass, A. 280
- Bleichenbacher, D. 401
- Bletchley Park 2, 191
- Blockchiffre *siehe* Chiffre, Block-
- Blömer, J. 401
- Blum, M. 66, 129, 137, 427
- Blumsche Axiome **66**
- Blumsches Komplexitätsmaß **66**, 129, 137
- Blum-Zahl *siehe* Zahl, Blum-
- Bob 1
- Böhler, E. 284
- Boneh, D. 401
- Book, R. 76, 81, 137, 143, 245, 291
- Boolean Hierarchy Tower, *siehe auch* BHT 201, 245
- boolesche Algebra **199**
- boolesche Formel **33**, **35**, **39**, 96
 - disjunktive Normalform einer, *siehe auch* DNF **55**
 - erfüllbare **36**
 - konjunktive Normalform einer, *siehe auch* KNF **35**
 - in der Aussagenlogik **33**
 - in der Prädikatenlogik **39**
 - *siehe auch* Struktur
 - *siehe auch* Term
 - isomorphe —n **284**
 - quantifizierte *siehe* quantifizierte boolesche Formel
 - semantisch äquivalente —n, *siehe auch* \equiv **36**, **42**
 - Wahrheitsbelegung einer **36**
- boolesche Hierarchie
 - erweiterte 281
 - Normalformen der
 - *siehe* Hausdorff-Hierarchie
 - *siehe* Hierarchie der alternierenden Summen
 - *siehe* symmetrische Differenzen-hierarchie
 - *siehe* Vereinigung-von-Differenzen-Hierarchie; *siehe auch* Hausdorff-Hierarchie
 - *siehe* verschachtelte Differenzen-hierarchie
 - über \mathcal{C} , *siehe auch* $\text{BH}(\mathcal{C})$ **199**, 272
 - k -te Stufe der *siehe* $\text{BH}_k(\mathcal{C})$
 - über NP, *siehe auch* $\text{BH}(\text{NP})$ 7, 195, **198**, 200, 202, 244, 245, 272, 281, 286, 289
 - Kollaps der 201, 245, 281
 - k -te Stufe der *siehe* $\text{BH}_k(\text{NP})$
 - zweite Stufe der *siehe* $\text{BH}_2(\text{NP})$; *siehe auch* DP

- über RP 281
 - über UP 281
 - boolesche Hülle, *siehe auch* Mengenklasse, boolesche Hülle einer
 - von \mathcal{C} , *siehe auch* BC(\mathcal{C}) **199**, 200, 272
 - von NP, *siehe auch* BC(NP) **199**, 200, 244, 272
 - boolesche Konstante **35**
 - boolesche Operation
 - *siehe* Äquivalenz, *siehe auch* \iff
 - *siehe* Disjunktion, *siehe auch* \vee
 - *siehe* Exklusives-Oder, *siehe auch* \oplus
 - *siehe* Implikation, *siehe auch* \implies
 - *siehe* Konjunktion, *siehe auch* \wedge
 - *siehe* Negation, *siehe auch* \neg
 - boolescher Ausdruck
 - siehe* boolesche Formel
 - boolesche Variable **35**
 - freie **37**
 - gebundene **37**
 - quantifizierte *siehe* boolesche Variable, gebundene
 - Boppana, R. 344, 457
 - Borchert, B. VII, 284
 - Borges, J. 5
 - Borodin, A. 142
 - *siehe auch* Satz, von Borodin und Demers
 - Bovet, D. 10, 136, 458
 - BPP 8, 143, 293, **310**, 312–315, 318, 320–324, 326, 339, 342–345, 347
 - BPP_{path} 339, **342**, 343
 - BPP^{BPP} 323
 - Brandstädt, A. 59, 136, 138
 - Brassard, G. 458, 461
 - break-elgamal **417**
 - break-rabin **428**, 429, 430
 - Brent, R. 385
 - Brickell, E. 459
 - Brillhart, J. 385
 - Brueggemann, T. 295, 297, 343
 - Bruschi, D. 345
 - Bruß, D. VIII, 10, 191
 - Bshouty, N. 284
 - Buchmann, J. 5, 190, 344, 399, 400, 412, 457, 458
 - Buhrman, H. 137, 140, 284, 289
 - Bungter, S. VII
 - Bush, G. W. 233
 - Buss, S. 244, 286, 287
- C**
- $c_{(\cdot)}$ **30**
 - $\mathbb{C}=\mathbb{P}$ 143, **327**, 329, 330, 334, 337, 339, 343, 345–347
 - $\mathbb{C}=\mathbb{P}\text{-low}$ 329, 348
 - $\mathbb{C}=\mathbb{P}^{\text{SPP}}$ 329
 - Cai, J. 141, 143, 191, 279, 281, 287, 288, 290, 292, 400, 460
 - Canetti, R. 288
 - Cantor, G. 122
 - *siehe auch* Satz, von Cantor und Bernstein
 - Cantor-Bernstein-Theorem *siehe* Satz, von Cantor und Bernstein
 - Carmichael-Zahl
 - siehe* Zahl, Carmichael-
 - Carroll, L. *siehe* Dodgson, C.
 - Carter, J. 330, 346
 - Cäsar, J. 2, 151, 191
 - *siehe auch* Chiffre, Cäsar-Cäsar-Chiffre *siehe* Chiffre, Cäsar-
 - Castro, J. 287
 - CBC-Modus **167**, 168–170, 188
 - census(\cdot) **124**
 - CF **23**, 58
 - CFB-Modus **168**, 169, 188
 - Chakaravarthy, V. 288
 - Challenge-and-Response-Protokoll 433, 436
 - Chandra, A. 290
 - Chang, R. 246, 281, 289
 - charakteristische Funktion
 - siehe* Funktion, charakteristische
 - Chen, Z. 284
 - Chiffre
 - Beaufort- **187**

- Block- 7, **149**, 155–170
 - affine 7, **151**
 - affin-lineare 7, 155, 156, **158**, 162, 188
 - lineare **158**
- Cäsar- **151**
- Hill- 7, **158**, 164, 170, 185, 188
- Permutations- 7, 149, **150**, 159, 166, 168, 183, 188, 191
- Strom- 7, 165, **170**, 188, 189, 191
- Substitutions- 7, **149**
- Transpositions- *siehe* Chiffre, Permutations-
- Verschiebungs- **150**, 151, 153, 155, 184, 189
- Vigenère- 7, **155**, 156, 158–161, 163, 166, 170, 172, 184, 186, 188, 191
 - *siehe auch* kryptoanalytischer Angriff
 - *siehe auch* Kryptosystem
- Chinesischer Restesatz *siehe* Satz, Chinesischer Reste-
choice set **234**
- Chomsky, N. 23
- Chomsky-Hierarchie **23**
- Chor, B. 459
 - *siehe auch* Chor–Rivest–Kryptosystem
- Chor–Rivest–Kryptosystem 459
- Chosen-Ciphertext-Angriff *siehe* Angriff, Chosen-Ciphertext-
- Chosen-Message-Angriff *siehe* Angriff, Chosen-Plaintext-
- Chosen-Plaintext-Angriff *siehe* Angriff, Chosen-Plaintext-
- Cicero, Q. 191
- cipherblock chaining mode* *siehe* CBC-Modus
- cipher feedback mode* *siehe* CFB-Modus
- ciphertext* *siehe* Schlüsseltext
- Ciphertext-only-Angriff *siehe* Angriff, Ciphertext-only-
- ciphertext space*
 - siehe* Schlüsselraum
- cleartext space* *siehe* Klartextraum
- Clique **106**, 131
- Clique-Facet 273, **280**
- Cliquenpolytop,
 - siehe auch* Polytope(.) 280
- Facette eines —s 280
- Clique-Problem, *siehe auch* Clique **106**, 131, 216, 279
- coAM 324, 325, 328, 332, 347, 433
- Cobham, A. 137
- coBPP 312
- co \mathcal{C} *siehe* Mengenklasse, co-Operator, angewandt auf eine
- coC=P 328, 347
- Cocks, C. 399, 400
- Codebrecher 2
- coDP **198**, 201, 202, 281
- co-Graph **106**
- Cohen, S. VIII
- coMA 324, 339, 347
- coMAM 324
- complete search reducing to partial search* 140
- Computational Social Choice 232, 286
- Computeretzwerk
 - siehe* Netzwerk, Computer-
- Condorcet, Marquis de, J.-A.-N. de Caritat 234
- Condorcet-Gewinner **234**, 285
- Condorcet-Paradox **234**
- Condorcet-Prinzip **235**, 285, 286
- Condorcet-SCF **235**
- coNL **88**, 139
- coNLINSPACE **88**
- coNP **124**, 133, 142, 197, **198**, 201, 202, 221, 222, 246, 263, 271–273, 275, 280, 283, 284, 287–289, 314, 328, 339, 343, 345, 347
- coNP-hart 141, 209, 246, 273, 275, 280, 283

coNP-vollständig 132, 246, 272, 275,
 283, 284, 288
 coNP^{NP}, *siehe auch* Π_2^P 221, 222
 Conrad, S. VIII
 coNSPACE(\cdot) **88**
 Conveyor-Flow-Shop-Problem 282
 – exaktes 282
 Cook, S. 7, 64, 100–102, 127, 128,
 131, 138, 284
 – *siehe auch* Cook-Reduktion
 – *siehe auch* Cooks Kriterium
 – *siehe auch* Satz, von Cook
 Cook-Levin-Theorem
 siehe Satz, von Cook
 Cook-Reduktion *siehe* Satz, von Cook
 Cooks Kriterium 7, 257
 co-Operator *siehe* Mengenklasse, co-
 Operator, angewandt auf eine
 Coppersmith, D. 401, 459
 Cormen, T. 59
 coRP 305, 307, 347, 363, 365, 396
 coRP(\cdot) 396
counting class *siehe* Zählklasse
counting hierarchy
 siehe Zähl-Hierarchie
 coUP 128, 132, 142, 341, 347, 376,
 400, 446, 461
 Crépeau, C. VIII, 1, 458
 Crescenzi, P. 9, 10, 136, 282, 458
 Critical-Clique **279**, 280
 CS **23**, 88
 cs($\cdot|\cdot$) **133**

D

$\delta(\cdot)$ **110**, 196
 Δ_2^P , *siehe auch* P^{NP} **220**, 221, 222,
 284, 287, 345, 347
 Δ_2^P -hart 284
 Δ_2^P -vollständig 284
 Δ_3^P , *siehe auch* P<sup>NP^{NP} **220**, 221,
 222, 289
 Δ_i^P **220**, 221, 222, 224, 274, 291
 $D(\cdot)$ **32**, 66
 DAAD VIII
 Daemen, J. 193</sup>

Dantsin, E. 295, 297, 343
 Data Encryption Standard, *siehe auch*
 DES 192
 DEA **24**, 26, 55
 Deck *siehe* Graph, Deck eines —en
 – legitimes **347**
 – Urbild eines —s **347**
 Deck-Checking **347**
 Delbrouck, T. 192
 Demers, A. 142
 – *siehe auch* Satz, von Borodin und
 Demers
 deMorgan, A. 36
 deMorgansche Regeln **36**
 DES 192
 det 143, **157**, 189
 Determinante einer Matrix *siehe*
 Matrix, Determinante einer
 deterministische Polynomialzeit
 siehe P
 Deutsche Wehrmacht 2, 189
 DFG VIII
 Díaz, J. 10, 136, 344, 458
 Dietzfelbinger, M. 375, 396, 399, 400
 Diffie, W. 9, 399, 400, 404, 456
 – *siehe auch* Diffie–Hellman-
 Problem
 – *siehe auch* Diffie–Hellman-
 Protokoll
 diffie-hellman **409**, 417, 450
 Diffie–Hellman 408, 450
 Diffie–Hellman-Problem **409**, 417,
 450, 461
 – als Entscheidungsproblem
 siehe Diffie–Hellman
 – als funktionales Problem
 siehe diffie-hellman
 Diffie–Hellman-Protokoll 9, 350, 399–
 403, **404–412**, 414, 416, 418,
 431, 442, 450, 456, 461
 – Sicherheit des —s *siehe* Kryp-
 toanalyse, von Diffie–Hellman
 digitale Signatur 8, **147**
 – Fälschung einer *siehe* Fälschung
 – *siehe* Protokoll, digitale Signatur

- digitales Signatur-Schema *siehe*
Protokoll, digitale Signatur
- Digital Signature Standard *siehe*
United States Digital Signature
Standard
- diktatorisch 233
- Disjunktion, *siehe auch* \vee 34, 35
- diskreter Logarithmus 46, 405
 - Bit-Problem des, *siehe auch* dlogbit 418, 420
 - Bit-Sicherheit des 418, 454
 - Problem des 9, 403–405, 407, 409, 410, 412, 414, 416, 421, 440, 450, 457, 461
 - als Entscheidungsproblem *siehe* DLog
 - als funktionales Problem *siehe* dlog
- Distanzfunktion 279
- Distributivität 36
- Dixon, J. 384, 400
- dlog 409, 450
- DLog 450
- dlogbit 418, 420
- DNA-Test 5, 113, 114
- DNF 55
- DNF-SAT 55
- DNP 110, 196
- Dodgson, C. 285
 - *siehe auch* Dodgson-Wahlsystem
- Dodgson-Gewinner 285
- Dodgson-Score 285
- Dodgson-Wahlsystem 285
 - Gewinnerproblem für das 285
 - homogene Variante des —s 285
 - Rankingproblem für das 285
- Dogma 5, 70, 71
- Domatische-Zahl-Problem, *siehe auch* DNP 110, 138, 196
 - allgemeine Version des exakten —s, *siehe auch* Exact- M_k -DNP 198, 204
 - Approximation des —s 139
 - exaktes, *siehe auch* Exact- i -DNP 196, 204, 209, 271, 272
- Doppelnegation 36
- DOTM 33
- Downward Collapse
siehe Upward Separation
- Downward Separation
siehe Upward Collapse
- DP 197, 198, 201, 202, 204, 209, 214, 271, 273, 277–281, 283, 284
- DPOTM 33, 218
- DP-vollständig 197, 204, 214, 271, 273, 277–281, 283, 284
- Drache 5, 432
- Dreifarbarkeitsproblem
siehe 3-Colorability
- DSPACE(\cdot) 66, 69, 77, 79, 85, 129, 130, 253–257, 276
- DTIME(\cdot) 66, 69, 72, 74, 80, 85, 129, 257–260, 276
- DTM 28, 65
- Du, D. 10, 136, 141, 458
- Durfee, G. 401
- Dwork, C. 286, 460, 461
- E**
- ε 19
- e 306, 339
- E 70, 80, 82, 130, 143
- $E(\cdot)$ 54
- $E(\cdot)$ 93
- EASY $^\forall_\forall$ 142
- Easy-Hard-Technik 246, 247
- ECB-Modus 165, 166, 168, 188
- Echtzeit, *siehe auch* REALTIME 61, 76, 133
- Edmonds, J. 137
- effektive Nummerierung von \mathbb{P}
siehe Funktion, partiell rekursive, Gödelsierung von —en
- Ehrlichkeit *siehe* Funktion, ehrliche Eindeutigkeitsprobleme 279, 280
- Einwegfunktion 3, 7, 9, 122, 125, 126, 141
 - assoziative 445
 - Falltür- 9, 439, 440

- injektive 141
- kommutative **446**
- *one-to-one* *siehe* Einwegfunktion, injektive
- *polynomial-to-one* 142
- starke **444**, **445**, 446, 453
- surjektive 128, 132
- *trapdoor*
 siehe Einwegfunktion, Falltür
- Worst-Case- 404, 407, 443, 460, 461
- Einweg/Isomorphie-Vermutung **125**
- Einwegpermutation **142**
- Einweg-Vermutung **125**
- Eiter, T. 283, 287
- electronic codebook mode*
 siehe ECB-Modus
- ElGamal, T. 9, 403, 412–424, 428, 440, 450, 451, 457
 - *siehe auch* ElGamal-Kryptosystem
 - *siehe auch* ElGamal-Signatur
- ElGamal-Kryptosystem 403, **412–414**, 427, 440, 450, 457
 - Sicherheit des —s *siehe* Kryptanalyse, von ElGamal
- ElGamal-Signatur 9, **414–416**, 451, 457
 - Sicherheit der *siehe* Kryptanalyse, von ElGamal
- Ella *siehe* Rothe, E.
- Elliptische-Kurven-Methode 384, 387, 400
- Ellis, J. 399, 400
- ELow₂ **291**
- ELow_k **291**
- endlicher Automat
 - deterministischer, *siehe auch* DEA **24**, 26, 55, 61
 - Endzustand eines —en **24**, **25**
 - erweiterte Überführungsfunktion eines —en **24**, **25**
 - Gödelisierung von —en 55
 - nichtdeterministischer, *siehe auch* NEA **25**, 26, 55, 59, 61
 - Startzustand eines —en **24**, **25**
- stochastischer **297**, 300
- absorbierender Zustand eines —en **297**
- Überführungsgraph eines —en *siehe* Markoff-Kette
- Überführungsfunktion eines **24**, **25**
- von einem —en akzeptierte Sprache **24**, **25**
- Zustand eines —en **24**, **25**
- Enigma 2, 189
- Entropie 7, 172, 177, **178**, 189, 192, 301
 - bedingte **181**, 182, 189
 - Eigenschaften der **179**, 182, 189
 - Gruppierungseigenschaft der **180**
 - Subadditivität der **180**
- Entscheidbarkeit **30**
- Eratosthenes 356
 - Sieb des
 siehe Sieb des Eratosthenes
- Erdélyi, G. VII, VIII, 10
- Ereignis **53**
- Erfüllbarkeitsproblem 6, 64, **96**, 101, 114, 124, 216, 225, 257, 265, 293–301, 312
 - *siehe* 2-SAT
 - *siehe* 3-SAT
 - *siehe* 4-SAT
 - *siehe* 5-SAT
 - *siehe* 6-SAT
 - *siehe* DNF-SAT
 - *siehe* k -SAT
 - *siehe* Majority-SAT
 - *siehe* Minimal-3-UNSAT
 - *siehe* Odd- k -SAT
 - *siehe* Odd-Max-SAT
 - *siehe* Odd-SAT
 - *siehe* SAT
 - *siehe* SAT-UNSAT
 - *siehe* Threshold-SAT
 - *siehe* Unique-SAT
- Erich 1
- Erlkönig 195
- Erwartungswert, *siehe auch* $E(\cdot)$ **54**

- erweiterte Lowness
 – *siehe* ELow₂
 – *siehe* ELow_k
 – *siehe* Low-Hierarchy, erweiterte
- ERWEITERTER-EUKLID **14**
- Espelage, W. 282
- EUKLID **12**, 18, 19, 390, 391
- Euklid 12
 – Algorithmus von *siehe* EUKLID
 – erweiterter Algorithmus von
 siehe ERWEITERTER-EUKLID
- Euklidischer Algorithmus
 siehe EUKLID
- Euler, L. 45–47, 56, 152, 350, 369,
 370, 385, 387, 418, 419, 426
 – *siehe auch* Euler-Funktion
 – *siehe auch* Euler-Kriterium
 – *siehe auch* Satz, von Euler
- Euler-Funktion, *siehe auch* $\varphi(\cdot)$ **45**,
 56, 152, 350, 387
- Euler-Kriterium **46**, 47, 369, 370,
 418, 419, 426
- Eulers Theorem
 siehe Satz, von Euler
- eval(\cdot) **250**
- Even, S. 346, 460
- Exact-2-DNP 209, 272
- Exact-3-Colorability 209, 283
- Exact-3-DNP 209
- Exact-4-Colorability 214, 283
- Exact-4-DNP 209
- Exact-5-DNP 204
- Exact-7-Colorability 273
- exact cover by 3-sets problem*
 siehe exaktes Überdeckungsproblem durch Dreiermengen
- Exact-*i*-Clique 273
- Exact-*i*-Colorability **209**
- Exact-*i*-DNP **196**
- Exact-*i*-Favorite 273
- Exact-*i*-IS 273
- Exact- M_k -Clique 273
- Exact- M_k -Colorability **209**, 214,
 273
- Exact- M_k -DNP **198**, 204
- Exact- M_k -Favorite 273
- Exact- M_k -IS 273
- exaktes Überdeckungsproblem durch Dreiermengen **111**, **117**
 – *siehe auch* X-3-Cover
- existenzielle Fälschung
 siehe Fälschung, existenzielle existenzieller Quantor *siehe* Quantor, existenzieller
 – polynomell längenbeschränkter
 siehe \exists^p
- Exklusives-Oder, *siehe auch* \oplus **167**
- EXP **70**, 80
- Exponentialzeit **70**
 – deterministische
 siehe E; *siehe* EXP
 – nichtdeterministische
 siehe NE; *siehe* NEXP
- exponentieller Raum
 – deterministischer
 siehe EXPSPACE
 – nichtdeterministischer
 siehe NEXPSPACE
- $\exp_{r,p}(\cdot)$ **46**, 405
- EXPSPACE **70**, 79, 130
- F**
- $\varphi(\cdot)$ **45**
- $F(\cdot)$ **385**, 398
- $f(\cdot)$ **15**
- Facette *siehe* Cliquenpolytop,
 Facette eines —s
- Facettenproblem **7**, **279**
 – *siehe* Clique-Facet
 – *siehe* TSP-Facet
- factoring **375**, 397
- Factoring **397**, 400
- Faktorbasis 381
- Faktorisierungsalgorithmus 9, 349,
 375–385, 387, 398, 400
 – *siehe* Elliptische-Kurven-Methode
 – *siehe* Pollards $(p - 1)$ -Methode
 – *siehe* Probdivision
 – *siehe* quadratisches Sieb

- *siehe* Spezial-Faktorisierungs-methoden
- *siehe* Universal-Faktorisierungs-methoden
- Faktorisierungsangriff 386
 - mit *brute force* **386**
 - mit elliptischen Kurven 387
 - mit Spezial-Faktorisierungs-methoden **387**
 - mit Universal-Faktorisierungs-methoden **387**
- Faktorisierungsmethode *siehe* Faktorisierungsalgorithmus
- Faktorisierungsproblem 6, 8, 120, **375**, 386, 399
 - als Entscheidungsproblem *siehe* Factoring
 - als funktionales Problem *siehe* factoring
- Faliszewski, P. VIII, 233
- Falltür-Einwegfunktion
 - siehe* Einwegfunktion, Falltür-Information
- Falltür-Information **439**, 440, 442
- Fälschung
 - existenzielle **420**
 - selektive **420**
 - *siehe auch* totaler Bruch
- Färbbarkeitsproblem 64, **108**
 - *siehe* 2-Colorability
 - *siehe* 3-Colorability
 - *siehe* k -Colorability
 - allgemeine Version des exakten —s, *siehe auch* Exact- M_k -Colorability **209**, 214
 - exaktes, *siehe auch* Exact- i -Colorability **209**, 214, 283
- Favorite 232, 273
- Favorite-Equ 232
- Favorite-Geq 232
- Favorite-Odd 232
- Feige, U. 139, 459
- Feigenbaum, J. 191, 458
- Feller, W. 59
- Fellows, M. 376, 400
- Fenner, S. 141, 142, 345, 461
- FERMAT **360**
- Fermat, P. de 45, 385, 400
 - *siehe auch* FERMAT
 - *siehe auch* Primzahltest, Fermat
 - *siehe auch* Satz, Fermats Kleiner
- Fermat-Lügner **358**, 359
- Fermats Kleiner Satz
 - siehe* Satz, Fermats Kleiner
- Fermat-Test
 - siehe* Primzahltest, Fermat
- Fermat-Zahl *siehe* Zahl, Fermat
- Fermat-Zeuge **358**, 359
- FewE 137
- FewP 137, **141**, 142, 288, 291, 305
- FI **284**
- Fiat, A. 438, 439, 452, 459
 - *siehe auch* Fiat-Shamir-Protokoll
- Fiat-Shamir-Protokoll **439**, 452, 459
 - *siehe auch* Zero-Knowledge-Protokoll
- Fibonacci-Zahl
 - siehe* Zahl, Fibonacci
- first-order logic *siehe* Logik, Prädikaten-, erster Stufe
- Fishburn, P. 235, 285
- FL **91**
- F-Liars(.) **361**, 362, 366, 373
- Formelisomorphie-Problem, *siehe auch* FI **284**
- Forstinger, C. VIII
- Fortnow, L. 141–143, 289, 292, 343–345, 461
- Fortune, S. 123, 141, 461
- FP **89**, 143
- FP-Invertierbarkeit **125**
- FP^{NP} 218, 284
- Frage-Hierarchie über NP, *siehe auch* P^{NP[$\mathcal{O}(1)$]}; P^{NP[log]} 8, **240**, 286
 - k -te Stufe der, *siehe auch* P^{NP[k]} **240**
 - parallele, *siehe auch* P^{NP}_{btt}; P^{NP}_{tt} **240**, 286, 289

- *k*-te Stufe der, *siehe auch* $P_{k\text{-tt}}^{\text{NP}}$ **240**
- Friedberg, R. 139
 - *siehe auch* Theorem, Friedberg–Muchnik-
- Friedberg–Muchnik-Satz *siehe* Theorem, Friedberg–Muchnik-Funktion
 - allgemein rekursive, *siehe auch* \mathbb{R} **30**, 66
 - assoziative **445**
 - berechenbare *siehe* Funktion, allgemein rekursive; *siehe* Funktion, partiell rekursive
 - charakteristische, *siehe auch* $c_{(\cdot)}$ **30**
 - Definitionsbereich einer, *siehe auch* $D_{(\cdot)}$ **32**, 66
 - ehrliche **125**, 444, 453
 - Einweg- *siehe* Einwegfunktion
 - Euler- *siehe* Euler-Funktion
 - Exponentiel-
 - modulare — mit Basis r und Modul p , *siehe auch* $\exp_{r,p}(\cdot)$ **46**, 405
 - mit linearem Exponenten, *siehe auch* 2^{Lin} **68**
 - mit polynomialem Exponenten, *siehe auch* 2^{Pol} **68**
 - FP-invertierbare
 - siehe* FP-Invertierbarkeit
 - in logarithmischem Raum berechenbare, *siehe auch* FL **91**
 - kommutative **446**
 - Komposition von —en, *siehe auch* \circ **68**
 - lineare, *siehe auch* $\mathbb{L}\text{in}$ **68**
 - logarithmische, *siehe auch* \log **19**
 - diskret — mit Basis r und Modul p *siehe* diskreter Logarithmus; *siehe auch* $\log, \alpha \bmod p$ **46**, 405
 - partiell rekursive, *siehe auch* \mathbb{P} **30**
- Gödelisierung von —en **66**
- polynomiale, *siehe auch* $\mathbb{P}\text{ol}$ **68**
- polynomialzeit-berechenbare, *siehe auch* FP **89**
- produktive **125**
- Raum-, *siehe auch* $\text{Space}_{(\cdot)}(\cdot)$ **65**
- raumkonstruierbare **77**, 130
- s-ehrliche **444**, 453
- Social-Choice-
 - siehe* Social-Choice-Funktion;
 - siehe auch* SCF
- totale **30**
- Wachstumsrate einer **68**, 70, 71
- Wertebereich einer, *siehe auch* $R_{(\cdot)}$ **32**
- Zeit-, *siehe auch* $\text{Time}_{(\cdot)}(\cdot)$ **65**
- zeitkonstruierbare **77**, 130
- Zensus-, *siehe auch* $\text{census}_{(\cdot)}$ **124**, 143, 144
- *siehe auch* Hash-Funktion
- Funktionenklasse,
 - Abschlusseigenschaften einer
 - unter Addition **327**, **340**
 - unter Binomialkoeffizienten **327**, **340**, **341**
 - unter eingeschränkter Komposition **327**, **340**, **341**
 - unter Multiplikation **327**, **340**
 - unter Potenzierung **327**, **340**, **341**
 - unter Subtraktion **327**, **340**
 - *siehe auch* Komplexitätsklasse
- Funktionssymbol **40**
- Furst, M. 345

G

- γ -Reduzierbarkeit **290**
- GA **50**
- Gabarró, J. 10, 136, 344, 458
- Gács, P. 344
- Gál, A. 140, 278
- Gandalf 433, 434
- Ganesan, K. 141
- ganze Zahl *siehe* Zahl, ganze

ganzzahlige lineare Programmierung
siehe lineare Programmierung,
 ganzzahlige
 GAP 64, **94**, 139
 $\text{gap}_{(\cdot)}$ **326**
 $\text{GAP}_{\text{acyclic}}$ **99**, 100, 131
 GapP 294, **326**, 329, 339, 341, 343,
 345
 – Abschlusseigenschaften von,
siehe Funktionenklasse,
 Abschlusseigenschaften einer
 GapP-low 328, 329
 Garey, M. 10, 71, 100, 112, 136,
 138, 282
 Gasarch, W. 120
 Gauß, C. 143, 349, 382
 Gaußsches Eliminierungsverfahren
 143, 382
 GCHQ *siehe* Government Communications Head Quarters
 $\text{ggT}(\cdot, \cdot)$ **13**
GI **50**
 Gibbs, J. 180, 192
 – *siehe auch* Entropie, Eigenschaften der
 – *siehe auch* Lemma von Gibbs
 Gill, J. 292, 343
 Gisin, N. 10, 191
 gitter-basierte Kryptographie *siehe*
 Kryptographie, gitter-basierte
 Gitter-Problem 393, 459, 461
 – Average-Case-Härte von —en 461
 – Worst-Case-Härte von —en 461
 Gitter-Reduktion 393, 459, 461
 Glaßer, C. VII
 Gleichverteilung **52**
GNI **332**
 Gödel, K. 32, 138
 Gödelisierung
 – *siehe* endlicher Automat, Gödelisierung von —en
 – *siehe* Funktion, partiell rekursive, Gödelisierung von —en
 – *siehe* Turingmaschine, Gödelisierung von —en

Gödel-Zahl *siehe* Zahl, Gödel
 Goethe, J. W. von 195
 goldener Schnitt 17, **54**
 Goldreich, O. 6, 10, 190, 191, 282,
 344, 346, 399, 433, 437, 438,
 457, 458
 – *siehe auch* Goldreich–Micali–Wigderson-Protokoll
 Goldreich–Micali–Wigderson-Protokoll
433–438, 452, 458
 Goldschlager, L. 136, 345
 Goldsmith, J. VIII, 140, 141, 143
 Goldwasser, S. 10, 191, 344, 346,
 393, 457, 458, 460
 Golumbic, M. 59, 136, 138
 Gottlob, G. 283, 287
 Government Communications Head
 Quarters 399
 Grammatik **21**
 – Ableitungsrelation bezüglich einer, *siehe auch* \vdash **21**
 – reflexive, transitive Hülle von \vdash , *siehe auch* \vdash^* **21**
 – kontextfreie **23**
 – kontextsensitive **23**
 – Nichtterminale einer **21**
 – Produktionen einer
siehe Grammatik, Regeln einer
 – Regeln einer **21**
 – reguläre **23**
 – Sprache einer **21**
 – Startsymbol einer **21**
 – Terminale einer **21**
 – Typ-0- **23**
 – Typ einer **23**
 – Variablen einer *siehe* Grammatik, Nichtterminale einer
 Graph **49**, 93
 – alternierender **261**
 – Erreichbarkeit in einem —en **261**
 – bipartiter **111**, 139
 – Deck eines —en **346**
 – gerichteter **93**
 – azyklischer **99**
 – Hamiltonkreis in einem —en **277**

- Pfad in einem —en **94**
 - Zyklus in einem —en **99**
 - isomorphe —en, *siehe auch* \cong **50**
 - Knotenmenge eines —en,
siehe auch $E(\cdot)$ **93**
 - Knotenmenge eines —en,
siehe auch $V(\cdot)$ **93**
 - kritischer **279**
 - Minimalgrad eines —en, *siehe auch* $\text{min-deg}(\cdot)$ **110**, 206
 - Netzwerk- **196**
 - planarer **279**
 - ungerichteter **106**
 - Automorphismus eines —en
siehe Automorphismus eines Graphen
 - chromatische Zahl eines —en
siehe Zahl, chromatische
 - Clique eines **106**
 - domatische Zahl eines —en
siehe Zahl, domatische
 - dominierende Menge eines —en **110**, 196
 - Färbung eines —en **108**
 - Hamiltonkreis in einem —en **277**
 - Isomorphismus zwischen —en
siehe Isomorphismus, zwischen Graphen
 - *k*-färbbarer **108**
 - Knotenüberdeckung eines —en **106**
 - unabhängige Menge eines —en **106**
 - Unabhängigkeitszahl eines —en
siehe Zahl, Unabhängigkeits-
- Grapherreichbarkeitsproblem, *siehe auch* GAP **64**, **94**, 139
- alternierendes, *siehe auch* AGAP **261**
 - eingeschränkt auf azyklische Graphen, *siehe auch* GAP_{acyclic} **99**, 100, 131
- Graphautomorphie-Problem, *siehe auch* GA **50**, 346
- Graphisomorphie-Problem, *siehe auch* GI **6**, **8**, **49**, 64, 120, 122, 132, 139, 140, 196, 271, 284, 290, 330–337, 341, 345–348, 376, 433–438, 452, 458
- kleinste Lösung einer Instanz des —s **218**
- Graph-Nichtisomorphie-Problem, *siehe auch* GNI **332**
- Graphrekonstruktionsvermutung **346**, 348
- Green, F. **345**
- Greibach, S. **76**
- Grollmann, J. **142**, 460, 461
- Große, A. **VIII**, 140, 278, 284
- größter gemeinsamer Teiler, *siehe auch* ggT(\cdot, \cdot) **13**
- Gruppe **43**
 - abelsche **44**
 - Abschlusseigenschaften einer **43**
 - inverses Element in einer **43**
 - kommutative *siehe* Gruppe, abelsche
 - neutrales Element einer **43**
 - Operation einer, *siehe auch* \circ **43**
 - Ordnung einer endlichen — **44**
 - Ordnung eines Gruppenelements **43**
 - Permutations- *siehe* Permutationsgruppe
 - Untergruppe einer, *siehe auch* \leq **44**
- Gruppenaxiome **43**
- Gundermann, T. **281**
- Gupta, S. **345**
- Gurevich, Y. **280**
- Gurski, F. **VII**
- Guruswami, V. **210**, 213, 282
- *siehe auch* Guruswami–Khanna–Reduktion
- Guruswami–Khanna–Reduktion **210**, 214, 282

H

Halbordnung

- polynomiell längenbeschränkte
121

- polynomiell wohlfundierte **121**

Halevi, S. 140, 278

Halldórsson, M. 9, 139, 282

Halteproblem 265

Hamiltonkreis-Problem **277**

- für gerichtete Graphen **277**

- für ungerichtete Graphen **277**

Han, Y. 343

Harary, J. 59

Hardy, G. 59

Härte

- *siehe* \leq_m^{\log} -Härte

- *siehe* \leq_m^p -Härte

- *siehe* \leq_T^p -Härte

- *siehe* \leq_{tt}^p -Härte

- *siehe* $BH_k(NP)$ -hart

- *siehe* coNP-hart

- *siehe* Δ_2^p -hart

- *siehe* NP-hart

- *siehe* Θ_2^p -hart

Hartmanis, J. 7, 122–125, 136–138,
140–143, 287, 288Hash-Funktion **331**

- Familie von —en **331**

- kollisionsfreie **331**

Hashing

- universelles **331**, 346

Hassan, N. **VIII**

Håstad, J. 344, 393, 401, 457

Häufigkeitsanalyse *siehe* Kryptanalyse, durch Häufigkeitsanalyse

Hausdorff, F. 199, 281

- *siehe auch* Hausdorff-Hierarchie

Hausdorff-Hierarchie **199**, 281

- *siehe auch* boolesche Hierarchie, Normalformen der

Hay, L. 244, 286, 287

Hegernes, P. 282

Heiliger Gral 5, 432

Heiratsproblem *siehe* Matching-
Problem, bipartites

Heller, H. 318, 344, 457

Hellman, M. 9, 399, 400, 404, 411,
439, 456, 459

- *siehe auch* Diffie–Hellman-
Problem

- *siehe auch* Diffie–Hellman-
Protokoll

- *siehe auch* Merkle–Hellman-
Kryptosystem

Hemachandra, L.

siehe Hemaspaandra, L.Hemaspaandra, E. **VII**, 137, 140,
233, 246, 283–286, 289, 290,
348Hemaspaandra, L. **VII**, 10, 72, 125,
136, 137, 140–142, 144, 199,
233, 244, 246, 281, 284–292,
341, 343, 345, 348, 453, 454,
460–462Hempel, H. **VIII**, 246, 289, 290

Hennie, F. 80, 136

HH **262**, 263, 264, 269, 270, 276

Hierarchie

- Arthur-Merlin- *siehe* AMH

- *siehe auch* Arthur-Merlin-Spiele

- boolesche *siehe* boolesche Hier-
archie

- Normalformen der *siehe*
boolesche Hierarchie, Normal-
formen der

- Chomsky- *siehe* Chomsky-
Hierarchie

- der alternierenden Summen
siehe Hierarchie der alternieren-
den Summen

- Frage- *siehe* Frage-Hierarchie
über NP

- Hausdorff- *siehe* Hausdorff-
Hierarchie

- High- *siehe* High-Hierarchie

- Low- *siehe* Low-Hierarchie

- parallele Frage- *siehe* Frage-
Hierarchie über NP, parallele

- Polynomialzeit-
siehe Polynomialzeit-Hierarchie

- Raum- *siehe* Raumhierarchie
 - symmetrische Differenzen-
siehe symmetrische Differenzen-
hierarchie
 - Vereinigung-von-Differenzen-
siehe Hausdorff-Hierarchie
 - verschachtelte Differenzen-
siehe verschachtelte Differenzen-
hierarchie
 - Zähl- *siehe* Zähl-Hierarchie
 - Zeit- *siehe* Zeithierarchie
- Hierarchie der alternierenden Summen
281
- *siehe auch* boolesche Hierarchie,
Normalformen der
- High_0 263, 330
 High_1 263
 High_2 330
- High-Hierarchie, *siehe auch* HH 8,
262, 263, 264, 269, 270, 276,
290
 - erste Stufe der *siehe* High_1
 - k -te Stufe der *siehe* High_k
 - nullte Stufe der *siehe* High_0
 - zweite Stufe der *siehe* High_2
- High_k **262**, 263, 264, 268, 270, 276
- Highness* 262, 267, 291
 - *siehe auch* High-Hierarchie
- Hill, L. 158
 - *siehe auch* Chiffre, Hill-
- Hill-Chiffre *siehe* Chiffre, Hill-
- Hinrichs, M. VIII
- Hoffman, C. 346
- Hoffstein, J. 459
- Holzer, M. 135
- Holzer, W. 135
- Homan, C. VIII, 142, 460, 461
- Homer, S. 10, 59, 136, 141, 288
- Hopcroft, J. 59, 112, 461
- Hörner, H. 459
- Huang, M. 400
- I**
- IBM 192
- id 48**
- id 48**
- Idempotenz **36**
- Identifikationsschema
siehe Fiat-Shamir-Protokoll
- Imberman, N. 88, 137, 139
- Immunität **291**
 - *siehe auch* balancierte Immunität
 - *siehe auch* Bi-Immunität
- Impagliazzo, R. 460
- impersonation attack* *siehe* Angriff,
Personifikations-
- Implikation, *siehe auch* \implies **34**, 35
- Informations- und Codierungstheorie
siehe Theorie, Informations- und
Codierungs-
- integer linear programming* *siehe*
lineare Programmierung, ganz-
zahlige
- interaktives Beweissystem
siehe Beweissystem, interaktives
- Intruder-in-the-Middle-Angriff *siehe*
Angriff, Man-in-the-Middle-
- IP **275**, **344**, 345
 - *siehe auch* Beweissystem, inter-
aktives
- Irrfahrt 300
- IS **106**, 107, 131
- $\text{Iso}(\cdot, \cdot)$ **50**
- Isomorphievermutung 7, 122, **123**,
125, 140, 143, 287, 461
- Isomorphismus
 - zwischen booleschen Formeln **284**
 - zwischen Graphen, *siehe auch*
 $\text{Iso}(\cdot, \cdot)$ **50**
 - zwischen Mengen **122**
- Ito, T. 457
- Iwama, K. 295, 343
- J**
- Jacobi-Symbol, *siehe auch* (\cdot) **47**,
370
- Jacobson, N. 59

Jensens Ungleichung **179**, 180
 Jerschow, Y. VII, 5
 Jha, S. 137
 Jiang, Z. VIII, 291
 Johnson, D. 10, 71, 100, 112, 136,
 138, 282
 Jones, N. 139
 Joseph, D. 124, 125, 140, 141
jump operator
 siehe Sprung-Operator

K

$\kappa(\cdot)$ **117, 235**
 κ **265**, 266, 276
 $K(\cdot)$ **265**, 267, 276
 $K^{(\cdot)}$ **265**, 267, 268, 276
 $K^{(\cdot)}(\cdot)$ **265**, 267, 268, 276
 Kadin, J. 246, 281, 287–289
 Kahn, D. 190
 Kaliski, B. 193, 401
 Kandidat 234
 Kann, V. 9, 282
 Kante *siehe* Graph, Kante eines —en
 Kaplan, H. 138
 Karp, R. 112, 138, 287
 – *siehe auch* Satz, von Karp und Lipton
 Karpinski, M. 9, 282
 Karp–Lipton-Theorem
 siehe Satz, von Karp und Lipton
 Kasiski, F. 160–163, 166, 172, 185,
 191
 – *siehe auch* Kasiskis Methode
 Kasiskis Methode **160**, 161–163, 166,
 185,
 – *siehe auch* Kryptoanalyse, der Vi-
 genère-Chiffre
 Kayal, N. 121, 133, 349, 374, 375,
 385, 399, 400
k-Colorability **108**
 Kellerer, H. 459
 Kelly, P. 346
 Kemenisierung
 – lokale 286
 Kemeny-Wahlsystem 285, 286

Kerckhoffs von Nieuwenhof, J. 147
 – *siehe auch* Kerckhoffssches Prinzip
 Kerckhoffssches Prinzip **147**
 Kern, W. 295, 297, 343
 Kettenbruch **390**
 Kettenbruchentwicklung **390**
 – Konvergent einer **390**
 Kettsäge *siehe* Werkzeuge, Kettsäge
 Key-only-Angriff
 siehe Angriff, Key-only-
key space *siehe* Schlüsselraum
 Khanna, S. 210, 213, 282
 – *siehe auch* Guruswami–Khanna-
 Reduktion
 Khuller, S. 284
 Kiometzis, C. VIII
k-KNF **36**
 Klartext **145**
 Klartextraum **145**
 Kleene, S. 20, 59, 445
KNF **35**
 – *siehe auch* *k*-KNF
 Knoten *siehe* Graph, Knotenmenge
 eines
 – Grad eines **206**
 Knotenüberdeckungsproblem, *siehe*
 auch VC **106**, 131
 Known-Plaintext-Angriff *siehe* An-
 griff, Known-Plaintext-
 Ko, K. 10, 136, 141, 290, 345, 458,
 461
 Köbler, J. 139, 143, 244, 281, 286–
 288, 291, 341, 344–346, 348,
 458
 Koblitz, N. 376, 399, 400
 Kommunikationsnetzwerk *siehe*
 Netzwerk, Kommunikations-
 Kommutativität **36, 446**
 komplexer Durchschnitt
 siehe Mengenklasse, komplexer
 Durchschnitt von —n

komplexe symmetrische Differenz *siehe* Mengenklasse, komplexe symmetrische Differenz von —n
komplexe Vereinigung
siehe Mengenklasse, komplexe Vereinigung von —n
Komplexitätsklasse 5
– \leq_m^{\log} -Hülle einer 91, 93, 131
– \leq_m^p -Hülle einer 89, 131, 220, 291, 339
– \leq_T^{NP} -Hülle einer, *siehe auch* NP \mathcal{C} 219
– \leq_T^p -Hülle einer, *siehe auch* P \mathcal{C} 219, 220, 274, 329
– \leq_{tt}^p -Hülle einer, *siehe auch* P $_{tt}^{\mathcal{C}}$ 229
– alternierende Zeit- und Raum- 251
– Average-Case- 5
– deterministische Zeit- und Raum- 66
– Name einer *siehe* Komplexitätsklasse, Ressourcenfunktion einer
– nichtdeterministische Zeit- und Raum- 67
– probabilistische 8, 291
– Ressourcenfunktion einer 66, 68
– Worst-Case- 5, 64–72
Komplexitätsmaß 5, 62
– alternierendes Zeit- und Raum- 251
– Average-Case- 5
– deterministisches Zeit- und Raum- 65
– nichtdeterministisches Zeit- und Raum- 67
– Worst-Case 5, 64–72
Komplexitätstheorie
siehe Theorie, Komplexitäts-
Kongruenz modulo einer Zahl,
siehe auch \equiv 57
Königstein, G. VIII
Konjunktion, *siehe auch* \wedge 34, 35
konstruierbar in der Zeit 77

K-Operator *siehe* K; *siehe* K(.)
– iterierter *siehe* K (\cdot) ; *siehe* K $(\cdot)(\cdot)$
Körper 44
Kortsarz, G. 139
Kozen, D. 290
Kraïtchik, M. 400
Kratsch, D. VIII, 348
Krentel, M. 284, 286
kritisches Problem 7, 195, 279
– *siehe* Critical-Clique
– *siehe* MDNHC
– *siehe* Minimal-3-Uncolorability
– *siehe* Minimal-3-UNSAT
– *siehe* MNHC
Kryptoanalyse 2, 7, 8, 145–193, 386–394, 403–462
– der affinen Chiffre 7, 152, 154, 184
– der affin-linearen Blockchiffre 162–165, 185–187
– der Hill-Chiffre 7, 164, 185
– der Permutationschiffre 7, 183
– der Substitutionschiffre 7, 152, 154, 184
– der Verschiebungschiffre 184
– der Vigenère-Chiffre 7, 159–163, 184, 186
– durch Häufigkeitsanalyse 154, 184
– von Diffie–Hellman 403, 408–412
– von ElGamal 416–424, 450, 451
– *siehe auch* break-elgamal
– von Merkle–Hellman 454, 459
– von Rabin 427–430, 451
– *siehe auch* break-rabin
– von RSA 8, 375–377, 385, 386–394, 398–401
– *siehe* Low-Exponent Attack
– *siehe* Small-Message Attack
– *siehe* Superverschlüsselung
– *siehe* Wieners Angriff
– von Stromchiffren 188, 189
– *siehe auch* Tripel-Verschlüsselung 183

- kryptoanalytischer Angriff
siehe Kryptoanalyse
- Kryptographie 1–9, 145–193, 350–355, 403–462
 - asymmetrische *siehe* Kryptosystem, Public-Key-
 - gitter-basierte 393, 459, 461
 - Private-Key- *siehe* Kryptosystem, Private-Key-
 - Public-Key- *siehe* Kryptosystem, Public-Key-
 - symmetrische *siehe* Kryptosystem, Private-Key-
 - Worst-Case- 6, 407, 443, 460, 461
- Kryptologie 1–9
- Kryptosystem
 - asymmetrisches *siehe* Kryptosystem, Public-Key-
 - Chor–Rivest- *siehe* Chor–Rivest-Kryptosystem
 - ElGamal- *siehe* ElGamal-Kryptosystem
 - Entropie eines —s 178
 - Merkle–Hellman- *siehe* Merkle–Hellman-Kryptosystem
 - monoalphabetisches *siehe* monoalphabetisches Kryptosystem
 - NTRU- *siehe* NTRU-Kryptosystem
 - polyalphabetisches *siehe* polyalphabetisches Kryptosystem
 - Private-Key- 7, **146**
 - Public-Key- 6, 9, **146**
 - Rabin- *siehe* Rabin-Kryptosystem
 - RSA- *siehe* RSA-Kryptosystem
 - Sicherheit eines —s 3
 - symmetrisches, *siehe* Kryptosystem, Private-Key-
- k*-SAT **96**, 343
- Kumar, R. 191, 286, 460
- künstliche Intelligenz 283, 287
- Kuroda, S. 88
- Kurosawa, K. 457
- Kurtz, S. 125, 140, 141, 345
- Kurur, P. 335, 346
- L**
- L **70**, 79, 82, 93, 131, 141, 219, 274
- $L(\cdot)$ **65**, **29**
- \mathfrak{L}_0 **23**, 32
- \mathfrak{L}_1 *siehe* CS
- \mathfrak{L}_2 *siehe* CF
- \mathfrak{L}_3 *siehe* REG
- Laaser, W. 139
- Ladner, R. 120, 139, 271, 284
- Lagarias, J. 459
- Lagrange, J. 45, 361
 - *siehe auch* Satz, von Lagrange
- LaMacchia, B. 457
- Landers-Appell, C. VIII
- Landers-Appell, K. VIII
- Landry, F. 385
- Lange, K. 290
- Las-Vegas-Algorithmus *siehe* Algorithmus, Las-Vegas
- Lauscher *siehe* Erich
- Lautemann, C. 344
- Legendre-Symbol, *siehe auch* (\cdot) **47**
- Legitimate-Deck **347**, 348
- legitimes Deck *siehe* Deck, legitimes
- Leibert, M. VIII
- Leiserson, C. 59
- Le Lasseur, H. 385
- Lemma von Gibbs **180**
- Lenstra, A. 384, 385, 400, 459
- Lenstra Jr., H. 118, 384, 385, 387, 400, 454, 459
- LERC **334**, 335–337
- Lercher, M. VIII
- Leuschel, M. VIII
- Levin, L. 100, 138
- Lewis, P. 136
- LH **262**, 263, 264, 269, 270, 276
- Lien, Y. 139
- Lieven, P. VII
- $\mathbb{L}\text{in}$ **68**
- $\mathbb{L}\text{in}(\cdot)$ **68**
- Lindner, C. 5
- linear beschränkter Automat **30**

- lineare Beschleunigung *siehe* Satz,
 linearer Beschleunigungs-
- lineare Kompression *siehe* Satz,
 linearer Kompressions-
- lineare Programmierung **118**, 280,
 286, 286
 – ganzzahlige 118, 286
 – Problem der **118**
 – Problem der **118**
- linearer Raum
 – deterministischer *siehe*
 LINSPACE
 – nichtdeterministischer *siehe*
 NLINSPACE
- lineares Rückkopplungsschieberegister **172**
linear feedback shift register
 siehe lineares Rückkopplungs-
 schieberegister
- linear programming*
 siehe lineare Programmierung
- Linearzeit *siehe* LINTIME
- Linial, N. 282
- LINSPACE **70**, 79, 130
- LINTIME **70**, 76
- Lipton, R. 140, 278, 287
 – *siehe auch* Satz, von Karp und
 Lipton
- Lischke, G. VIII, 292
- Liśkiewicz, M. 143
- LLL-Algorithmus 459
- log 19
- $\log_r \alpha \bmod p$ **46**
- logarithmischer Raum
 – alternierender *siehe* AL
 – deterministischer *siehe* L
 – nichtdeterministischer *siehe* NL
- Logarithmusfunktion *siehe* Funkti-
 on, logarithmische
- Logik 5, **33–43**, 59
 – Aussagen- **33–39**
 – modale 287
 – nichtmonotone 283
 – Prädikaten- **39–43**
 – erster Stufe 43
- zweiter Stufe 43
- Long, T. 141, 287, 290, 291
- Longpré, L. 137, 288
- Lovász, L. 280, 459
- Low₀ 263
- Low₁ 263
- Low₂ 325, 328, 330, 332, 333
- Low-Exponent Attack **393**, 401
- Low-Hierarchie, *siehe auch* LH 8,
 8, 195, **262**, 263, 264, 269, 270,
 276, 290, 291
 – erste Stufe der *siehe* Low₁
 – erweiterte, *siehe auch* ELow_k
291
 – *k*-te Stufe der *siehe* Low_k
 – nullte Stufe der *siehe* Low₀
 – zweite Stufe der *siehe* Low₂;
 siehe auch Σ_2^p -low
- Low_k **262**, 263, 264, 268, 270, 276,
 328
- Lowness 262, 267, 290, 294, 326,
328, 330–337, 343, 344, 346
 – *siehe auch* Low-Hierarchie
 – *siehe auch* Selbst-Lowness
- Luby, M. 10, 190
- Lund, C. 282
- Lutz, D. VIII
- Lynch, N. 139, 284
- M**
- MA **322**, 324, 328, 339, 342, 344,
 347
 – *siehe auch* Arthur-Merlin-Spiele
- Mahaney, S. 123–125, 140, 141, 287,
 288
- Majoritätsregel **234**
 – Gewinner gemäß der
 siehe Condorcet-Gewinner
 – schlagen gemäß der **234**
- Majority-SAT **308**
- MAM **322**, 324, 339
 – *siehe auch* Arthur-Merlin-Spiele
- Manasse, M. 385
- Manders, K. 290

Man-in-the-Middle-Angriff *siehe Angriff, Man-in-the-Middle-*
 Many-one-Reduzierbarkeit *siehe Reduzierbarkeit, Many-one-*
 – *siehe auch* \leq_m^{\log}
 – *siehe auch* \leq_m^p
 Markoff-Kette **297**, 300
 Matching
 – bipartites 112
 – tripartites 112, 132
 Matching-Problem **111**
 – bipartites 112
 – dreidimensionales *siehe* 3-DM
 – tripartites *siehe* Matching-Problem, dreidimensionales
 – zweidimensionales *siehe* Matching-Problem, bipartites
 Matrix
 – adjungierte *siehe* adjungierte Matrix
 – Determinante einer, *siehe auch* det 143, **157**, 189
 – inverse **157**
 – Permanente einer, *siehe auch* $\text{perm}(\cdot)$ **143**
 Maurer, U. 461
 Mauve, M. VIII
maximal non-hamiltonian circuit **277**, 280
 – für gerichtete Graphen,
 siehe auch MNHC **277**, 280
 – für ungerichtete Graphen,
 siehe auch MDNHC **277**
 Max-SetPacking-Geq **235**, 275
 May, A. 401
 McColl, W. 136
 MDNHC **277**
 MEE **283**
 MEE-DNF **275**, **283**
 Mehrheitsentscheid *siehe* Majoritätsregel
 Mehrheitsquantor
 – *siehe* \exists^+
 – *siehe* Quantor, polynomiell längenbeschränkter, Mehrheits-

Menge, *siehe auch* Sprache
 – balanciert immune *siehe* balancierte Immunität
 – bi-immune *siehe* Bi-Immunität
 – co-endliche 269
 – disjunkte Vereinigung von —n
 siehe Menge, markierte Vereinigung von —n
 – dünne *siehe* Sprache, dünne
 – endliche 123, 269
 – entscheidbare **30**, 216
 – Gewinner- *siehe* choice set
 – immune *siehe* Immunität
 – isomorphe —n **122**
 – k-kreative **124**
 – kreative **125**
 – markierte Vereinigung von —n,
 siehe auch \oplus **291**
 – nicht-dünne **123**
 – Orakel- **33**, 215, 217
 - generische 291
 - zufällige 291
 – p-isomorphe —n
 siehe P-Isomorphismus
 – Potenzmenge einer, *siehe auch* $\wp(\cdot)$ **117**
 – P-printable *siehe* P-printability
 – P-selektive *siehe* P-Selektivität
 – rekursiv aufzählbare **31**, 32, 216, 265, 268
 – selbstreduzierbare *siehe* Selbstreduzierbarkeit
 – symmetrische Differenz von —n,
 siehe auch Δ **269**
 – von Wörtern bis zur Länge n,
 siehe auch $S^{\leq n}$ **123**, **246**
 Mengenklasse, *siehe auch* Komplexitätsklasse
 – Abschluss einer — unter endlicher Variation **269**, 270, 276
 – Abschluss einer — unter Komplement **199**
 – Abschluss einer — unter Schnitt **199**, 272, 291

- Abschluss einer — unter Vereinigung **199**, 272, 291
- boolesche Hülle einer, *siehe auch* $\text{BC}(\mathcal{C})$ **199**, 291
- co-Operator, angewandt auf eine, *siehe auch* $\text{co}\mathcal{C}$ **88**, **198**
- komplexer Schnitt von —n, *siehe auch* \wedge **198**, 271
- komplexe symmetrische Differenz von —n, *siehe auch* Δ **289**
- komplexe Vereinigung von —n, *siehe auch* \vee **198**, 271
- Mengenpackungsproblem, *siehe auch* **SetPacking** **111**, **117**
- Mengenring **199**, 272
- Mengenüberdeckungsproblem, *siehe auch* **SetCovering** **111**, **117**
- Merkle, R. 404, 439, 459
 - *siehe auch* Merkle–Hellman-Kryptosystem
- Merkle–Hellman-Kryptosystem 404, **439–442**, 453, 454, 459
 - iteriertes 459
 - Sicherheit des —s *siehe* Kryptanalyse, von Merkle–Hellman
- Merlin 5
 - *siehe auch* Arthur-Merlin-Spiele
- message* *siehe* Klartext
- message digest* 424
- message space* *siehe* Klartextrraum
- Meyer, A. 121, 140, 141, 283, 287
- Meyer, G. 279
- Meyer, T. VIII, 10
- Micali, S. 344, 346, 433, 437, 438, 457, 458
 - *siehe auch* Goldreich–Micali–Wigderson-Protokoll
- Micciancio, D. 10, 191, 393, 459
- Miller, G. 9, 349, 362–368, 375, 396, 398–400
 - *siehe auch* MILLER–RABIN
 - *siehe auch* Primzahltest, Miller–Rabin
- MILLER–RABIN** **363**
- Miller–Rabin–Lügner *siehe* MR–Lügner
- Miller–Rabin–Test *siehe* Primzahltest, Miller–Rabin
- Miller–Rabin–Zeuge *siehe* MR–Zeuge
- Mind-Change–Technik **242**
- $\min\text{-}\deg(\cdot)$ **110**, 206
- Minimal **283**
- Minimal-3–Uncolorability **279**, – eingeschränkt auf planare Graphen 279
- Minimal-3–UNSAT **277**, **279**
- Minimum–Equivalent–Expression–Problem, *siehe auch* MEE, MEE–DNF, Minimal 195, 275, **283**
- MNHC **277**, 280
- mod *siehe* Kongruenz modulo einer Zahl
- Monien, B. 295, 297
- monoalphabetisches Kryptosystem 150, 151, **153**, 155, 159, 162, 190–192
- Monoid **43**
 - abelscher **44**
 - kommutativer *siehe* Monoid, abelscher
- Monte-Carlo–Algorithmus *siehe* Algorithmus, Monte-Carlo
- Moore, J. 401
- Moran, S. 8, 344, 457
- Morgenstern, C. 184
- Morrison, M. 385
- Mothers of Invention 185
- Motwani, R. 59, 282
- MR–Liars (\cdot) **366**, 396
- MR–LIARS (\cdot) **367**, 368
- MR–Lügner **364**
- MR–Zeuge **364**
- Muchnik, A. 139
 - *siehe auch* Theorem, Friedberg–Muchnik
- Müller, H. VIII, 292
- Multimenge **234**

N**N 13**Nachrichtenauthentikation **148**Nachrichtenintegrität **148**Nader, R. **233**Nagelfeile *siehe* Werkzeuge, NagelfeileNaik, A. **140, 142, 461**Naor, M. **286, 460**Nasser, N. **281**natürliche Zahl *siehe* Zahl, natürliche Navajo-Code **191**NE **70, 82, 130, 143**NEA **25, 26, 55**Negation, *siehe auch* \neg **34, 35**

Netzwerk

– Kommunikations- **196**– Computer- **196**NEXP **70**NEXPSPACE **70**Nguyen, P. **191, 460**Nichtapproximierbarkeit **9, 139, 282, 286**nichtdeterministische Polynomialzeit
siehe NP

Nichtrest

– quadratischer, *siehe auch* QNR
46Niedermeier, R. **290**NIST **192, 457**NL **6, 70, 82, 88, 93, 94, 96, 99, 131, 139, 141**NLINSPACE **70, 88, 130**NL-vollständig **94, 96, 99, 131, 139, 141**– *siehe auch* \leq_m^{\log} -VollständigkeitNöckel, B. **VIII**Norris, M. **388, 401**NOTM **33**Nowak, M. **VII**NP **6, 70, 82, 89, 120–124, 130, 133, 137, 140, 142, 144, 198, 201, 202, 204, 209, 216, 219–222, 246, 262–266, 268, 271, 280, 285–289, 271, 273, 274,**

276, 277, 305, 312–314, 322, 325, 328, 329, 347, 397, 433, 446, 449, 458–462

– \leq_m^p -Hülle von, *siehe auch* P_{tt}^{NP}
240, 287

– Lösung einer Probleminstanz

siehe Zeuge– P-versus-NP-Frage **6, 64, 120–122, 138**– Zertifikat einer Probleminstanz
siehe ZeugeNP $^\mathcal{C}$ **219, 274**NP-hart **196, 210, 271, 280, 282–285**NP NP , *siehe auch* Σ_2^p **219, 221, 222, 275**NPOTM **33, 219**NP P **219, 274**NP PSPACE **219, 274**NPSPACE **70, 87**NP-vollständig **6, 8, 101, 104, 106, 108, 110, 113, 117, 119, 120, 123, 124, 131, 138, 139, 266, 271, 273, 276–280, 285, 288, 375, 376, 400, 404**– exakte Varianten von —en Problemen **7, 195, 197, 279**– *siehe auch* \leq_m^{\log} -Vollständigkeit– *siehe auch* \leq_m^p -Vollständigkeit– *siehe auch* Theorie der NP-VollständigkeitNSF **VIII**NSpace (\cdot) **67**NSPACE (\cdot) **68, 76, 85, 88, 129, 257, 260**NTIME (\cdot) **67**NTIME (\cdot) **68, 76, 85, 129, 253, 257**NTM **27, 65**NTRU-Kryptosystem **459**Nutzerauthentikation **148****O** $o(\cdot)$ **69** $\mathcal{O}(\cdot)$ **19, 39, 69** $\tilde{\mathcal{O}}(\cdot)$ **294**

- Odd- k -SAT **240**
 Odd-Max-SAT **284**
 Odd-SAT **276**
 Odifreddi, P. 59
 Odlyzko, A. 457, 459
 OFB-Modus **170**, 188
 Ogihara, M. VIII, 10, 72, 124, 136,
 140, 141, 143, 246, 281, 288,
 289, 341, 345
 Ogiwara, M. *siehe* Ogihara, M.
 One-time Pad
 siehe Vernams One-time Pad
one-way function
 siehe Einwegfunktion
 Orakel *siehe* Menge, Orakel-
 – *siehe auch* Turingmaschine,
 Orakel-
 Orponen, P. 291
 Ottmann, T. 59
output feedback mode
 siehe OFB-Modus
- P**
- Π_2^P , *siehe auch* coNP^{NP} **220**, 221,
 222, 283, 284, 289, 318, 320,
 324, 325, 347
 $\Pi_2^{P,\text{AM} \cap \text{coAM}}$ 324
 Π_2^P -vollständig 283, 284
 Π_3^P 314
 Π_i^P **220**, 221, 222, 224, 226, 289
 Π_i^P -vollständig 226, 246
 Π_i SAT **226**, 275
 Π_i SAT-Formel **226**
 P 6, 7, **70**, 80–82, 89, 120, 123,
 124, 126–128, 130, 137–141, 198,
 201, 202, 216, 219–221, 224,
 260–263, 271, 274, 294, 305,
 315, 328, 347, 374, 399, 400,
 460, 461
 – P-versus-NP-Frage 6, 64, **120–122**, 138
P **30**
 $\mathfrak{P}(\cdot)$ **117**
 \mathcal{P} **219**, 274
 P^{NP} , *siehe auch* Δ_2^P 219–222
- $P^A[k]$ **240**
 $P^{\mathcal{C}}[k]$ **240**
 $P^{\text{NP}}[k]$ **240**
 $P^{\text{NP}}[\mathcal{O}(1)]$ **240**, 244
 $P^{\text{NP}}[\log]$, *siehe auch* Θ_2^P **228**, **240**,
 244, 287, 288
 $P^{\Sigma_{i-1}^P[\mathcal{O}(\log)]}$ **228**
 $P_{k\text{-tt}}^A$ **240**
 $P_{k\text{-tt}}^{\mathcal{C}}$ **240**
 $P_{k\text{-tt}}^{\text{NP}}$ 285
 $P_{\text{bf}}^{\text{NP}}$ **240**, 244, 287
 $P_{k\text{-tt}}^{\text{NP}}$ **240**
 $P_{k\text{-tt}}^{\Sigma_i^P}$ 289
 $P_{k\text{-tt}}^{\mathcal{C}}$ **229**
 $P_{\text{tt}}^{\text{NP}}$ **240**, 244, 285, 287
 $P^{\text{NP}}^{\text{NP}}$, *siehe auch* Δ_3^P 221, 289
 P^P 219, 274
 P^{PP} 345, 347
 $P^{\text{PP}^{\text{PH}}}$ 345
 P^{PSPACE} 219, 274
 P^{SPP} 329
 Papadimitriou, C. 10, 59, 136, 197,
 228, 273, 278–280, 284, 344,
 345, 458
 paralleler Orakelzugriff 229
 paralleler Zugriff auf NP, *siehe auch*
 Θ_2^P ; $P^{\text{NP}}[\log]$; $P_{\text{tt}}^{\text{NP}}$ 227, **229**,
 232, 235, 287
 parallele Zeit 257
 Parberry, I. 345
 Pareto-Prinzip 233
parsimonious
 siehe Reduktion, geizige
 partiell rekursive Funktion *siehe*
 Funktion, partiell rekursive
 Pasanen, K. VIII, 453, 460
 Paterson, M. 121, 140
 Paturi, R. 295, 343
 Paula *siehe* Rothe, P.
 Pavan, A. 143
PCP **282**
 PCP-Theorem *siehe* Theorem, PCP-
 perfekte Geheimhaltung 7, **172–177**,
 189

$\text{perm}(\cdot)$ **143**

Permanente einer Matrix *siehe* Matrix, Permanente einer

Permutation, *siehe auch* $S_{(\cdot)}$ **48**
– Komposition von —en **48**

Permutationsgruppe **48**

– Generator einer **48**

– starker **48**

– Identität einer, *siehe auch* id **48**

– (punktweiser) Stabilisator in einer **48**

– rechte co-Menge einer **48**

– Turm von Stabilisatoren in einer **48**

– vollständige rechte Transversale in einer **49**

Petrrank, E. 140, 278

Pferschy, U. 459

PH 143, 195, **220**, 221, 222, 224, 226, 246, 263, 274, 281, 284, 287–289, 347

PHT **245**

Pipher, J. 459

Pisinger, D. 459

P-Isomorphie *siehe* P-Isomorphismus

P-Isomorphismus, *siehe auch* \cong_p **122**, 123, 125, 132

Poe, Edgar A. 145, 190

Pohlig, S. 411

Pol **68**

IPol(\cdot) **68**

POLLARD **377**

Pollard, J. 377, 378, 384, 385, 387, 397, 400, 411

– *siehe auch* Pollards ($p - 1$)-Methode

Pollards ($p - 1$)-Methode, *siehe auch* POLLARD **377**, 378, 384, 387, 397, 400

polyalphabetisches Kryptosystem **155**, 159, 160, 185

Polygamie 4, 112

POLYLOGSPACE **79**, 130

Polymerchemie 143

polynomialer Raum

– deterministischer *siehe* PSPACE

– nichtdeterministischer

siehe NPSPACE

Polynomial Hierarchy Tower, *siehe auch* PHT **245**

Polynomialzeit **70**

– alternierende *siehe* AP

– deterministische *siehe* P

– nichtdeterministische *siehe* NP

– probabilistische *siehe* PP

– mit beschränktem Fehler *siehe* BPP

– mit einseitigem Fehler *siehe* RP; coRP

– mit nullseitigem Fehler *siehe* ZPP

– *siehe auch* Probabilistic Polynomial Time; PP

– *siehe auch* Random Polynomial Time; RP

– *siehe auch* Stoic Probabilistic Polynomial Time; SPP

– *siehe auch* Unambiguous Polynomial Time; UP

Polynomialzeit-Hierarchie, *siehe auch*

PH 7, 143, 195, **220**, 221, 222, 224, 226, 245, 246, 262–264, 270, 283, 284, 287, 291, 294, 312, 315, 322, 324, 330, 333, 344–346

– Downward Collapse in der **289**

– i -te Stufe der, *siehe auch* Σ_i^P ; Π_i^P ; Δ_i^P **220**, 221, 222, 224, 226, 227, 246, 262, 263, 267, 268, 270, 276, 289, 291

– Kollaps der **224**, 245, 246, 263, 264, 270, 281, 284, 287–290, 333

– zweite Stufe der, *siehe auch* Σ_2^P ; NP^{NP} ; Π_2^P ; coNP^{NP} ; Δ_2^P ; P^{NP} ; Θ_2^P ; $\text{P}^{\text{NP}[\log]}$ **220**, 221, 222, 240, 275, 283, 284, 289, 291

Polytope(\cdot) **280**

Pomerance, C. 400

- Porta, G. 191
 Post, E. 139
 – *siehe auch* Postsches Problem
 Postsches Problem 139
 Potthoff, M. VIII
 PP 8, 143, 291, 293, **303**, 305, 308,
 312, 326–330, 334, 337, 339,
 341–345, 347
 PP-low 329, 345, 348
 PP_{path} 339, **342**
 P-printability **141**, 143
 PP^{SPP} 329
 PP-vollständig 8, 308
 Pr(.) **53**
 Pr(· | ·) **53**
 Prädikatensymbol **40**
 Präferenzordnung **234**
 Präferenzprofil **234**
 Präfixsuche 217, 248
 Pratt, V. 133
 Primalitätstest *siehe* Primzahltest
 Primes 356
 primitives Element *siehe* Zahl, Pri-
 mitivwurzel einer
 Primitivwurzel *siehe* Zahl, Primi-
 tivwurzel einer
 Primzahl *siehe* Zahl, Prim-
 Primzahl-Problem, *siehe auch* Primes
 8, 72, 121, 133, 349, **356**, 357–
 360, 363, 365, 370, 373, 374,
 376, 396, 399, 400
 Primzahltest 9, **355–375**
 – Fermat-, *siehe auch* FERMAT
 9, **358–362**, 364, 395
 – Miller–Rabin-, *siehe auch*
 MILLER-RABIN 9, 349, **362–**
 368, 396, 398, 400
 – Solovay–Strassen-, *siehe auch*
 SOLOVAY-STRASSEN 9, 349,
 368–374, 400
 Primzahl-Theorem *siehe* Theorem,
 Primzahl-
 Private-Key-Kryptographie *siehe*
 Kryptographie, Private-Key-
- Private-Key-Kryptosystem *siehe*
 Kryptosystem, Private-Key-
 probabilistically checkable proof sys-
 tem *siehe* PCP
 Probabilistic Polynomial Time *siehe*
 PP
 – Bounded-Error *siehe* BPP
 – One-sided Error *siehe* RP; coRP
 – Stoic *siehe* SPP
 – Zero-Error *siehe* ZPP
 Probedivision 286, **377**
 – *siehe auch* TRIAL-DIVISION
 Problem des Brechens von ElGamal
 siehe break-elgamal
 Problem des Brechens von Rabin
 siehe break-rabin
 Projektionstheorem
 siehe Theorem, Projektions-
 promise class *siehe* Promise-Klasse
 Promise-Klasse 64, **304**, 307, 310,
 327, 339, 345
promise problem
 siehe Promise-Problem
 Promise-Problem 346, 460
prover *siehe* Beweiser
 Protokoll
 – Authentikations- *siehe*
 Authentikationsprotokoll
 – Challenge-and-Response- *siehe*
 Challenge-and-Response-
 Protokoll
 – digitales Signatur-
 – ElGamal- *siehe*
 ElGamal-Signatur
 – Rabi–Sherman- *siehe*
 Rabi–Sherman-Signatur
 – RSA- *siehe* RSA-Signatur
 – ElGamal- *siehe*
 ElGamal-Kryptosystem
 – Merkle–Hellman- *siehe* Merkle–
 Hellman-Kryptosystem
 – Rabin *siehe*
 Rabin-Kryptosystem
 – RSA- *siehe*
 RSA-Kryptosystem

- Schlüsseltausch-
 - Diffie–Hellman- *siehe*
Diffie–Hellman-Protokoll
 - Rivest–Sherman- *siehe*
Rivest–Sherman-Protokoll
 - Shamirs No-Key- *siehe*
Shamirs No-Key-Protokoll
 - Zero-Knowledge- *siehe*
Zero-Knowledge-Protokoll
 - P-Selektivität 285, 290, 291
 - PSPACE 7, **70**, 79, 82, 87, 89, 130, 219, 220, 226, 227, 257, 272, 274, 305, 345, 347
 - PSPACE-vollständig 226, 257, 275
 - Public-Key-Kryptographie *siehe*
Kryptographie, Public-Key-
 - Public-Key-Kryptosystem *siehe*
Kryptosystem, Public-Key-
 - Pumping Lemma
 - für kontextfreie Sprachen **58**
 - für reguläre Sprachen **58**
 - P-vollständig 139, 141, 260, 261, 268, 269
- Q**
- \mathbb{Q} **56**
 - QBF **37**
 - QBF 225, 226, 227, 257, 275
 - QBF-Problem **225**
 - mit beschränkter Zahl von Alternierungen, *siehe auch* Σ_i SAT;
 Π_i SAT 7, **226**, 275
 - mit unbeschränkter Zahl von Alternierungen, *siehe auch* QBF
7, **225**, 226, 227
 - QBF_{simple} **275**
 - QNR **46**
 - $(\mathfrak{Q}_1 \mid \mathfrak{Q}_2)$ **314**
 - QR **46**, 418
 - QR_p **418**
 - quadratischer Nichtrest *siehe* Nichtrest, quadratischer
 - quadratischer Rest *siehe* Rest, quadratischer
- quadratisches Sieb 9, **378–383**, 387, 398, 400
 - quantifizierte boolesche Formel,
siehe auch QBF **37**
 - einfache, *siehe auch* QBF_{simple}
275
 - erfüllbare **42**
 - geschlossene **37**
 - gültige **42**
 - in Pränexform **38**
 - offene **37**
 - Quantor
 - existenzieller, *siehe auch* \exists ; \vee
37
 - polynomiel längenbeschränkter
 - existenzieller, *siehe auch* \exists^P
215, **216**, 221, 224
 - Mehrheits-, *siehe auch* \exists^+ **315**, 316–333, 339
 - universeller, *siehe auch* \forall^P 215, **216**, 221, 224
 - universeller, *siehe auch* \forall ; \wedge **37**
 - Quantorenfolge 314
 - durch —n definierte Komplexitätsklasse, *siehe auch* $(\mathfrak{Q}_1 \mid \mathfrak{Q}_2)$ **314**
 - vernünftiges Paar von —n **314**
 - Query Order 290
- R**
- \mathbb{R} **30**, 66
 - \mathbb{R} **56**
 - $R_{(\cdot)}$ **32**
 - $\mathcal{R}_{(\cdot)}$ *siehe* Sprache, Redundanz einer
 - Rabi, M. 404, 442, 453, 454, 460
 - *siehe auch* Rabi–Sherman-Signatur
 - Rabin, M. 9, 26, 59, 73, 349, 362–368, 375, 396, 398–400, 403, 424–430, 451, 457
 - *siehe auch* MILLER–RABIN
 - *siehe auch* Primzahltest, Miller–Rabin–
 - *siehe auch* Rabin-Kryptosystem
 - *siehe auch* Satz, von Rabin
 - Rabin-Kryptosystem 403, **424–430**, 451, 457

- Rabins Satz *siehe* Satz, von Rabin
 Rabi–Sherman-Signatur 453
 – Sicherheit der 460, 461
 Rackoff, C. 141, 344, 457
 Radziszowski, S. 348
 Rajasethupathy, K. 286
RANDOM-FACTOR 429, 430
 randomisierter Algorithmus *siehe*
 Algorithmus, randomisierter
 Random Polynomial Time *siehe* RP
RANDOM-SAT 294, 299, 301, 339,
 343
random walk *siehe* Irrfahrt
 Ranjan, D. 284
 Rao, R. VIII, 137
 rationale Zahl *siehe* Zahl, rationale
 Raumfunktion *siehe* Funktion,
 Raum-
 Raumhierarchie 6, 77, 136
 Raumhierarchiesatz
 siehe Satz, Raumhierarchie-
 raumkonstruierbar *siehe* Funktion,
 raumkonstruierbare
 Razborov, A. 345
RE 31, 32, 139, 216, 265
REALTIME 70, 76
 rechte co-Menge *siehe* Permutations-
 gruppe, rechte co-Menge einer
 – *siehe auch* LERC 48
 Reduktion
 – geizige 127, 136
 – *siehe auch* Reduzierbarkeit
 Reduzierbarkeit 6
 – Many-one-
 - Log-Space- *siehe auch* \leq_m^{\log}
 63, 91, 93, 94, 96, 99, 100, 139,
 141, 219, 260, 261, 274, 284
 - polynomialzeit-beschränkte,
 siehe auch \leq_m^P 63, 89, 90, 131,
 138, 210, 219, 220, 266, 267,
 269, 274, 276, 284, 288, 308,
 312, 330, 333, 339, 348
 - randomisierte 280, 287
 – polynomialzeit-beschränkte
 Truth-table-, *siehe auch* \leq_{tt}^P
 228, 230, 284, 288
 – disjunktive 288
 – polynomialzeit-beschränkte
 Turing-
 - deterministische, *siehe auch* \leq_T^P
 143, 218, 220, 267, 274, 276,
 284, 287, 288, 345, 408, 417
 - nichtdeterministische, *siehe*
 auch \leq_T^{NP} 219, 219, 284, 274
 - positive, *siehe auch* \leq_{pos-T}^P 219,
 274, 284
 - randomisierte 397, 400, 428
 - stark nichtdeterministische,
 siehe auch \leq_{sT}^{NP} 262, 266, 276,
 290
 – *siehe auch* γ -Reduzierbarkeit
 – *siehe auch* Selbstreduzierbarkeit
 reelle Zahl *siehe* Zahl, reelle
 Reflexivität 57, 132
REG 23, 58
 Reingold, N. 342–344
 Reischuk, R. 10, 136
 Reith, S. 289
 rej_M 326
 rekursive Aufzählbarkeit *siehe*
 Sprache, rekursiv aufzählbare
 – *siehe auch* RE
 rekursiv präsentierbar 268, 270, 276
 relativierte Welt
 – *siehe* Menge, Orakel-
 – *siehe* Relativierung
 Relativierung 291
 Ressource *siehe* Komplexitätsmaß
 Ressourcenfunktion *siehe* Komple-
 xitätsklasse, Ressourcenfunktion
 einer
 Rest
 – quadratischer, *siehe auch* QR 46
 – *siehe auch* Restklasse
 Restklasse 58
 Ribordy, G. 10
 Riege, T. VII, VIII, 5, 10, 282, 343
 Rijmen, V. 193

Ring 44

- der Eine 5, 432
- Einselement eines —s 44
- Invertierbarkeit in einem 44
- kommutativer 44
- mit Eins 44
- Nullelement eines —s 44

Ringautomorphismus 400

- Zählproblem für —men,
siehe auch #RA 400

Rivest, R. 59, 193, 349, 350, 399,

400, 404, 442, 443, 445, 446,
449, 453, 459–461

- *siehe auch Chor–Rivest–Kryptosystem*
- *siehe auch Rivest–Sherman–Protokoll*
- *siehe auch RSA–Kryptosystem*
- *siehe auch RSA–Signatur*

Rivest–Sherman–Protokoll 404,

442–449

- Sicherheit des —s 460, 461

Robshaw, M. 401**Rogers, J. 126, 141, 142, 461****Rogers Jr., H. 59, 122, 265, 290****Rohatgi, P. 281, 287****Rolf, D. 295, 343****Rosen, A. 59****Rosenberg, A. 76****Rossmannith, P. 290****Rothe, E. V, VIII, 80, 81, 156, 157,
186, 245, 246, 455, 456****Rothe, I. V, VIII****Rothe, J. 10, 136, 137, 140, 142,
143, 191, 199, 232, 278, 281–
286, 290, 291, 343, 345, 346,
401, 453, 454, 458, 460–462****Rothe, P. V, VIII, 80, 81, 162, 186,
245, 246, 455, 456****Royer, J. 125, 140, 141****RP 8, 281, 288, 293, 302, 303, 304–
307, 312, 326, 339, 341, 343,
347, 362****RP_(.) 306, 396****RP_{path} 304, 339, 342****RSA-*d*-Zahl *siehe* Zahl, RSA-*d*****RSA–Kryptosystem 6, 8, 121, 349,
350–354, 359, 375–377, 384, 399,
403, 427, 439**

- Sicherheit des —s *siehe* Kryptanalyse, von RSA

RSA–Signatur 8, 355, 395, 420

- Fälschung von —en 383
- Sicherheit der *siehe* Kryptanalyse, von RSA

**RSA–Superverschlüsselung 388, 398,
401****Rubinstein, R. 141****Rucksack–Problem 6, 111, 117**

- hoher Dichte 459

- niedriger Dichte 459

Rueppel, R. 191**Russel, A. 288****Russo, D. 291, 345****S** **Σ 19** **Σ^* 19** **$\mathfrak{S}_{(.)}$ 48** **Σ_2^p , *siehe auch* NP^{NP} 220, 221,
222, 275, 283, 284, 289, 291,
318, 320, 324, 330, 333, 343,
346, 347** **$\Sigma_2^{p,\text{AM}\cap\text{coAM}}$ 324, 328** **Σ_2^p -low, *siehe auch* Low₂ 324,
328** **Σ_2^p -vollständig 226, 275, 283, 284** **Σ_3^p 314** **Σ_i^p 220, 221, 222, 224, 226, 227,
246, 262, 263, 267, 268, 270,
276, 289, 291** **Σ_i^p -vollständig 226, 267, 268, 276** **Σ_i SAT 226, 275** **Σ_i SAT-Formel 226** **$S^{\leq n}$ 123, 246** **S_2^p 288****Saari, D. 233****Sach- und Autorenverzeichnis 4, 5****Safra, S. 282**

- Salomaa, A. 10, 59, 153, 190, 344, 399, 457, 459
- Saruman 433–439, 452
- SAT 64, **96**, 101, 122, 126–128, 247, 280, 293–301, 312
- SAT-UNSAT **247**, 271
- Satz
- Chinesischer Reste- **47**
 - Fermats Kleiner **45**, 46, 352, 358, 364, 365, 377, 415
 - linearer Beschleunigungs- 6, 63, 68, 72, **74**, 136
 - für nichtdeterministische Klassen **76**
 - linearer Kompressions- 6, 63, **73**, 136
 - für nichtdeterministische Klassen **76**
 - Projektions- **216**
 - Raumhierarchie- 6, **77**, 136
 - Vietascher Wurzel- 388
 - von Bayes **53**
 - von Borodin und Demers 142
 - von Cantor und Bernstein 122
 - von Cook 64, **101**, 127, 128, 131, 138
 - von Cook und Levin *siehe* Satz, von Cook
 - von Euler **45**
 - von Lagrange **45, 361**
 - von Rabin **73**
 - von Savitch **85**, 226, 256, 275
 - von Shannon **174**, 176
 - Zeithierarchie- 6, **80**, 136
- Savitch, W. 85, 137, 139, 226, 256, 275
- *siehe auch* Satz, von Savitch
- Saxe, J. 345
- Saxena, A. VIII, 460
- Saxena, N. 121, 133, 349, 374, 375, 385, 399, 400
- SCF **234**
- Schaefer, M. 283
- Schaltkreise polynomiauer Größe 287, 290
- Schaltkreiskomplexität 136,
- Schäuble, W. 183, 192
- Schlüssel **145**
 - falscher **190**
 - privater **146**
 - öffentlicher **146**
- Schlüsselmehrdeutigkeit 172, 177, **181**, 182
- Schlüsselraum **145**
- Schlüsselstrom *siehe* Chiffre, Strom-Schlüsseltausch 9, 350, 399, **404**
 - *siehe auch* Protokoll, Schlüssel-tausch-
 - *siehe auch* Schlüsseltausch-Problem
- Schlüsseltausch-Problem 350, **404**
- Schlüsseltext **145**
- Schlüsselextraum **145**
- Schlüter, T. 5
- Schneider, D. 5
- Schneier, B. 5, 10, 190
- Schnorr, C. 140, 284, 414, 457, 459
- Schöning, U. VIII, 10, 59, 120, 139, 244, 262, 265, 267, 270, 281, 286, 287, 290, 291, 295, 297, 324, 325, 333, 341, 343–346, 348, 458
- Schöttner, M. VIII
- schwach assoziativ *siehe* Assozia-tivität, schwache
- Scott, D. 26, 59
- Seara, C. 287
- search reducing to decision* 140
- second-order logic* *siehe* Logik, Prä-dikaten-, zweiter Stufe
- secret-key agreement problem* *siehe* Schlüsseltausch-Problem
- s-Ehrlichkeit *siehe* Funktion, s-ehr-liche
- selektive Fälschung *siehe* Fälschung, selektive
- Selbst-Lowness **328**, 329, 341, 343, 345
- Selbstreduzierbarkeit **121**, 124, 129, 139, 285

- disjunktive **122**
- Selbstreduzierbarkeitsbaum **122**
- Self-Avoiding-Walk-Problem **143**
- Selman, A. 10, 59, 136, 140–143, 191, 262, 284, 290, 291, 346, 460, 461
- Sendergruppe **196**
- Separation 291
 - Downward *siehe* Upward Collapse
 - durch immune Mengen *siehe* Separation, starke
 - starke 291
 - Upward *siehe* Upward Separation
- sequenzieller Raum 257
- Set Covering **117**
- Sethupathy, P. 286
- Set Packing **117**, 132
- Sewelson, V. 137
- Shamir, A. 275, 345, 349, 350, 399–401, 438, 439, 452, 454, 456–460
 - *siehe auch* Fiat–Shamir-Protokoll
 - *siehe auch* RSA-Kryptosystem
 - *siehe auch* Shamirs No-Key-Protokoll
- Shamir, R. 138
- Shamirs No-Key-Protokoll **455**, 456
- SHANKS **410**, 411, 418, 450
- Shanks, D. 410, 411, 418, 450, 457
 - *siehe auch* SHANKS
- Shannon, C. 7, 172, 174, 176, 177, 192
 - *siehe auch* Satz, von Shannon
- Sherman, A. 193, 404, 442, 443, 445, 446, 453, 454, 460, 461
 - *siehe auch* Rabi–Sherman-Signatur
 - *siehe auch* Rivest–Sherman-Protokoll
- Sheu, M. 291
- Shoenfield, J. 59
- Shortest-Lattice-Vector-Problem 461
- Sieb des Eratosthenes **356**, 376, 377
- Silverman, J. 459
- Simmons, G. 388, 401
- Simon, J. 343, 345
- simulated annealing* 286
- Singh, S. 2, 190, 191, 400
- Sipser, M. 77, 138, 287, 344–346
 - *siehe auch* Sipser Coding Lemma
- Sipser Coding Lemma **346**
- Sivakumar, D. 141, 143, 191, 286, 460
- Small-Message Attack **389**
- Smolensky, R. 345
- Soare, R. 290
- Social-Choice-Funktion, *siehe auch* SCF **234**
 - Condorcet- *siehe* Condorcet-SCF
- Social-Choice-Theorie *siehe* Theorie, Social-Choice
- Solovay, R. 292, 349, 368–374, 400
- SOLOVAY–STRASSEN **370**, 372
- Solovay–Strassen-Lügner *siehe* SS-Lügner
- Solovay–Strassen-Test *siehe* Primzahltest, Solovay–Strassen
- Solovay–Strassen-Zeuge
 - siehe* SS-Zeuge
- SOS **118**, 119, 132, 440, 442, 446, 453
 - Größen einer —-Instanz **118**
 - Zielsumme einer —-Instanz **118**
 - *siehe auch* superwachsende Folge
- Spaan, E. *siehe* Hemaspandra, E.
- space_(.)(.) **65**, 129
- Space_(.)(.) **65**
- Spakowski, H. VII, VIII, 232, 283, 285, 286
- Spam 286
- sparse* *siehe* Sprache, dünne
- Speckenmeyer, E. 295, 297
- Spezial-Faktorisierungsmethoden **387**
- Spielman, D. 342, 344
- SPP **143**, **327**, 329, 330, 334, 337, 339, 343, 345–347
- SPP^{SPP} 329
- Sprache **19**

- dünne 123, 124, 137, 141, 143, **246**, 287, 291
- Durchschnitt von —n, *siehe auch* \cap **20**
- formale **19**
- Iteration einer, *siehe auch** **20**
 - ε -freie, *siehe auch* + **20**
- Kardinalität einer, *siehe auch* $\|\cdot\|$ **20**
- Kleene-Hülle einer *siehe* Sprache, Iteration einer
- Komplement einer, *siehe auch* $\bar{\cdot}$ **20**
- Konkatenation von —n **20**
- kontextfreie, *siehe auch* CF; \mathfrak{L}_2 **23**, 58
- kontextsensitive, *siehe auch* CS; \mathfrak{L}_1 **23**, 88
- nichttriviale **90**
- Operation auf —n **20**
- Reduktion auf eine 287, 288
 - *siehe auch* Reduzierbarkeit
- Redundanz einer, *siehe auch* $\mathcal{R}_{(\cdot)}$ 189, 192
- reguläre, *siehe auch* REG; \mathfrak{L}_3 **23**, 58
- rekursiv aufzählbare, *siehe auch* RE **31**, 32
- *tally* **81**, 142, 144
 - Binärdarstellung einer, *siehe auch* Bin(\cdot) **81**
- Typ-0-, *siehe auch* \mathfrak{L}_0 **23**, 32
- Typ einer **23**
- Unärcodierung einer, *siehe auch* Tally(\cdot) **81**
- Vereinigung von —n, *siehe auch* \cup **20**
- Sprung-Operator **265**
- SQUARE-AND-MULTIPLY **353**
- Srinivasan, A. 139
- SS-Liars(\cdot) **372**, 374
- SS-Lügner **372**
- SS-Zeuge **372**
- starke Exponentialzeit-Hierarchie
 - Kollaps der 144
- starke Nichtinvertierbarkeit *siehe* Einwegfunktion, starke statistische Physik 143
- Stearns, R. 80, 136, 137
- Steiglitz, K. 136
- Stein, C. 59
- Stelzer, A. VIII, 5
- Stephan, F. 284
- Stern, J. 191, 460
- Stinson, D. 10, 190–192, 344, 384, 398–401, 412, 454, 457, 458
- stochastischer Automat
 - siehe* endlicher Automat, stochastischer
- Stöcker, P. 5
- Stockmeyer, L. 138, 282, 283, 290
- Stoic Probabilistic Polynomial Time
 - siehe* SPP
- Stoyan, D. VIII
- Strassen, V. 349, 368–374, 400
- strategisches Wählen *siehe* Wahl-system, Manipulation eines —s
- Stromchiffre *siehe* Chiffre, Strom-
- Stromnes, M. VIII
- Struktur **40**
- Subset-of-Sums-Problem, *siehe auch* SOS **118**, 119, 132, 440, 442, 446, 453
- Substitutionsangriff
 - siehe* Angriff, Substitutions-
- Suchmaschine 286
- Sudan, M. 282
- Sundaram, R. 288
- superwachsende Folge **440**, 442, 453
- Symmetrie **57**, 132
- symmetrische Alternierung, *siehe auch* S_2^p **288**
- symmetrische Differenzenhierarchie 281
 - *siehe auch* boolesche Hierarchie, Normalformen der
- symmetrische Kryptographie
 - *siehe* Kryptographie, Private-Key-
 - *siehe* Kryptographie, symmetrische

- symmetrisches Kryptosystem
 – *siehe* Kryptosystem,
 Private-Key-
 – *siehe* Kryptosystem, symmetrisches
- Szegedy, M. 282
- Szelepcsényi, R. 88, 139
- T**
- Θ_2^p , *siehe auch* $P^{NP[\log]}$ **228**, 230, 236, 239, 240, 244, 275, 283, 285–288, 345, 347
- Θ_2^p -hart 230, 235, 275, 283
- Θ_2^p -vollständig 230, 236, 239, 275, 285, 287
- Θ_i^p , *siehe auch* $P^{\Sigma_{i-1}^p[\mathcal{O}(\log)]}$ **228**
- Takeuchi, M. 457
- tally *siehe* Sprache, tally
- Tally(\cdot) **81**, 130
- TALLY **81**
- Tamaki, S. 295, 343
- Tantrix™-Rotation-Puzzle-Problem
133–136
 – *siehe* 2-TRP
 – *siehe* 3-TRP
 – *siehe* 4-TRP
- Tarjan, R. 138
- Tarui, J. 345
- Tautologie 35, **36**, 56
- Tautologieproblem **275**
- Tautologieregel **36**
- Tchernin, A. 5
- Technik
 – Easy-Hard-
 siehe Easy-Hard-Technik
 – Mind-Change-
 siehe Mind-Change-Technik
 – Wagner- *siehe* Wagner-Technik
- Teile-und-Herrsche-Strategie **14**, 297
- Telle, J. 282
- Tenenbaum, P. 5
- Term **40**
- Thakur, M. 142, 461
- Theorem
 – Friedberg–Muchnik- 139
 – Karp–Lipton- 287, 288
- PCP- 282
- Primzahl- **353**, **356**, 383
- Unmöglichkeits- **233**
- Theorie
 – Berechenbarkeits- 2, 5, 11, **19**–
 33, 59, 122, 124, 216, 265, 284, 290, 291
 – der Datenkompression 177
 – der formalen Sprachen 11, **19**–
 33, 59
 – der NP-Vollständigkeit **100–120**, 131, 138
 – Graph- 5, 11, **43–47**, 59
 – Informations- und Codierungs- 7, 177
 – Komplexitäts- 1–9, 61–144, 195–
 348, 355–385, 430–449
 – Lern- 284
 – Social-Choice- 233, 234, 286
 – Wahrscheinlichkeits- 5, 11, **52**–
 54, 59
 – Zahlen- 5, 11, **43–47**, 59
- Thermodynamik 177
 – zweites Prinzip der 177
- Thierauf, T. 284, 343
- Threlfall, R. 400
- Threshold-SAT **308**
- time_(.)(\cdot) **65**, 73, 129
- Time_(.)(\cdot) **65**
- Tittel, W. 10
- Toda, S. 143, 284, 291, 345
- Tomaszewski, J. VIII
- Torán, J. 139, 288, 291, 341, 344–
 346, 348, 458
- Torenvliet, T. 140, 284, 285
- totale Funktion *siehe* Funktion, totale
- totaler Bruch **420**
- Tovey, C. 285, 286
- Transducer **269**
- Transitivität **57**, 132
- Traveling-Salesperson-Problem,
siehe auch TSP **280**
- Traveling-Salesperson-Tour **279**
 – eindeutige optimale 284

TRIAL-DIVISION 357

Trick, M. 285, 286

tripartites Matching-Problem

siehe Matching-Problem, dreidimensionales

Tripathi, R. 348

Triple-DES 193

Tripel-Verschlüsselung 165

Trithemius, J. 191

Trusted Third Party *siehe auch* TTP 431Truth-table-Hülle von NP *siehe* NP, \leq_{tt}^p -Hülle von – *siehe auch* P $_{tt}^NP$ Truth-table-Reduzierbarkeit *siehe* Reduzierbarkeit, polynomialzeitbeschränkte Truth-table-
– *siehe auch* \leq_{tt}^p

TSP 280

TSP-Facet 277, 280

TTP 431

Turing, A. 2, 26, 59

Turing Award 59, 137, 138, 399

Turinggrad 139

Turing-Hülle *siehe* Komplexitätsklasse, \leq_T^p -Hülle einer
– *siehe auch* P C

Turingmaschine 2, 4, 11, 26, 27, 28, 31–33, 55, 59, 61

– Akzeptierungsmodus einer 62, 253

– Alphabet einer

- Arbeits- 28

- Eingabe- 28

– alternierende, *siehe auch* ATM 7, 61, 62, 67, 120, 195, 249, 251, 252, 257, 290– Adressregister einer *siehe* Turingmaschine, alternierende, Indexband einer

– akzeptierender alternierender Teilbaum einer 249, 250

– Bewertungsfunktion einer, *siehe auch* eval(·) 250

– Indexband einer 253

– Semantik einer 250

– Sprache einer 250

– Syntax einer 249

– Berechnung einer 65

– ablehnende 29

– akzeptierende 29

– deterministische, *siehe auch* DTM 28, 62, 65

– Berechnung einer 65

– effektive Nummerierung von

 —n *siehe* Turingmaschine, Gödelisierung von —n

– Einweg- 62

– Gödelisierung von —n 31, 72, 77, 80, 269

– kategorische *siehe* Turingmaschine, nichtmehrdeutige– Komposition von —n, *siehe auch* o 307

– Konfiguration einer 28, 65

– ablehnende 250, 294

– akzeptierende 250, 294

– End- 29, 65

– existenzielle 250

– Start- 29, 65

– universelle 250

– Kreuzungsfolge einer, *siehe auch* cs(|·|) 133– Leerzeichen einer, *siehe auch* □ 28

– Mehrband- 62

– nichtdeterministische, *siehe auch* NTM 27, 62, 65

– Berechnung einer 65

– nichtmehrdeutige 62, 126

– normalisierte 303, 341

– Orakel- 33, 215, 217

– deterministische Polynomialzeit-, *siehe auch* DPOTM 218– nichtdeterministische Polynomialzeit-, *siehe auch* NPOTM 219

– positive 274

– *siehe auch* DOTM; NOTM

– probabilistische 62, 302, 343

- randomisierte 433
 siehe auch Turingmaschine, probabilistische
 - Schwellwert- **303**, 343
 - Semantik einer **28**
 - Sprache einer, *siehe auch* $L(\cdot)$
29, 65
 - Syntax einer **27**
 - Überführungsfunktion einer **28**
 - *unambiguous* *siehe* Turingmaschine, nichtmehrdeutige
 - Zustand einer **28**
 - ablehnender **29**, 249
 - akzeptierender **29**, 249
 - End- **28**
 - existenzieller 249
 - Start- **28**
 - universeller 250
 - Zweiwege- 62
 - Turing-Reduzierbarkeit *siehe* Reduzierbarkeit, polynomialzeit-beschränkte Turing-
– *siehe auch* \leq_T^p
 - U**
 - Ulam, S. 346
 - Ullman, J. 59
 - Umans, C. 283
 - Umanski, O. 5
 - Unabhängige-Menge-Problem,
 siehe auch IS **106**, 107, 131
 - Approximationsheuristiken für das 286
 - Unabhängigkeit irrelevanter Alternativen 233
 - Unabhängigkeitszahl-Probleme **228**
 - *siehe* IN-Equ **228**, 230, 275
 - *siehe* IN-Geq **228**, 230, 275, 285
 - *siehe* IN-Odd **228**, 230, 275, 285
 - Unambiguous Polynomial Time
 siehe UP
 - Unerfüllbarkeitsregeln **36**
 - Unique-SAT 273, **280**
 - United States Digital Signature Standard 9, 414, 457
 - Universal-Faktorisierungs-methoden **387**
 - *siehe auch* Zahlkörpersieb 384, 385, 387, 400
 - universeller Quantor *siehe* Quantor, universeller
 - polynomiel längenbeschränkter
 siehe \forall^p
 - Unizitätsabstand 189, 192
 - Unmöglichkeitstheorem *siehe* Theorem, Unmöglichkeits-UP 6, **126**, 127, 128, 132, 137, 141, 142, 199, 272, 281, 305, 328, 329, 341, 345, 347, 376, 400, 446, 460, 461
 - Upward Collapse **201**, 224
 - Upward Separation **81**, 137, 143
 - Grenzen der 137
 - US-Kongress 286
 - U.S. Presidential Election 233
 - US-Verteidigungsminister 430
- V**
- V**(.) **54**
 - V**(.) **93**
 - Valiant, L. 126, 141–143, 280, 345
 - van Helden, P. 140
 - Varianz, *siehe auch* V(.) **54**
 - Vaudenay, S. 459
 - Vazirani, V. 9, 279, 280, 282, 284, 345
 - VC **106**, 131
 - Vereshchagin, N. 344
 - verifier* *siehe* Verifizierer
 - Verifizierer 344, 458
 - ehrlicher 458
 - unehrlicher 458
 - *siehe auch* Beweissystem, interaktives
 - *siehe auch* Zero-Knowledge-Protokoll
 - Vernam, G. 172, 176
 - *siehe auch* Vernams One-time Pad
 - Vernams One-time Pad 172, **176**

- verschachtelte Differenzenhierarchie 199, **200**, 281
 – *siehe auch* boolesche Hierarchie, Normalformen der
- Verschiebungschiffre *siehe* Chiffre, Verschiebungs-
- Vieta, F. 388
 – *siehe auch* Satz, Vietascher Wurzel-
- Vietascher Wurzelsatz *siehe* Satz, Vietascher Wurzel-
- Vigenère, B. de 155
 – *siehe auch* Vigenère-Chiffre
 – *siehe auch* Vigenère-Quadrat
- Vigenère-Chiffre *siehe* Chiffre, Vigenère-
- Vigenère-Quadrat **156**, 185
- Vogel, J. VIII, 232, 283, 285
- Voigt, L. VIII
- Vollmer, H. 10, 90, 136
- Vollständigkeit
 – *siehe* \leq_m^{\log} -Vollständigkeit
 – *siehe* \leq_m^P -Vollständigkeit
 – *siehe* \leq_T^P -Vollständigkeit
 – *siehe* \leq_{tt}^P -Vollständigkeit
 – *siehe* #P-Vollständigkeit
 – *siehe* #P₁-Vollständigkeit
 – *siehe* BH_k(NP)-vollständig
 – *siehe* coNP-vollständig
 – *siehe* Δ₂^P-vollständig
 – *siehe* DP-vollständig
 – *siehe* NL-vollständig
 – *siehe* NP-vollständig
 – *siehe* Π₂^P-vollständig
 – *siehe* Π_i^P-vollständig
 – *siehe* P-vollständig
 – *siehe* PP-vollständig
 – *siehe* PSPACE-vollständig
 – *siehe* Σ₂^P-vollständig
 – *siehe* Σ_i^P-vollständig
 – *siehe* Θ₂^P-vollständig
- vollständige rechte Transversale **49**, 338
- von Haeseler, A. VIII
- von Neumann, J. 138
- Vyskoč, J. 290
- W**
- Wachstumsrate *siehe* Funktion, Wachstumsrate einer
- Wagner, K. VIII, 10, 88, 136, 203, 204, 230, 244, 246, 281, 283, 285–287, 289, 345
 – *siehe auch* Wagner-Technik
- Wagner-Technik **203**, 209, 214, 215, **230**, 232, 283
- Wähler 234
- Wahlsystem 232, **234**, 285
 – Eigenschaften eines —s 285
 – *siehe* Condorcet-Prinzip
 – *siehe* diktatorisch
 – *siehe* Pareto-Prinzip
 – *siehe* Unabhängigkeit irrelevanter Alternativen
 – *siehe* Wahlsystem, Homogenität eines —s
 – *siehe* Wahlsystem, Monotonität eines —s
 – Homogenität eines —s **286**
 – Manipulation eines —s 286
 – Monotonität eines —s 233
 – *siehe auch* Dodgson-Wahlsystem
 – *siehe auch* Kemeny-Wahlsystem
 – *siehe auch* Majoritätsregel
 – *siehe auch* Young-Wahlsystem
- Wahrscheinlichkeit, *siehe auch* Pr(·)
53
 – bedingte, *siehe auch* Pr(· | ·) **53**
- Wahrscheinlichkeitsraum **52**
- Wahrscheinlichkeitstheorie *siehe* Theorie, Wahrscheinlichkeits-Wahrscheinlichkeitsverstärkung 8, 306, 310
- Wahrscheinlichkeitsverteilung **52**
 – *siehe auch* Gleichverteilung
- Wang, J. 6
- Wanke, E. VIII, 282
- Watanabe, O. VIII, 124, 137, 140, 141, 288, 291, 292
- Website-Ranking 286

- Manipulation eines —s 286
- Wechsung, G. VII, VIII, 10, 88, 136, 137, 140, 142, 143, 195, 278, 281, 283, 284, 286, 290, 344, 345, 458, 462
- Wegener, I. 10, 136, 458
- Wegman, M. 330, 346
- Welsh, D. 10, 143, 190
- Werkzeuge
 - Axt 62
 - Kettensäge 4, 62
 - Nagelfeile 62
 - *siehe* Turingmaschine
- Wichert, H. VII
- Widmayer, P. 59
- Wiener, M. 389, 391, 398, 401
 - *siehe auch* Wieners Angriff
- Wieners Angriff 389, **391**, 398, 401
- Wigderson, M. 346, 433, 437, 438, 458
 - *siehe auch* Goldreich–Micali–Wigderson-Protokoll
- Williams, H. 457
- Williamson, M. 399, 400
- $\text{Wit}_{(\cdot)}(\cdot)$ **216**
- Woeginger, G. 9, 282, 343
- Wolf, S. 461
- Wolfe, D. 279, 280
- Wollermann, O. 5
- Wolters, I. 5
- Worst-Case-Kryptographie *siehe* Kryptographie, Worst-Case-Wort **19**
 - hartes **247**
 - Konkatenation von Wörtern **20**
 - Länge eines, *siehe auch* $|\cdot|$ **19**
 - leeres, *siehe auch* ε **19**
 - leichtes **247**
 - Operation auf Wörtern **20**
- Wrathall, C. 283
- Wright, E. 59

X

- $\chi(\cdot)$ 108
- X-3-Cover **117**

– *siehe auch* exaktes Überdeckungsproblem durch Dreiermengen

XP 345

Y

- Yacobi, Y. 346, 460
- Yamakami, T. 292
- Yamamoto, M. VIII
- Yannakakis, M. 197, 273, 278–280
- Yap, C. 246, 287
- Young, H. 232, 235, 285, 286
 - *siehe auch* Young-Wahlsystem
- Young, P. 124, 125, 140, 141
- Young-Gewinner **235**
- YoungRanking **235**, 275
- Young-Score,
 - *siehe auch* YScore(\cdot, \cdot, \cdot) **235**
- Young-Wahlsystem 232, **235**, 285
 - homogene Variante des —s 286
 - Rankingproblem für das,
 - *siehe auch* YoungRanking **235**, 275
 - Gewinnerproblem für das,
 - *siehe auch* YoungWinner 232, **235**, 239, 275
- YoungWinner 232, **235**, 239, 275
- YScore(\cdot, \cdot, \cdot) **235**

Z

- \mathbb{Z} **13**
- \mathbb{Z}^+ **279**
- $\mathbb{Z}_{(\cdot)}$ **44**
 - Arithmetik in **58**
- $\mathbb{Z}_{(\cdot)}^*$ **44**
- Zachos, S. 228, 318, 344, 345, 457
- Zahl
 - Blum- **427**
 - Carmichael- **360**, 361, 362, 364, 366, 367, 396
 - chromatische, *siehe auch* $\chi(\cdot)$ **108**
 - domatische, *siehe auch* $\delta(\cdot)$ **110**, 139, 196
 - Fermat-, *siehe auch* $F_{(\cdot)}$ **385**, 398
 - Fibonacci-, *siehe auch* $f_{(\cdot)}$ **15**

- ganze, *siehe auch* \mathbb{Z} 13
 - Gödel- **32**
 - natürliche, *siehe auch* \mathbb{N} **13**
 - Binärdarstellung einer, *siehe auch* bin(\cdot) 20
 - Binärentwicklung einer 309
 - Prim- **44**, **356**
 - Primitivwurzel einer **8**, 405
 - rationale, *siehe auch* \mathbb{Q} 56
 - reelle, *siehe auch* \mathbb{R} 56
 - RSA-*d* 385
 - Unabhängigkeits-, *siehe auch* $\alpha(\cdot)$ **228**
- Zahlentheorie *siehe* Theorie, Zahlen-Zähl-Hierarchie 345
 Zählklasse 326–330
 Zappa, F. 185
 Zbinden, H. 10
 Zeitfunktion *siehe* Funktion, Zeit-Zeithierarchie 6, **80**, 136
 Zeithierarchiesatz
 siehe Satz, Zeithierarchie-zeitkonstruierbar *siehe*
 Funktion, zeitkonstruierbare
 Zensusfunktion
 siehe Funktion, Zensus-Zero-Knowledge **430–439**, 458
 - *almost-perfect* 458
 - *computational* 458
 - *honest-verifier* 436, 458
 - *perfect* 436, 458
 - *statistical* 458
 Zero-Knowledge-Eigenschaft 434, **436**, 452
 Zero-Knowledge-Protokoll 3, 9, 48, 294, 403, **430–439**, 458
 - *siehe* Fiat–Shamir-Protokoll
 - *siehe* Goldreich–Micali–Wigderson-Protokoll
 Zertifikat *siehe* Zeuge
 Zeuge **215**, 242
 - Anzahl von —n *siehe* akzeptierende Berechnung, Anzahl von —en; *siehe* acc(\cdot); - *siehe auch* #P
 - Menge der —n *siehe* Zeugmenge
 - Zeugenmenge, *siehe auch* Wit(\cdot) **216**
 - Zimand, M. 282, 286, 292
 - Zippel, R. 454, 459
 - ZPP 8, 288, 293, 302, **307**, 326, 339, 343, 347
 - ZPP^{AM ∩ coAM} 400
 - ZPP^{NP} 288, 346
 - Zufallsvariable **54**
 - ZUP 345
 - Zweiter Weltkrieg 2, 189, 191