Komplexitätstheorie

Hans U. Simon

24. Oktober 2019

Inhaltsverzeichnis

1 Einleitung

Die Komplexitätstheorie beschäftigt sich mit der Frage, welche Berechnungsprobleme sich automatisch mit einem realistischen Aufwand an Zeit (= Rechenzeit) bzw. Platz (= Speicherplatz) lösen lassen. Dabei wird der Verbrauch von Zeit oder Platz stets als Funktion in der Eingabelänge n gemessen. In der Regel wird die Anzahl von Rechenschritten (bzw. die Anzahl von belegten Speicherzellen) bei Eingaben der Länge n nicht exakt bestimmt, sondern nur größenordnungsmäßig. Hierbei kommt die Landau'sche O-Notation zum Einsatz, welche die asymptotische Wachstumsordnung einer Funktion nach oben abschätzt. Zum Beispiel ist aus der Vorlesung „Datenstrukturen“ bekannt, dass sich n Elemente einer total geordneten Menge in $O(n \log n)$ Schritten sortieren lassen. Aber selbst bei einer solchen groben Bestimmung der benötigten Rechenzeit kommen wir nicht umhin, einige technische Details exakt festzulegen:

1. Wie messen wir die „Länge“ $|I|$ einer Eingabeinstanz I?
2. Welches *Maschinenmodell* legen wir zugrunde und was kann diese Maschine in *einem* Rechenschritt leisten?

Zur Beantwortung der ersten Frage werden wir die „Kodierungslänge“ einer Eingabeinstanz heranziehen: wieviele Bits hat eine „natürliche Kodierung“ von I?[1] Als Maschinenmodell verwenden wir die nach ihrem Erfinder Alan Turing benannte „Turing-Maschine (TM)“, die den meisten Hörerinnen und Hörern aus der Vorlesung „Theoretische Informatik“ bekannt sein dürfte. Dabei unterscheiden wir „Deterministische Turing-Maschinen (DTMs)“ und „Nichtdeterministische Turing Maschinen (NTMs)“. Eine TM besitzt eines oder mehrere Arbeitsbänder, welche jeweils in Zellen unterteilt sind. Der Bedarf an Speicherplatz ergibt sich aus der Anzahl der während einer Rechnung besuchten Zellen.[2]

Im Rahmen der Komplexitätstheorie werden Rechenprobleme, die einen ähnlichen Verbrauch an Zeit bzw. Platz erfordern, zu sogenannten Komplexitätsklassen zusammengefasst. Einige der wichtigsten Klassen sind die folgenden:

Deterministisch Logarithmischer Platz: Die DTM besucht bei Eingaben der Länge n lediglich $O(\log n)$ Zellen ihrer Arbeitsbänder. Probleme, die sich auf diese Weise lösen lassen, landen in der Klasse $\mathcal{L}$ (= Logspace).

Nichtdeterministisch Logarithmischer Platz Die entsprechende Klasse für NTMs an Stelle von DTMs, notiert als $\mathcal{N}L$.

Deterministisch Polynomielle Rechenzeit: Die DTM rechnet bei Eingaben der Länge n lediglich $p(n)$ Schritte für ein geeignet gewähltes Polynom $p(n)$. Probleme, die sich auf diese Weise lösen lassen, landen in der Klasse P (= Polynomialzeit).

Nichtdeterministisch Polynomielle Rechenzeit: Die entsprechende Klasse für NTMs an Stelle von DTMs, notiert als NP.

Polynomieller Platz: Die DTM besucht bei Eingaben der Länge n lediglich $p(n)$ Zellen ihrer Arbeitsbänder für ein geeignet gewähltes Polynom p. Probleme, die sich auf diese Weise lösen lassen, landen in der Klasse *PSpace*.

In Abschnitt 14.2 werden wir sehen, dass jede NTM mit einem Platzbedarf von $O(S(n))$ — wobei $S(n) \geq \log n$ — von einer DTM mit Platzbedarf $O(S(n)^2)$ simuliert werden kann (Satz von Savich). Hieraus ergibt sich, dass die Klasse *PSpace* alternativ betrachtet werden kann als die Klasse von Problemen, die sich mit *nichtdeterministisch* polynomiellem Platz lösen lassen.

In Abschnitt 14.4 werden wir sehen, dass die oben aufgeführten Komplexitätsklassen eine durch Mengeninklusion geordnete Kette bilden, nämlich die folgende *Platz-Zeit-Hierarchie*:

$$\mathcal{L} \subseteq \mathcal{N}L \subseteq P \subseteq NP \subseteq PSpace \tag{1}$$

[1] Auf die naheliegende nächste Frage, was unter einer „natürlichen Kodierung“ zu verstehen ist, gehen wir im Abschnitt 2 ein.

[2] Bei sublinearem Verbrauch an Platz hat die TM ein „Read-only“ Eingabeband und nur die Zellen auf den restlichen Arbeitsbändern werden gezählt.

Wie wir in Abschnitt 15 sehen werden, gilt $\mathcal{NL} \subset PSpace$. Daher muss in (1) an mindestens einer Stelle eine echte Inklusion vorliegen. Es wird vermutet, dass *jede* der Inklusionen in (1) echt ist, aber von keiner einzigen konnte dies bisher gezeigt werden!

Die Klasse P gilt als die Klasse der Probleme, welche sich mit einem tolerierbaren Aufwand an Rechenzeit lösen lassen. Die Klasse NP ist unglaublich reich an Problemen (und zwar aus allen möglichen Wissenschafts- und Anwendungsbereichen), die man gerne lösen würde. Es überrascht daher nicht, dass der Frage „$P \neq NP$?“ eine sowohl theoretisch wie praktisch große Bedeutung zugeschrieben wird (P–NP Problem). Das Clay Mathematics Institute hat zur Jahrtausendwende eine Liste von 7 „Millenium–Problemen“ veröffentlicht:

- Birch and Swinnerton-Dyer Conjecture
- Hodge Conjecture
- Navier-Stokes Equations
- P versus NP
- Poincaré Conjecture (im Jahr 2010 von Grigoriy Perelman aus St. Petersburg gelöst)
- Riemann Hypothesis
- Yang-Mills Theory

Wie zu sehen ist, befindet sich das P–NP Problem auf Platz 4 dieser Liste und kann als eines der wichtigsten und spannensten Probleme der Mathematik bzw. der Theoretischen Informatik betrachtet werden.
Das Skriptum zur Komplexitätstheorie ist aufgebaut wie folgt:

- In Kapitel 2 stellen wir ein paar grundlegende Begriffe und Fakten zusammen. U.a. erinnern wir an das Konzept der *NP-vollständigen Probleme*. Das sind schwerste Probleme in der Klasse NP mit der Eigenschaft, dass die Lösbarkeit *eines* dieser Probleme in deterministisch Polynomialzeit die Gleichung $P = NP$ implizieren würde.[3] Einige der Grundlagen in Kapitel 2 sollten aus der Veranstaltung „Theoretische Informatik“ bereits bekannt sein.
- In den Kapiteln 3 und 4 zeigen wir, dass in der sequentiellen Komplexitätstheorie[4] in der Regel die Gleichung „Entscheiden = Konstruieren“ gilt. Genauer: wir können ohne wesentlichen Effizienzverlust einen Algorithmus, der lediglich *entscheidet*, ob zu einem Problem eine akzeptable Lösung existiert, transformieren in einen Algorithmus, der eine akzeptable Lösung (so sie existiert) *konstruiert*. Dieser Sachverhalt kann dann als Rechtfertigung dienen, sich beim Studium der Komplexitätsklassen auf Entscheidungsprobleme (mit den möglichen Antworten „Ja“ und „Nein“) zu beschränken.

[3] Vermutet wird freilich, dass keines der NP-vollständigen Probleme in deterministisch Polynomialzeit lösbar ist.

[4] im Unterschied zur Komplexitätstheorie für Parallelrechner, um welche wir uns in dieser Vorlesung nicht kümmern werden

- In Kapitel 5 machen wir an Beispielen deutlich, dass sich zwischen P und NP mitunter nur ein schmaler Grat befindet: eine leichte Variation eines effizient lösbaren Problems kann zu einem schwersten Problem in der Klasse NP führen, und umgekehrt. Ein Lernziel der Vorlesung wird darin bestehen, effizient lösbare Problemvarianten von inhärent schweren Varianten unterscheiden zu können.

- In Kapitel 6 betrachten wir exemplarisch das Graphenfärbungsproblem und zeigen, dass selbst extrem eingeschränkte Varianten dieses Problems immer noch NP-vollständig sind.

- Mit n Bits lassen sich Zahlen bis zur Größe $2^n - 1$ darstellen. Probleme, deren Eingabeinstanzen Zahlparameter enthalten, sind manchmal nur darum inhärent schwer, weil die beteiligten Zahlen „riesengroß“ sind. Dies führt uns in Kapitel 7 auf das Thema der *pseudopolynomiellen Algorithmen* (sofern das zugrunde liegende Problem im Falle „kleiner“ Zahlparameter effizient lösbar wird) bzw. das Thema der *starken NP-Vollständigkeit* (sofern das Problem selbst bei „kleinen“ Zahlparametern inhärent schwer bleibt).

- Kapitel 8 führt uns auf das Thema der polynomiellen Approximationsalgorithmen (sofern das zugrunde liegende Problem effizient lösbar wird, wenn wir mit „guten“ anstelle von optimalen Lösungen zufrieden sind) bzw. das Thema der NP-harten Approximation (wenn selbst das Auffinden „guter“ Lösungen inhärent schwer ist). Zudem sprechen wir kurz das Thema der Komplexitätsklassen für Optimierungsprobleme an.

- Kapitel 9 widmet sich dem von Downey und Fellows entwickelten Ansatz der „parameterisierten Komplexität“. Dahinter steht der Versuch einen (oder mehrere) Eingabeparameter dingfest zu machen, der gewissermaßen für die inhärente Härte eines Problems verantwortlich ist. Ein *Festparameter-behandelbares Problem* ist eines, das bei konstanten Werten des Festparameters k für ein geeignetes Polynom p in $p(n)$ Rechenschritten lösbar ist, wobei der Grad des Polynoms p *nicht* von k abhängt. Wir diskutieren „Vertex Cover“ als ein (Muster-)Beispiel für ein Festparameter-behandelbares Problem. Es gibt aber (leider) eine Vielzahl von Problemen, die (unter der $P \neq NP$-Voraussetzung) beweisbar nicht Festparameter-behandelbar sind.

- Die Kapitel 6 bis 9 können unter dem Motto „Umgang mit inhärent schweren Problemen“ gesehen werden. Dahinter steht die Frage, ob das Problem effizient lösbar wird, wenn wir es geeignet einschränken, den Anspruch an Optimalität aufgeben oder bei einem „Festparameter“ nur kleine Werte zulassen. Kapitel 10 zieht eine Zwischenbilanz aus den voran gegangenen Kapiteln und listet ein paar weitere Möglichkeiten zum Umgang mit inhärent schweren Problemen auf. Eines der Lernziele der Vorlesung besteht darin zu erkennen, ob eine geeignete Variante eines inhärent schweren Problems effizient lösbar wird.

- In Kapitel 11 nehmen wir (unter der $P \neq NP$-Vorausetzung) die Struktur von NP unter die Lupe. Insbesondere weisen wir nach, dass es Probleme in NP gibt, die we-

der zu P gehören noch NP-vollständig sind. Der Nachweis wird mit der Technik der Diagonalisierung geführt.

- In Kapitel 12 beschäftigen wir uns mit der Theorie der „dünnen“ und „dichten“ Sprachen, die 1977 von Berman und Hartmanis ins Leben gerufen wurde. Insbesondere diskutieren wir die (bis heute offene) „Isomorphie-Vermutung“ und weisen (unter der $P \neq NP$-Voraussetzung) nach, dass es keine dünne NP-vollständige Sprache geben kann (Satz von Mahaney).

- Kapitel 13 widmet sich den Untersuchungen von Baker, Gill und Solovay zur „Relativierung“ der P-versus-NP Frage aus dem Jahre 1975. Aus diesen Untersuchungen wird hervorgehen, dass bestimmte Schlüsseltechniken, wie zum Beispiel „Simulation“ oder „Diagonalisierung“, weder zum Nachweis von $P = NP$ noch zum Nachweis von $P \neq NP$ erfolgreich eingesetzt werden können. Dieses sowie die beiden voran gegangenen Kapitel können als Versuch gesehen werden, Licht in das Dunkel des P-versus-NP Problems zu werfen (bzw. als Versuch zu verstehen, warum sich die Vermutung $P \neq NP$ bisher hat weder verifizieren noch falsifizieren lassen).

- Kapitel 14 beschäftigt sich mit Beziehungen zwischen Komplexitätsklassen und dahinter stehenden Fragen wie „Platz versus Zeit“ oder „Determinismus versus Nichtdeterminismus“. Insbesondere wird der Nachweis für die Platz-Zeit-Hierarchie (1) geführt. Außerdem sprechen wir kurz den Satz von Immerman und Szelepcsényi aus dem Jahre 1987 an, welcher besagt, dass Platzkomplexitätsklassen (ab „logspace“ aufwärts) unter Komplement abgeschlossen sind.

- Kapitel 15 behandelt Hierarchiesätze, welche in salopper Formulierung besagen, dass eine signifikante Erhöhung der Ressourcen „Zeit“ oder „Platz“ zu einer echten Vergrößerung der im Rahmen dieser Ressourcen lösbaren Probleme führt. Aus dem Platzhierachiesatz ergibt sich insbesondere die weiter oben bereits angesprochene echte Inklusion $\mathcal{NL} \subset PSpace$. Die Hierarchiesätze werden mit Hilfe von Diagonalisierung bewiesen.

- Kapitel 16 nimmt die Klasse $\mathcal{NL}$ unter die Lupe. In Analogie zur NP-Vollständigkeit (unter polynomiellen Reduktionen) lässt sich der Begriff der $\mathcal{NL}$-Vollständigkeit (unter logspace-Reduktionen) definieren. „Digraph-Reachability“ erweist sich zum Beispiel als $\mathcal{NL}$-vollständig. Wir geben zudem eine Charakterisierung von $\mathcal{NL}$ mit sogenannten „Read-once Zertifikaten“ an (in Analogie zu den kurzen und effizient verifizierbaren Zertifikaten, welche die Klasse NP charakterisieren).

- Kapitel 17 widmet sich der Klasse $PSpace$. Das Problem „True Quantified Boolean Formulae (TQBF)“ (ein prädikatenlogischer Verwandter des aus der NP-Vollständigkeitstheorie bekannten Problemes „Satisfiability“) erweist sich als $PSpace$-vollständig. Hieraus ergibt sich eine prädikatenlogische Charaktersierung von $PSpace$. Das Kapitel schließt mit einer Sammlung weiterer $PSpace$-vollständiger Probleme und mit einer Liste offener Probleme.

- In Kapitel 18 beschäftigen wir uns mit der polynomiellen Hierarchie, die Meyer und Stockmeyer 1972 definiert haben. Es handelt sich um eine (vermutlich) unendliche Hierarchie von Klassen, die zwischen P und *PSpace* zu liegen kommt. Sie wird mit Hilfe von sogenannten Orakel-Turing-Maschinen (OTMs) definiert.

- Kapitel 19 liefert eine beweisbar äquivalente Definition der polynomiellen Hierarchie, die die verschiedenen Stufen der Hierarchie mit alternierenden Quantorenketten ansteuert (Satz von Celia Wrathall). Dies führt zu einer prädikatenlogischen Charakterisierung der Sprachen, die auf den diversen Stufen der Hierarchie zu liegen kommen, und zu einer Reihe von Anwendungen. Insbesondere erhalten wir auf jeder Stufe ein vollständiges Problem über eine geeignete prädikatenlogische Erweiterung von „Satisfiability“.

- Die Turing-Maschine ist ein uniformes Maschinenmodell in dem Sinne, dass wir *das selbe* TM-Programm für *alle* Eingabelängen verwenden. Dem gegenüber stehen die nicht-uniformen Modelle wie sie in der Regel für Hardware verwendet werden. Kapitel 20 diskutiert das Verhältnis von uniformen und nicht-uniformen Rechenmodellen. Insbesondere werden Techniken vorgestellt, Software in Hardware zu transformieren, und umgekehrt. Wir stellen die Klasse $P/poly$ vor — die nicht-uniforme Schwester von P, und verwenden sogenannte *spärliche* und *kospärliche* Sprachen, um zu Charakterisierungen von $P/poly$ und von der P-versus-NP Frage zu gelangen. Schließlich präsentieren wir den Satz von Karp und Lipton aus dem Jahre 1980, welcher besagt, dass $NP \subseteq P/poly$ implizieren würde, dass die polynomielle Hierarchie auf der 2. Stufe kollabiert. Dieser Satz kann als Hinweis verstanden werden, dass superpolynomielle untere Schranken zur Schaltkreiskomplexität (zumindest theoretisch) zu einem Nachweis von $P \neq NP$ führen könnten.

- Das abschließende Kapitel 21 kümmert sich um *randomisierte Algorithmen*. Das dazu passende Maschinenmodell bilden die *probabilistischen Turing-Maschinen (PTMs)*, die den nächsten Rechenschritt von dem Ergebnis eines Münzwurfs abhängig machen können. Wir schauen uns im Detail die probabilistischen Komplexitätsklassen *BPP*, *RP* und *ZPP* an und analysieren, in welchem Verhältnis sie untereinander und in welchem Verhältnis sie zu den nicht-probabilistischen Komplexitätsklassen stehen. Das Kapitel schließt mit der Gesamtlandschaft aller im Skriptum betrachteten Komplexitätsklassen.

Analysetechniken und Lernziele: Im Laufe der Vorlesung machen die Hörerinnen und Hörer Bekanntschaft mit veschiedenen Schlüsseltechniken wie zum Beispiel *Simulation*, *Diagonalisierung* sowie verschiedene Arten der *Reduktion* (polynomielle Reduktion, logspace-Reduktion, Turing-Reduktion, Levin-Reduktion). Simulation hilft zu verstehen, wie gut sich eine Maschine von einem Typ A simulieren lässt durch eine Maschine vom Typ B. Mit Hilfe von Simulation lässt sich beispielsweise verstehen, dass die Komplexitätstheorie robust ist gegenüber kleineren Variationen des Standardmodelles einer TM. Mit Hilfe von Diagonalisierung lassen sich die Hierarchiesätze herleiten oder auch die Existenz einer Sprache in $NP \setminus P$,

die nicht NP-vollständig ist. Mit Hilfe von Problemreduktionen lassen sich schwerste Probleme in einer Komplexitätsklasse ausfindig machen (wie zum Beispiel die $\mathcal{C}$-vollständigen Probleme für die Komplexitätsklassen $\mathcal{C} = \mathcal{L}, NP, PSpace$). Der professionelle Umgang mit den genannten Techniken ist sicherlich ein zentrales Anliegen der Vorlesung. Weiterhin soll die Fähigkeit erworben werden, ein konkretes Problem in die Landschaft der Komplexitätsklassen einzuordnen und auf diese Weise seine inhärente Komplexität zu bestimmen. Dabei spielt freilich die Unterscheidung von effizient lösbaren Problemen und inhärent schweren Problemen eine für die Praxis besonders wichtige Rolle. Techniken zum Design von effizienten Algorithmen werden in den Vorlesungen „Datenstrukturen“ und „Effiziente Algorithmen“ erworben. In der Vorlesung „Komplexitätstheorie“ wird das komplettiert durch den Erwerb von Techniken zum Umgang mit inhärent schweren Problemen.

Wir wünschen den Teilnehmerinnen und Teilnehmern an der Vorlesungsreihe viel Spaß und (Lern-)Erfolg!

Bochum, der 07.09.2017 Hans Ulrich Simon

2 Grundlegende Begriffe und Resultate

Bekanntschaft mit der Turing-Maschine aus der Vorlesung „Theoretische Informatik“ wird weitgehend vorausgesetzt, obschon die folgenden beiden Abschnitte ein paar zentrale Definitionen und Resultate rekapitulieren. Die meisten Beweise in diesem Kapitel werden wir nur grob skizzieren, weil der eigentliche neue Stoff der Vorlesungsreihe erst mit dem folgenden Kapitel beginnen wird. Das laufende Kapitel dient gewissermaßen als Erinnerungsstütze und kleine Aufwärmübung.

Ein paar Bemerkungen vorab. Wenn in diesem Skriptum von „Polynomen“ die Rede ist, dann sind in der Regel Polynome mit nicht-negativen Koeffizienten gemeint, da es meist nur darum geht die Rechenzeit $T(n)$ eines Algorithmus nach oben durch ein Polynom $p(n)$ in der Eingabelänge n zu begrenzen. Wenn dabei nur wichtig ist, dass ein solches Polynom existiert, dann notieren wir das in der Form $T(n) \leq \mathrm{pol}(n)$. Dabei steht $\mathrm{pol}(n)$ also immer für ein geeignet gewähltes Polynom, ohne dass dieses konkretisiert wird. Zwei Funktionen $f(n), g(n)$ heißen *polynomiell verknüpft*, wenn $f(n) \leq \mathrm{pol}(g(n))$ und $g(n) \leq \mathrm{pol}(f(n))$, d.h. also, wenn Polynome p, q existieren mit $f(n) \leq p(g(n))$ und $g(n) \leq q(f(n))$. Insbesondere folgt dann: wenn eine der Funktionen $f(n), g(n)$ polynomiell beschränkt ist, dann trifft das auch auf die andere Funktion zu.

2.1 Die Deterministische Turing-Maschine (DTM)

Die Komponenten des 1-Band Standardmodells einer deterministischen Turing-Maschine (DTM) M sind wie folgt:

- Q, die endliche Zustandsmenge,

- Σ, das endliche Eingabealphabet,
- $\Gamma \supseteq \Sigma$, das endliche Bandalphabet,
- $B \in \Gamma \setminus \Sigma$, der Blank (Leerzeichen),
- $q_0 \in Q$, der Startzustand,
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{R, L, N\}$, die Überführungsfunktion,
- $F \subseteq Q$, die Menge der akzeptierenden Zustände.

Dies kann in der Gleichung $M = (Q, \Sigma, \Gamma, B, q_0, \delta, F)$ zusammengefasst werden. M verfügt über einen (Lese/Schreib-)Kopf und über ein zweiseitig unendliches Band, das in Zellen mit den Nummern $\ldots, -2, -1, 0, 1, 2, \ldots$ unterteilt ist. In jeder Zelle steht ein Symbol aus Γ. Zu Anfang einer Rechnung mit dem Eingabewort $w = a_1 \ldots a_n \in \Sigma^n$ gilt folgendes:

- Das Eingabewort w steht in den Zellen $1, \ldots, n$. Alle anderen Zellen enthalten das Leerzeichen B.
- Der Kopf von M ist auf Zelle 1 positioniert.
- M befindet sich im Startzustand q_0.

Ein Rechenschritt von M wird formal beschrieben durch die Überführungsfunktion $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$. Die Interpretation von $\delta(q, X) = (q', X', d)$ ist die folgende: Wenn M im Zustand q Symbol X liest, so geht sie in Zustand q' über, ersetzt X durch X' und bewegt den Kopf eine Zelle in Richtung links (bzw. rechts), falls $d = L$ (bzw. $d = R$). $d = N$ steht für „nicht bewegen“.

Die Überführungsfunktion δ ist gewissermaßen das „Programm“ von M, welches die Rechnung auf der Eingabe w steuert. Falls M im Zustand q das Symbol X liest und $\delta(q, X) = (q, X, N)$, dann sagen wir, dass M *stoppt* (Ende der Rechnung). Falls dabei $q \in F$, dann handelt es sich um eine *akzeptierende Rechnung*. Andernfalls heißt die Rechnung *verwerfend*. Die Menge $L(M)$ der von M akzeptierten Wörter heißt *die Sprache zu M*. M heißt *Akzeptor* der Sprache $L \subseteq \Sigma^*$, falls $L = L(M)$. Wir sagen dann auch *M erkennt die Sprache L*.

Um die Rechnung einer DTM formal zu verfolgen, macht man Schritt für Schritt eine Momentaufnahme genannt „Konfiguration“. Eine *Konfiguration* von M ist ein String $\alpha q \beta$ mit $\alpha, \beta \in \Gamma^*,\ q \in Q$. Sie bedeutet, dass auf dem Band die Inschrift $\alpha\beta$ eingerahmt von lauter Blanks steht, die Maschine im Zustand q und der Kopf auf dem ersten Buchstaben von β positioniert ist. Eine Rechnung von M kann dann auf natürliche Weise als eine Folge von Konfigurationen beschrieben werden.

2.2 Die Nichtdeterministische Turing-Maschine (NTM)

In der Vorlesung „Theoretische Informatik“ wurde das Standardmodell einer NTM M mit ähnlichen Komponenten eingeführt wie das Standardmodell der DTM, bis auf die folgenden Abänderungen:

- Die NTM hat pro Schritt mehrere Wahlmöglichkeiten für den nächsten Rechenschritt, sagen wir oBdA zwei Wahlmöglichkeiten. Formalisieren ließe sich das durch zwei Überführungsfunktionen δ_0, δ_1. In jedem Schritt kann M auswählen, ob sie die Funktion δ_0 oder δ_1 zugrunde legt.

- Da es i.A. nun viele mögliche Rechnungen gibt (möglicherweise einige akzeptierend, andere verwerfend) müssen wir festlegen, wann ein Eingabewort w als „akzeptiert" gilt: wir sagen M *akzeptiert* w, falls mindestens eine akzeptierende Rechnung auf Eingabe w existiert. $L(M)$ bezeichnet wieder die Menge der von M akzeptierten Eingabewörter.

Sprechweise: Das Auswählen einer der beiden Funktionen δ_0, δ_1 wird auch als „Raten" bezeichnet. Wenn die Rateentscheidungen schließlich zum Akzeptieren von w führen, spricht man auch davon, dass die NTM „richtig geraten" hat.

Aus der Vorlesung „Theoretische Informatik" wissen wir, dass jede NTM M in eine äquivalente NTM M' transformiert werden kann, welche nach dem „Rate-Verifikationsprinzip" vorgeht:

Nichtdeterministische Ratephase: M' schreibt in Zelle 0 das Trennsymbol #. Danach rät sie einen String $y \in \{0,1\}^*$ bitweise und notiert ihn auf den Zellen $-1, \ldots, -|y|$. (Auch die Länge $|y|$ des Strings y wird dabei geraten.) Zum Schluss positioniert sie ihren Kopf auf Zelle 0.

Deterministische Verifikationsphase: In einer zweiten Phase rechnet M' deterministisch auf der erweiterten Eingabe $y\#x$.

Bemerkung 2.1 *Wenn die NTM M in Polynomialzeit arbeitet, dann hat der von M' geratene String y ebenfalls eine polynomiell in $n = |w|$ beschränkte Länge.*

Sprechweise: Wenn die erweiterte Eingabe $y\#x$ in der Verifikationsphase akzeptiert wird, heißt y auch ein „Zertifikat" für $x \in L(M)$.

2.3 Komplexitätsklassen

Wir sagen die DTM M hat *Platzschranke* $S(n)$ (bzw. *Zeitschranke* $T(n)$, falls sie, angesetzt auf Eingabewörter der Länge n, maximal $O(S(n))$ Zellen besucht (bzw. maximal $O(T(n))$ Schritte rechnet). Die Platzschranke (bzw. Zeitschranke) nennen wir *exakt*, wenn M auf Eingaben der Länge n exakt $S(n)$ Zellen besucht (bzw. exakt $T(n)$ Schritte rechnet). $DSpace(S(n)$ bezeichnet die Klasse aller Sprachen, die von einer $S(n)$-platzbeschränkten DTM erkannt werden können. $DTime(T(n))$ bezeichnet die Klasse aller Sprachen, die von einer $T(n)$-zeitbeschränkten DTM erkannt werden können. $NSpace(S(n))$ und $NTime(T(n))$

sind die analog definierten nichtdeterministischen Komplexitätsklassen. Aus diesen „generischen“ Komplexitätsklassen ergeben sich die Klassen der Platz-Zeit Hierarchie wie folgt:

$$\begin{aligned}
\mathcal{L} &= DSpace(\log n) \\
\mathcal{NL} &= NSpace(\log n) \\
P &= \bigcup_{k\geq 1} DTime(n^k) \\
NP &= \bigcup_{k\geq 1} NTime(n^k) \\
PSpace &= \bigcup_{k\geq 1} DSpace(n^k) = \bigcup_{k\geq 1} NSpace(n^k)
\end{aligned}$$

Die letzte Gleichheit gilt nach dem Satz von Savich (s. Einleitung). Es sei auch daran erinnert, dass wir bei sublinearen Platzschranken voraussetzen, dass die TM über ein zusätzliches Read-only Eingabeband verfügt und beim Platzbedarf nur die besuchten Zellen des Arbeitsbandes gezählt werden.

2.4 Varianten des Standardmodells

Es gibt zahlreiche Variationen des Standardmodells. Zum Beispiel:

- Das Band ist *einseitig* unendlich mit Zellnummern $0, 1, 2, \ldots$. (In diesem Fall steht auf Zelle 0 eine Endmarkierung, damit M nicht versehentlich vom Band fällt.)
- Der Kopf muss sich in jedem Schritt (nach links oder rechts) bewegen.
- Es gibt nur einen akzeptierenden Zustand.

Es ist einfach, zwischen dem Standardmodell und den genannten Variationen wechselseitige effiziente Simulationen anzugeben. Das heißt (glücklicherweise!), dass die Ergebnisse der Komplexitätstheorie nicht von solchen nickligen kleinen technischen Voraussetzungen abhängen. Man sagt auch: die Theorie ist *robust*.

Eine interessantere Variation des Standardmodells besteht darin, mehrere Arbeitsbänder zuzulassen. Eine k-Band DTM ist ähnlich definiert wie eine 1-Band DTM, außer dass sie über k Köpfe, einen pro Band, verfügt, die sich unabhängig von einander bewegen dürfen. Die Überführungsfunktion δ hat dann das Format

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{R, L, N\}^k$$

mit der offensichtlichen Interpretation. Eine analoge Bemerkung gilt für k-Band NTMs. Die Komplexitätstheorie ist auch gegenüber solchen Erweiterungen des Standardmodells robust, wie die folgenden Bandreduktionssätze belegen:

Satz 2.2 *Eine k-Band DTM (bzw. k-Band NTM) M mit Platzschranke $S(n)$ und Zeitschranke $T(n)$ kann simuliert werden von einer 1-Band DTM (bzw. einer 1-Band NTM) M' mit der selben Platzschranke $S(n)$ und der Zeitschranke $T(n)S(n) \leq T(n)^2$.*

Der Beweis sollte aus der Vorlesung „Theoretische Informatik“ bekannt sein. Die wesentliche Idee besteht darin, bei M' das Bandalphabet Γ^k zugrunde zu legen, so dass die k Bandinschriften von M auf dem *einem* Band von M' untergebracht werden können. Wenn M das Symbol $(X_1, \ldots, X_k)$ in einer Zelle abspeichert, tun wir anschaulich so, als hätte die Zelle k Spuren, so dass X_i sich in Spur i befindet. M' kann dann *einen* Schritt von M in $O(S(n))$ Rechenschritten simulieren. Zum Beispiel könnten Bewegungen von Köpfen von M in Richtung links (bzw. rechts) von M' dadurch simuliert werden, dass die Inschriften der entsprechenden Spuren in Richtung rechts (bzw. Richtung links) verschoben werden. Wir erhalten für M' demnach die Zeitschranke $T(n)S(n)$. Die Ungleichung $T(n)S(n) \leq T(n)^2$ gilt, da offensichtlich $S(n) \leq T(n)$: bei $O(T(n))$ Rechenschritten können natürlich auch nur $O(S(n))$ Zellen besucht werden. Bei Interesse an weiteren Details sei auf die Vorlesung „Theoretische Informatik“ verwiesen.

2-Band TMs können führen zu noch effizienteren Simulationen:

Satz 2.3 *1. Eine k-Band DTM M mit Platzschranke $S(n)$ und Zeitschranke $T(n)$ kann simuliert werden von einer 2-Band DTM M' mit der selben Platzschranke $S(n)$ und der Zeitschranke $T(n) \log S(n) \leq T(n) \log T(n)$.*

2. Eine k-Band NTM M mit Zeitschranke $T(n)$ kann simuliert werden von einer 2-Band NTM M' der selben Zeitschranke $T(n)$.

Wir lassen den Beweis von diesem Satz aus und skizzieren nur die wesentlichen Ideen, die zum Beweis von Satz 2.2 noch hinzu kommen:

1. Die 2-Band DTM M' benutzt Band 1 als „Schmierblatt“ und Band 2 zur Abspeicherung der k Bandinschriften. Diese werden aber so „aufgelockert“ in Blöcken verschiedener Längen abgespeichert, dass eine Bandinschrift nicht immer komplett verschoben werden muss. Die Speicherorganisation ist so listig gewählt, dass lange Blöcke nur selten in das Verschieben einbezogen werden. Das Schmierblatt (Band 1) dient im Übrigen dazu, die Blockverschiebungen effizient zu bewerkstelligen.

2. Die 2-Band NTM M' benutzt ein Band, um die komplette Rechnung von M (kodiert als Folge von Konfigurationen) zu raten. Das andere Band wird danach benutzt, um spurenweise (also eine Spur nach der anderen) zu verifizieren, dass die geratene Rechnung auf Band 1 konsistent ist zu dem, was auf Spur i (bzw. auf dem i-ten Band von M, deren Inschrift auf Spur i untergebracht ist) real passiert, wenn gemäß der Überführungsfunktion von M vorgegangen wird.

Bemerkung zur Spezifikation von Komplexitätsklassen: Im Folgenden schreiben wir $\mathrm{DSPACE}_k(S(n))$ bzw. $\mathrm{DTime}_k(T(n))$, wenn die betreffende Platz bzw. Zeitschranke für k-Band DTMs gelten soll. Wenn der Index k fehlt, sind beliebige Mehrband-DTMs zugelassen. Eine entsprechende Bemerkung gilt für nichtdeterministische Komplexitätsklassen. Aus den Bandreduktionstheoremen ergibt sich, dass es bei den Klassen $\mathcal{L}, \mathcal{NL}, P, NP, PSpace$ egal ist, ob wir eine beliebige Mehrband-DTM zulasssen oder uns auf 1-Band-DTMs beschränken.

DTMs bilden ein elegantes Maschinenmodell für theoretische Zwecke. Man könnte aber einwenden, dass es kein sehr praxisnahes Modell ist. Ein praxistauglicheres Modell (das wir aber im Rahmen dieser Vorlesung nicht formal einführen) ist die „Random Access Maschine (RAM)“, deren Programme in etwa dem 3-Adress-Code entsprechen, den Compiler (bei der Tansformation eines Programmes einer höheren Programmiersprache in ein Programm einer maschinennäheren Sprache) erzeugen. Wir merken hier kurz an, dass DTMs und RAMs sich wechselweitig simulieren können, so dass ihre Laufzeiten polynomiell verknüpft sind. (Polynome dritten Grades reichen bereits aus.) Das heißt zum Beispiel, dass die Klasse P übereinstimmt mit der Klasse der Sprachen, die von einer RAM (beim sogenannten logarithmischen Kostenmaß) in Polynomialzeit erkannt werden können. Die Komplexitätstheorie ist also auch weitgehend robust gegenüber dem Austausch der Turing-Maschine durch ein praxisrelevanteres Modell.

Verschärfte Church'sche These: Die Church'sche These besagt, dass eine Funktion genau dann intuitiv berechenbar ist, wenn sie von einer Turing-Maschine berechnet werden kann. Die Turing-Maschine ist in diesem Sinn eine *universelle Maschine*: sie kann alles berechnen, was überhaupt als berechenbar gelten kann. Die Verschärfung dieser These lautet: alle „vernünftigen“ Maschinenmodelle sind in ihrer Rechenzeit mit der Deterministischen Turing-Maschine polynomiell verknüpft.[5]

2.5 Kompression und Beschleunigung

In Verbindung mit den Bandreduktionssätzen habe wir schon einmal den (zugegebenermaßen schmutzigen) Trick verwendet eine TM mit einem Superalphabet der Form Γ^k auszustatten. Der selbe Trick kann verwendet werden, um Platz- oder Zeitschranken um einen konstanten Faktor zu stauchen (außer dass n Rechenschritte stets benötigt werden, um das Eingabewort vollständig zu lesen). Wir geben hier ohne Beweis die daraus resultierenden Sätze an:

Satz 2.4 (Kompressionstheorem) *Es sei M eine DTM (bzw. NTM), die auf Eingaben der Länge n maximal $S(n)$ Zellen ihrer Arbeitsbänder besucht. Weiter sei $\varepsilon > 0$ eine beliebig kleine Konstante. Dann kann M von einer DTM (bzw. NTM) simuliert werden, die lediglich $\varepsilon \cdot S(n)$ Zellen ihrer Arbeitsbänder besucht.*[6]

Satz 2.5 (Beschleunigungstheorem) *Es sei M eine DTM (bzw. NTM), die auf Eingaben der Länge n maximal $T(n)$ Schritte rechnet. Weiter sei $\varepsilon > 0$ eine beliebig kleine Konstante. Dann kann M von einer DTM (bzw. NTM) simuliert werden, die auf Eingaben der Länge n maximal $n + \varepsilon \cdot T(n)$ Schritte rechnet.*

Dies rechtfertigt im Nachhinein die Verwendung von der O-Notation bei der Definition von Platz- und Zeitschranken: die durch die Notation versteckte Konstante spielt wegen der Möglichkeit von Kompression und Beschleunigung ohnehin keine Rolle.

[5] Diese These ist etwas ins Wanken geraten, seitdem es ernsthafte Versuche gibt, Quantenrechner herzustellen. Hierauf können wir aber im Rahmen dieser Vorlesung nicht eingehen.

[6] Falls $\varepsilon S(n) < n$, dann hat M' wieder ein Read-only Eingabeband, dessen Zellen nicht mitgezählt werden.

Phänomene wie Bandreduktion, Kompression und Beschleunigung treten auf, weil wir im Prinzip *jedes* konstant große Informationspaket in die Zustandsmenge Q oder das Bandalphabet Γ hineinkodieren können. Wir werden bei Verwendung solcher Kodierungstricks in Zukunft sagen: diese Information „stecken wir in die endliche Kontrolle“ der Turing-Maschine.

Bemerkung 2.6 *Mit Hilfe von Kompression und Beschleunigung lässt sich für viele Funktionen $T(n)$ zeigen, dass eine TM M mit Zeitschranke $T(n)$ in eine TM M' mit exakter Zeitschranke $T(n)$ transformiert werden kann. Das gilt insbesondere für alle super-linearen, zeitkonstruierbaren Funktionen. Eine analoge Bemerkung gilt für Platzschranken und platzkonstruierbare Funktionen.*[7]

2.6 Natürliche Kodierung

TMs haben als Eingabe stets ein Wort w über ihrem Eingabealphabet. Bei natürlichen Problemen treten in der Regel strukturierte Eingaben auf wie zum Beispiel Vektoren (= 1-dimensionale Arrays), Matrizen (= 2-dimensionale Arrays) oder (gerichtete wie ungerichtete) Graphen. Um ein solches strukturiertes Objekt einer TM zum Fraß vorzuwerfen, müssen wir es als ein Wort über dem Eingabealphabet kodieren. In den folgenden Kapiteln des Skriptums werden wir uns nicht mit Codewörtern herumschlagen, sondern unterstellen stets implizit, dass die Eingabe in einer „natürlichen Kodierung“ vorliegt. In diesem Abschnitt gehen wir kurz darauf ein, was man sich unter „natürlicher Kodierung“ vorstellen kann.

Binärkodierung von Zahlen: Mit Hilfe von Bitstrings der Länge k können Zahlen im Bereich von 0 bis $2^k - 1$ kodiert werden. Zu einer Zahl n bezeichne $\text{bin}(n) \in \{0,1\}^+$ ihre Binärkodierung.

Binärkodierung von Elementen einer endlichen Menge: Die Elemente einer endlichen Menge können durchnummeriert und mit ihren Nummern identifiziert werden. Danach verwenden wir die Binärkodierung von Zahlen (Nummern).

Codewörter für Worttupel: Das Tupel $(u_1, \ldots, u_k)$ könnte unter Verwendung eines neuen Trennzeichens \$ kodiert werden als $u_1\$u_2\$\ldots u_{k-1}\$u_k$. Falls $u_1, \ldots, u_k \in \{0,1\}^*$ und falls wir den Ehrgeiz haben, Codewörter über $\{0,1\}$ herzustellen, können wir 00 als Code für 0, 11 als Code für 1 und 01 als Code für \$ benutzen. Das Tripel $(01, 110, 0)$ hätte dann zum Beispiel das Codewort 0011011111000100.

Codewörter für Wortmatrizen: Das geschieht analog zur Kodierung von Worttupeln. Man kann \$ als Trennzeichen zwischen den Einträgen einer Zeile verwenden und \$\$ als Trennzeichen zwischen verschiedenen Zeilen. Falls die Einträge der Matrix Wörter über $\{0,1\}$ sind, können wir wieder die Substitution $0 \mapsto 00$, $1 \mapsto 11$ und $\$ \mapsto 01$ einsetzen, um einen Binärcode für die Matrix zu erhalten.

[7] Die Definition von zeit- und platzkonstruierbaren Funktionen holen wir später in Abschnitt 14.1 nach.

Codewörter für Vektoren und Matrizen: Nachdem die Zahleneinträge binär kodiert sind, können wir das Kodierschema für Worttupel und Wortmatrizen einsetzen.

Codewörter für endliche Graphen: Eine mögliche Kodierung wäre das Codewort für die Adjazenzmatrix.

Da der Binärcode für Tupel aus Binärwörtern im Skriptum des Öfteren eingesetzt werden wird, führen wir dafür eine Notation ein. Es seien $u_1, \ldots, u_k \in \{0,1\}^*$. Dann bezeichnet $\langle u_1, \ldots, u_k \rangle$ das binäre Codewort für das Tupel $(u_1, \ldots, u_k)$. Zum Beispiel: $\langle 01, 110, 0 \rangle = 0011011111000100$.

Es sind i.A. verschiedene natürliche Kodierungen denkbar. Zum Beispiel könnten wir für Graphen auch ein Codewort bilden, das nicht der Adjazenzmatrix sondern der Adjazenzlistendarstellung entspricht. Glücklicherweise wird diese Mehrdeutigkeit für unsere Betrachtungen keine große Rolle spielen, und zwar aus folgenden Gründen:

- Die Längen verschiedener natürlicher Kodierungen sind i.A. polynomiell verknüpft und lassen sich wechselseitig effizient ineinander umrechnen.
- Die Komplexitätstheorie (mit ihren Klassen $P, NP, \ldots$) nimmt eine so grobe Einteilung der Rechenprobleme vor, dass polynomielle Unterschiede in der Eingabelänge nicht ins Gewicht fallen.

Aus dem gleichen Grund können wir im Übrigen auch mit natürlicheren Maßen für die Länge einer Eingabeinstanz arbeiten. Wenn zum Beispiel die Eingabeinstanz ein Graph $G = (V, E)$ ist, dann können wir als Eingabelänge statt der Länge des Codewortes für G auch die Anzahl $|V|$ seiner Knoten nehmen. Diese beiden Maße sind polynomiell verknüpft.

2.7 Die Universelle Turing-Maschine (UTM)

Ein Computer, so wie Sie in kennen, ist eine „general purpose“ Maschine, d.h., er ist nicht spezialisiert auf die Lösung *eines* Problems, sondern kann prinzipiell beliebige Programme einer Programmiersprache ausführen. Die UTM ist das Pendant zu einem „general purpose computer“ in der Welt der Turing-Maschinen. Die UTM mit festem Grundalphabet $\Sigma \supseteq \{0,1\}$ und fester Anzahl k von Bändern bekommt als Eingabe:

- die Beschreibung $\alpha \in \{0,1\}^*$ einer k-Band DTM $M = M_\alpha$
- das eigentliche Eingabewort $w \in \Sigma^*$

Ihre Aufgabe ist es, die DTM M_α, angesetzt auf das Eingabewort w, zu simulieren.

Zunächst enmal wirft das die Frage auf, wie wir eine k-Band DTM $M = (Q, \Sigma, \Gamma, B, q_0, \delta, F)$ durch einen Binärstring beschreiben. Wir skizzieren das kurz für $k = 1$, wobei das Prinzip auch für beliebige Werte von k deutlich werden wird. Die $r := |Q|$ Zustände von M identifizieren wir mit den Nummern von 1 bis r. Nummer 1 ist dabei für den Startzustand reserviert. Die $s := |\Gamma|$ Bandalphabetssymbole identifizieren wir mit den Nummern von 1 bis s. Die

Richtungsangaben $d_1 = L$, $d_2 = R$ und $d_3 = N$ identifizieren wir (in dieser Reihenfolge) mit den Nummern $1, 2, 3$. Ein Eintrag $\delta(q_i, X_j) = (q_{i'}, X_{j'}, d_k)$ der Funktionstabelle von δ wird kodiert durch $0^i 1 0^j 1 0^{i'} 1 0^{j'} 1 0^k$ (unäre Kodierung mit 1 als Trennsymbol). Das Codewort für die gesamte Funktionstabelle von δ ergibt sich als Konkatenation der Codewörter für die einzelnen Einträge jeweils abgetrennt durch 11. Als Präambel bezeichnen wir den String $0^r 1 b_1 \cdots b_r 1 0^s 11$. Dabei gilt $b_i = 1$, falls $q_i \in F$, und $b_i = 0$ sonst, d.h. die Bitmaske $b_1 \cdots b_r$ gibt uns die Menge F der akzeptierenden Zustände an. Das gesamte Codewort α für M hat dann die Form 111, gefolgt von der Präambel, gefolgt von dem Codewort für δ, gefolgt von 111. Die DTM, die durch das Codewort α beschrieben ist, wird auch mit M_α bezeichnet.

Wir geben ohne Beweis den folgenden Satz an:

Satz 2.7 *Für die Klasse der k-Band DTMs mit einem festen Grundalphabet $\Sigma \supseteq \{0, 1\}$ gibt es eine k-Band UTM U mit folgender Eigenschaft. Ist α das Codewort für eine k-Band DTM, so simuliert U, angesetzt auf $\langle \alpha, w \rangle$, die Rechnung der DTM M_α auf dem Eingabewort w. Zur Simulation* eines *Schrittes von M_α benötigt U lediglich $O(|\alpha|)$ Schritte, sofern $k \geq 2$, und $O(|\alpha|^2)$ Schritte, falls $k = 1$.*

Die Beweisidee ist, dass U ihre k Bänder so benutzt wie M_α, und das Codewort α sowie den aktuellen Zustand von M_α dabei in der „Hosentasche“ mit sich führt. Als „Hosentasche“ dient im Falle $k \geq 2$ eine Extraspur auf Band 1, auf welcher der String α vermerkt ist, sowie eine Extraspur auf Band 2, auf welcher die Nummer des aktuellen Zustands vermerkt ist. Beide Codewörter werden so verschoben, dass sie sich immer in der Nähe der aktuellen Kopfposition befinden. Eine kleine technische Schwierigkeit besteht darin, dass U ein Bandalhabet fester Größe hat, aber DTMs mit im Prinzip beliebig großen Bandalphabeten simulieren können muss. Zu diesem Zweck speichert sie ein Symbol von Γ in einem Block der Größe s ab: Symbol X_j wird dabei mit dem String $0^j 1^{s-j}$ kodiert. Die Polsterung mit $s - j$ Einsen führt zu einer festen Blocklänge s, die sich als nützlich erweist, wenn M ein Symbol X_j mit einem neuen Symbol $X_{j'}$ überschreibt. Wir verzichten auf die Angabe weiterer Details.

2.8 NP-Vollständigkeitstheorie

Ein wichtiges Werkzeug zur Entwicklung der Theorie sind die *polynomiellen Reduktionen*, die (benannt nach ihrem Erfinder Richard Karp) auch *Karp-Reduktionen* genannt werden.

Definition 2.8 *Seien $L_1, L_2 \subseteq \Sigma^*$.*

1. *Wir sagen, L_1 ist* Karp-reduzierbar *auf L_2 (in Zeichen: $L_1 \leq_{pol} L_2$), wenn eine Abbildung $f : \Sigma^* \to \Sigma^*$ mit*

   (i) $\forall w \in \Sigma^* : \; w \in L_1 \Leftrightarrow f(w) \in L_2$

   (ii) f ist von einer polynomiell zeitbeschränkten DTM berechenbar.

   existiert.

2. *Eine Sprache $L_0 \subseteq \Sigma^*$ heißt* NP-vollständig*, falls*

   (i) $L_0 \in NP$

   (ii) $\forall L \in NP: \; L \leq_{pol} L_0$.

   Falls die zweite Bedingung gilt, aber L nicht notwendig zur Klasse NP gehört, dann heißt L NP-hart*.*

Die folgenden Beobachtungen sind leicht zu beweisen.

Bemerkung 2.9 *1. $\leq_{pol}$ ist transitiv. Ketten von Reduktionen ergeben also wieder eine Reduktion.*

2. *Aus $L_1 \leq_{pol} L_2$ und L_1 ist NP-hart folgt, dass auch L_2 NP-hart ist.*

3. *Aus $L \leq_{pol} L_0$ (via Eingabentransformation f) und $L_0 \in P$ folgt $L \in P$. Die Frage, ob $w \in L$, kann nämlich entschieden werden wie folgt:*

   (a) Berechne $f(w)$.

   (b) Entscheide, ob $f(w) \in L_0$.

4. *Wenn ein NP-hartes Problem in (deterministisch) Polynomialzeit gelöst werden kann, dann folgt P=NP.*

Die Liste NP-vollständiger Probleme im Anhang 22 ist durch folgende Evolution entstanden:

1. Das Theorem von Cook — gewissermaßen der "Urknall" der NP-Vollständigkeitstheorie — lieferte mit SATISFIABILITY das erste "natürliche" NP-vollständige Problem.

2. Von SATISFIABILITY ausgehend hat sich mit Reduktionen $\leq_{\text{pol}}$ mit der Zeit eine Art "Stammbaum" NP-vollständiger Probleme gebildet. Ein Teil dieser Stammbaums ist in Abbildung 1 zu sehen. (Die Abkürzungen beziehen sich dabei auf die Problemliste im Anhang.)

Eine lange Liste mit NP-vollständigen Problemen ist in dem Buch *Garey and Johnson, Computers and Intactability, A Guide to the Theory of NP-Completeness, Freeman and Company, 1979*, zu finden. Es enthält zudem eine großartige Beschreibung der Theorie der NP-Vollständigkeit.

SATISFIABILITY (oder kurz SAT) ist ein Problem im Zusammenhang mit Booleschen Formeln. (Vgl. Teil I des Skriptes.) Ein *Literal* ist eine negierte oder nicht negierte Boolesche Variable. Eine *Klausel* ist eine Disjunktion (Logisches Oder) von Literalen, und eine *Formel in konjunktiver Normalform* (kurz: *CNF-Formel*) ist eine Konjunktion (Logisches Und) von Klauseln.

Beispiel 2.10 $(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge x_2$ *ist eine CNF-Formel.*

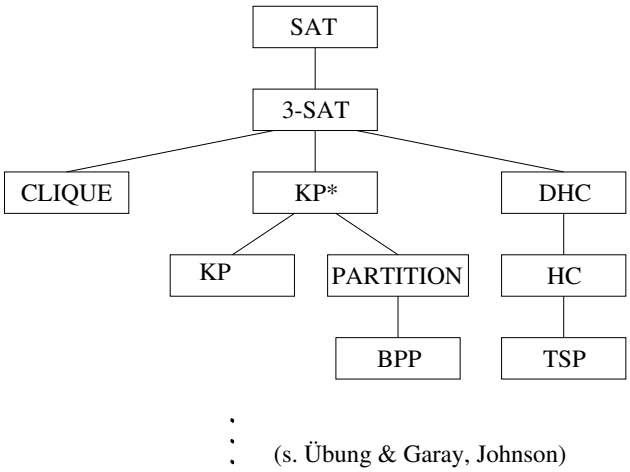


Abbildung 1: Stammbaum NP-vollständiger Probleme.

SAT ist folgendes Problem:

Eingabe eine CNF-Formel F.

Frage Ist die Formel *erfüllbar*, d.h., existiert eine Belegung von $x_1, \ldots, x_n$ mit 0 oder 1, so daß F zu 1 ausgewertet wird?

2.10 Beispiel (fortgesetzt) $x_1 = 1, x_2 = 1, x_3 = 0$ erfüllt obige Formel.

Satz 2.11 (Theorem von Cook)
SAT ist NP-vollständig.

Vorsicht: Der Beweis ist noch einzufügen mit einer Notation, die konsistent zum spaeteren Resultat der *NP*-Vollständigkeit von R_{SAT} ist. Vielleicht sogar schon auf die Lösungstransformationen (welche zur Levin-Reduktion führen) hinweisen.

Satz 2.12 *Alle Probleme aus der Liste im Anhang 22 sind NP-vollständig.*

Für den Beweis des letzten Satzes verweisen wir auf die Vorlesung „Theoretische Informatik“.

3 Entscheiden, Konstruieren und Optimieren

Mit „polynomieller Lösbarkeit“ oder auch „Polynomialzeitlösbarkeit“ eines Berechnungsproblems meinen wir stets die Lösbarkeit des Problems vermöge einer polynomiell zeitbeschränkten deterministischen Turing Maschine (kurz: polynomielle DTM).

Komplexitätsklassen enthalten formale Sprachen über einem Alphabet, die als Entscheidungsprobleme (gehört das Eingabewort zur Sprache?) aufgefasst werden können. Praktische Rechenprobleme hingegen haben oft den Charakter von Konstruktions- oder Optimierungsproblemen. In diesem Abschnitt wollen wir zeigen, dass die Frage der polynomiellen Lösbarkeit aller dieser Problemtypen weitgehend anhand von geeignet definierten formalen Sprachen diskutiert werden kann. Wir erhalten dadurch die Berechtigung, uns in der Komplexitätstheorie auf das Studium von Komplexitätsklassen zurückzuziehen.

Ein wichtiges Werkzeug, um verschiedene Probleme zueinander in Beziehung zu setzen, ist die Problemreduktion. Aus diesem Grund werden wir neben der uns bereits bekannten polynomiellen Reduktion zwei weitere Reduktionstypen einführen: die *Turing-Reduktion* und die *Levin-Reduktion* (benannt nach Alan Turing und Leonid Levin).

Dieser Abschnitt ist aufgebaut wie folgt. Zunächst schildern wir das *Problem des Handelsreisenden*, auch *Travelling Salesman Problem* oder kurz *TSP* genannt. Alle wesentlichen Ideen tauchen bereits in diesem Beispiel auf. Danach bringen wir die im TSP-Beispiel genannten Problemtypen in eine allgemeinere begrifflich gestraffte Form und führen die drei genannten Reduktionstypen formal ein.

3.1 Das Problem des Handelsreisenden

TSP liest sich als Optimierungsproblem wie folgt. Gegeben sei eine $(n \times n)$-Kostenmatrix $D = (d_{i,j})_{0 \leq i,j \leq n-1}$, wobei im Falle $i \neq j$ der Parameter $d_{i,j} \in \mathbb{N}$ die Kosten einer „Reise" von i nach j angibt. Gesucht ist die kürzeste Rundreise, also eine Permutation σ von $1, \ldots, n$, die die Kostenfunktion

$$|\sigma| := \sum_{i=0}^{n-1} d_{\sigma(i),\sigma(i+1 \bmod n)}$$

minimiert. σ_* bezeichne im Folgenden eine Permutation, die eine kürzeste Rundreise repräsentiert. Eine Abschwächung des TSP-Optimierungsproblems ist das TSP-Wertoptimierungsproblem, bei welchem nur nach der Länge einer kürzesten Rundreise gefragt ist.

TSP als Konstruktionsproblem hat als weiteren Eingabeparameter neben der Kostenmatrix D eine Kostenschranke K. Gesucht ist eine Rundreise, deren Kosten K nicht überschreiten bzw. die Meldung, dass eine solche Rundreise nicht existiert. Eine Abschwächung des TSP-Konstruktionsproblems ist das TSP-Entscheidungsproblem, bei welchem nur gefragt ist, ob eine Rundreise mit Kosten höchstens K existiert. Rundreisen (Permutationen), die die Kostenschranke K respektieren, nennen wir *zulässig*.

Im Folgenden skizzieren wir den Beweis, dass alle vier Problemvarianten polynomiell verknüpft sind, d.h., ein deterministischer Polynomialzeitalgorithmus für eine Variante kann in einen nicht wesentlich aufwändigeren Algorithmus für alle anderen Varianten transformiert werden.

Die in Abbildung 2 dargestellte Hierarchie ist offensichtlich. Hierbei bedeutet $A \to B$, dass eine polynomielle DTM M', die Problem (oder Problemvariante) B löst, in eine polynomielle DTM M für Problem (oder Problemvariante) A transformiert werden kann. Die einfache Argumentation zu den Reduktionen 1,2,3,4 überlassen wir dem Leser und der Leserin.

Auf eine griffige Formel gebracht haben wir bis jetzt gezeigt: Entscheiden ist nicht schwerer als Konstruieren, und Konstruieren ist nicht schwerer als Optimieren. Überraschender ist, dass die Problemreduktionen in der obigen Hierarchie auch in der umgekehrten Richtung möglich sind, wie es in Abbildung 3 zu sehen ist. Griffig formuliert: Optimieren ist nicht schwerer als Konstruieren, und Konstruieren ist nicht schwerer als Entscheiden.

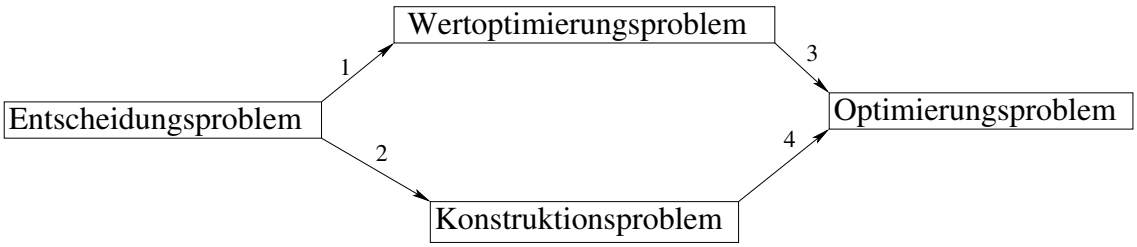


Abbildung 2: Reduktion des Entscheidungsproblems auf das Optimierungsproblem.

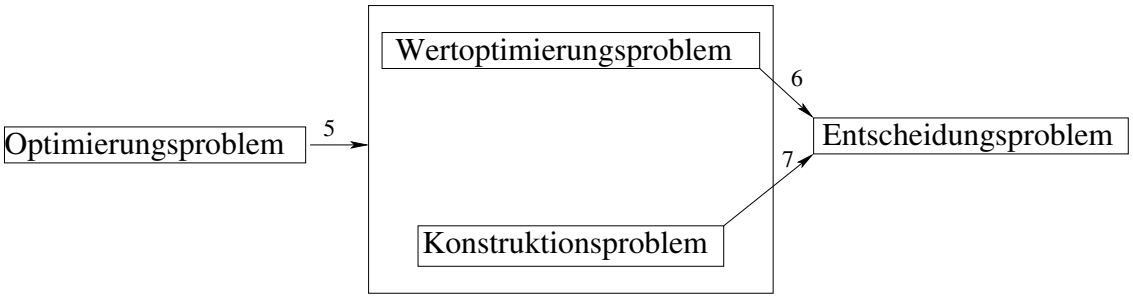


Abbildung 3: Reduktion des Optimierungproblems auf das Entscheidungsproblem.

Reduktion 5 Gegeben eine Prozedur zur Lösung des TSP-Wertoptimierungsproblems sowie eine Prozedur zur Lösung des TSP-Konstruktionsproblems. Das TSP-Optimierungsproblem kann dann wie folgt gelöst werden:

1. Bestimme die Länge K_* einer kürzesten Rundreise.
2. Bestimme eine zulässige Rundreise σ_* zu Kostenmatrix D und Kostenschranke K_*.

Reduktion 6 Gegeben eine Prozedur zur Lösung des TSP-Entscheidungsproblems, so erhalten wir die Länge K_* einer kürzesten Rundreise mit Hilfe von Binärsuche. Hierbei nutzen wir aus, dass $K^* \in \mathbb{N}$ nicht größer als $B := n \cdot \max_{i,j} d_{i,j}$ sein kann und dass die Binärsuche nach $K^* \in \{1, \ldots, B\}$ maximal $\log B$ Iterationen benötigt.

Reduktion 7 Die Reduktion des Konstruktionsproblems auf das Entscheidungsproblem ist die interessanteste von allen. Sie macht sich die Eigenschaft der "Selbstreduzierbarkeit" von TSP zunutze. Selbstreduzierbarkeit ist ein wichtiges Konzept der Komplexitätstheorie, das später noch formal sauber besprochen werden wird. Nehmen wir im Falle

von TSP oBdA an, dass eine zulässige Rundreise mit Kosten höchstens K existiert. (Andernfalls gibt es nichts zu konstruieren.) Die entscheidende Idee ist, bei jeder einzelnen "Kante" (i, j) von i nach j zu testen, ob sie für eine zulässige Rundreise entbehrlich ist. Dazu werden ihre Kosten $d_{i,j}$ versuchsweise auf $K + 1$ gesetzt. Die Entscheidungsprozedur verrät uns dann, ob trotzdem noch eine zulässige Rundreise (logischerweise dann ohne Kante (i, j)) existiert. Falls ja, dann lassen wir $d_{i,j}$ auf seinem hohen Wert. Dies ist nicht weiter schlimm, da wir uns zuvor von der Entbehrlichkeit der Kante (i, j) überzeugt haben. Falls aber die Entscheidungsprozedur signalisiert, dass keine zulässige Lösung mehr existiert, dann setzen wir $d_{i,j}$ auf seinen alten Wert zurück und nehmen Kante (i, j) in die die zu konstruierende Rundreise σ auf. Da wir nur unter Zugzwang Kanten in σ aufnehmen, entsteht auf diese Weise eine zulässige Rundreise.

3.2 Allgemeine Suchprobleme

Formale Sprachen und Entscheidungsprobleme sind zwei Seiten der selben Münze. Zu einer formalen Sprache $L \subseteq \Sigma^*$ können wir nämlich das Entscheidungsproblem assoziieren, ob ein vorgegebenes Wort $x \in \Sigma^*$ zu L gehört (das sogenannte *Mitgliedschaftsproblem* zu L). Umgekehrt können wir zu einem Entscheidungsproblem (mit den möglichen Antworten "Ja" und "Nein") die formale Sprache aller Eingabewörter assoziieren, welche zu der Antwort "Ja" führen.

Im vorigen Abschnitt haben wir das TSP-Entscheidungsproblem, das TSP-Konstruktionsproblem und das TSP-(Wert-)Optimierungsproblem diskutiert. In diesem Abschnitt werden wir den abstrakten Begriff des Suchproblems so allgemein fassen, dass alle Problemvarianten sich als Suchproblem darstellen lassen:

Definition 3.1 *Ein* Suchproblem *ist gegeben durch eine Relation $R \subseteq \Sigma^* \times \Sigma^*$. Zu einem Eingabewort $x \in \Sigma^*$ ist eine R-*zulässige Lösung[8] $y \in \Sigma^*$ *gesucht, d.h., ein y mit $(x, y) \in R$. Eine DTM* löst *das Suchproblem, wenn sie für jedes Eingabewort $x \in \Sigma^*$ ein zulässiges y ausgibt (bzw. eine entsprechende Meldung, wenn keine zulässige Lösung existiert). Im Folgenden bezeichne $\mathcal{S}(R)$ das zu Relation R assoziierte Suchproblem.*

Beispiel 3.2 *Eine formale Sprache $L \subseteq \Sigma^*$ kann als Suchproblem zur Relation*

$$R(L) = \{(x, 1) |\ x \in L\} \tag{2}$$

aufgefasst werden.

Beispiel 3.3 *Das durch eine Relation R und eine Bewertungsfunktion val gegebene Minimierungsproblem — zu gegebenem Eingabewort x finde eine R-zulässige Lösung y mit minimalem Wert $val(x, y)$ — kann als Suchproblem zur Relation*

$$R_{val} = \{(x, y_*) \in R |\ \forall y :\ (x, y) \in R \implies val(x, y) \geq val(x, y_*)\} \tag{3}$$

aufgefasst werden. Eine analoge Bemerkung gilt für Maximierungsprobleme, Wertoptimierungsprobleme und Konstruktionsprobleme.

[8] Wenn R aus dem Kontext hervorgeht, sprechen wir auch einfach von einer zulässigen Lösung

Ein Suchproblem könnte aus einem sehr banalen Grund schwer sein, zum Beispiel weil die zulässigen y für eine gegebenes Eingabewort x extrem lange Wörter sind. Evtl. braucht man dann superpolynomielle Zeit zum Lösen des Suchproblems, bloß weil das Aufschreiben der Ausgabe eine zeitraubende Angelegenheit ist. Die folgende Definition schließt diesen und weitere triviale Gründe für die Härte eines Suchproblems aus.

Definition 3.4 *Eine Relation R heisst* polynomiell beschränkt*, wenn ein Polynom p existiert so dass für alle $(x, y) \in R$: $|y| \leq p(|x|)$. R heisst* polynomiell entscheidbar*, wenn sich in Polynomialzeit entscheiden läst, ob zwei vorgegebene Wörter x, y in Relation R zueinander stehen. R heisst* polynomiell verifizierbar*, wenn R polynomiell beschränkt und polynomiell entscheidbar ist. Die Wertefunktion $val(x, y)$ heißt* polynomiell auswertbar*, wenn zu gegebenem $(x, y) \in R$ die Binärdarstellung von $val(x, y)$ in Polynomialzeit berechenbar ist.*

Beispiel 3.5 *Die Relation R_{TSP}, die dem TSP-Konstruktionsproblem entspricht, enthält genau die Paare (x, y) mit $x = (D, K)$, $y = \sigma$ und Permutation σ repräsentiert eine Rundreise mit Kosten höchstens K bezüglich der Kostenmatrix D.*[9] *R_{TSP} ist offensichtlich polynomiell verifizierbar. Beim TSP-Optimierungsproblem verwenden wir die Länge einer Rundreise als Wertefunktion. Diese ist offensichtlich polynomiell auswertbar.*

Eine besondere Rolle spielen in der Komplexitätstheorie die sogenannten NP-Relationen:

Definition 3.6 *Zu einer Relation $R \subseteq \Sigma^* \times \Sigma^*$ assoziieren wir die formale Sprache*

$$L(R) := \{x|\ \exists y :\ (x, y) \in R\}. \tag{4}$$

Falls $x \in L(R)$ und $(x, y) \in R$, dann heißt y ein Zertifikat für $x \in L(R)$.[10] *R heisst NP-*Relation*, falls $L(R) \in NP$.*

Das folgende Resultat basiert auf dem (in Abschnitt 2.2 kurz besprochenen) Rate-Verifikationsprinzip:

Lemma 3.7 *R ist eine NP-Relation genau dann, wenn R polynomiell verifizierbar ist, d.h.,*

$$NP = \{L(R)|\ R \text{ ist polynomiell verifizierbar}\}. \tag{5}$$

Beweis Falls $L \in NP$, dann gibt es ein Polynom T und eine T-zeitbeschränkte NTM M mit $L = L_M$. Wir können oBdA annehmen, dass M auf Eingabewort $x \in \Sigma^n$ nach dem Rate-Verifikationsprinzip arbeitet:

- In einer ersten Phase rät M einen binären String $y = a_1, \ldots, a_m$ mit $m = T(n)$ und schreibt diesen auf die Zellen $-1, \ldots, -m$. Auf Zelle 0 schreibt sie das Trennzeichen #.

[9] Streng genommen ist x ein Wort, das auf eine "natürliche Weise" D und K kodiert, und y ein Wort, das "auf natürliche Weise" σ kodiert. Wir gehen davon aus, dass Sie sich vorstellen können, wie man mathematische Objekte als Wörter über einem Alphabet kodiert und ziehen die informelle (aber weniger gestelzte) Formulierung vor.

[10] Aus der Definition von $L(R)$ ergibt sich, dass nur Wörter aus $L(R)$ ein solches Zertifikat besitzen!

- In der zweiten Phase arbeitet M deterministisch auf der um den Ratestring erweiterten Eingabe $y\#x$.

Wir setzen $M(x, y) = 1$, falls M in der deterministischen Phase 2 zu gegebenem Eingabewort x und gegebenem Ratestring y eine akzeptierende Rechnung vollzieht. Andernfalls setzen wir $M(x, y) = 0$. Mit

$$R = \bigcup_{n \geq 0} \{(x, y) \in \Sigma^n \times \Sigma^{T(n)} |\ M(x, y) = 1\}$$

ergibt sich dann $L = L(R)$. Offensichtlich ist Relation R polynomiell verifizierbar.
Sei nun R eine polynomiell verifizierbare Relation und $x \in \Sigma^n$. Dann gibt es zwei Polynome p, q und eine DTM V mit folgenden Eigenschaften:

- $x \in L(R) \Leftrightarrow (\exists y \in \Sigma^* : |y| \leq p(n) \wedge (x, y) \in R)$.
- V, angesetzt auf die Eingabe $y\#x$ mit $|x| = n$ und $|y| \leq p(n)$, stoppt nach maximal $q(n + p(n))$ Rechenschritten.
- V akzeptiert die Eingabe genau dann, wenn $(x, y) \in R$.

Offensichtlich ist $T(n) = p(n) + q(n + p(n))$ ein Polynom in n. Die folgende NTM M ist ein T-zeitbeschränkter Akzeptor von L:

1. Strecke den Platzbereich von Zelle -1 bis Zelle $-p(n)$ mit Hilfe zweier Markierungen ab.[11]
2. Zu gegebenem Eingabewort x mit $|x| = n$ rate einen String y mit $|y| \leq p(n)$.
3. Setze V auf die Eingabe $y\#x$ an.

Es folgt, dass $L = L_M \in NP$. •

3.3 Karp-, Cook- und Levin-Reduktionen

Die uns bereits bekannte Definition der polynomiellen Reduktion wurde von Richard Karp vorgeschlagen. In Würdigung ihres Erfinders und in Abgrenzung zu weiteren Reduktionstypen verwenden wir daher ab jetzt die Bezeichnung "Karp-Reduktion". Eine Karp-Reduktion setzt zwei formale Sprachen zueinander in Beziehung. Die Problemreduktionen, die wir im Zusammenhang mit TSP skizziert haben, verlaufen zwischen allgemeinen Suchproblemen. Um den hierfür geeigneten Reduktionsbegriff einzuführen, benötigen wir zunächst die Definition der Orakel Turing Maschine.

[11] Wir werden in Abschnitt 14.1 sehen, dass dies für ein Polynom p stets möglich ist.

Definition 3.8 *Eine TM M mit einer Relation R als* Orakel*, genannt* Orakel Turing Maschine (OTM) *und notiert als $M^{[R]}$, verfügt über ein zusätzliches* Orakelband *und drei ausgezeichnete Zustände $q_?, q_+, q_-$. Auf das Orakelband kann M ein beliebiges Wort $x \in \Sigma^*$ schreiben und dann den* Fragezustand $q_?$ *annehmen. Das Orakel ersetzt dann Wort x durch ein Wort y mit $(x, y) \in R$ (falls möglich) und M wechselt in den Zustand q_+. Falls kein zulässiges y existiert, wird x nicht ersetzt und M wechselt in den Zustand q_-. Ansonsten ändert sich die Konfiguration der Maschine bei diesem Übergang nicht. Wenn die Maschine ihre Frage auf dem Orakelband spezifiziert hat, kostet sie das Befragen des Orakels und das Erhalten der Antwort nur zwei Schritte.*[12] *Falls eine Relation der Form $R(L)$ für eine formale Sprache L als Orakel verwendet wird, sprechen wir der Einfachheit halber von einem L-Orakel und notieren die OTM als $M^{[L]}$. Deterministische (bzw. nichtdeterministische) Orakel Turing Maschinen bezeichnen wir kurz als DOTMs (bzw. NOTMs).*

Mit Hilfe der Orakel-Maschinen definieren wir Turing-Reduktionen wie folgt.

Definition 3.9 *R heisst* Cook-reduzierbar *auf R' , in Zeichen $R \leq_T R'$, falls eine polynomielle DOTM $M^{[R']}$ existiert, die in Kooperation mit einem R'-Orakel das Suchproblem $\mathcal{S}_R$ löst. Falls $R = R(L)$, dann schreiben wir der Einfachheit halber $L \leq_T R'$ statt $R(L) \leq_T R'$. Eine analoge Bemerkung gilt im Falle $R' = R(L')$.*

Offensichtlich ist eine Karp-Reduktion $L \leq_{pol} L'$ eine sehr spezielle Turing-Reduktion, bei der zunächst $f(x)$ aus x berechnet wird, um dann das Orakel für L' nach $f(x)$ zu befragen. Insbesondere wird auch nur eine einzige Frage an das Orakel gestellt. Es ist nicht schwer, die für Karp-Reduktionen bereits genannten Eigenschaften auch für Turing-Reduktionen zu beweisen:

Lemma 3.10 *1. Relation $\leq_T$ ist reflexiv und transitiv.*

2. Aus $R \leq_T R'$ und der Lösbarkeit von $\mathcal{S}_{R'}$ in Polynomialzeit folgt die Lösbarkeit von $\mathcal{S}_R$ in Polynomialzeit.

Mit Hilfe der Turing-Reduktionen können wir den Begriff der Selbstreduzierbarkeit präzise definieren.

Definition 3.11 *R heisst* selbstreduzierbar*, falls $R \leq_T L(R)$.*

NPC bezeichnet die Klasse der (bezüglich Karp-Reduktionen) *NP*-vollständigen Probleme. Eine Relation R heißt *NPC*-Relation, falls $L(R) \in NPC$. Aus Lemma 3.7 und $NPC \subseteq NP$ folgt, dass *NPC*-Relationen stets polynomiell verifizierbar sind. Wir werden später sehen, dass darüberhinaus **alle** *NPC*-Relationen selbstreduzierbar sind!

Alle am TSP-Beispiel vorgeführten Reduktionen waren Turing-Reduktionen. Man überlegt sich leicht, dass diese Reduktionen verallgemeinert werden können auf beliebige polynomiell verifizierbare und selbstreduzierbare Relationen mit polynomiell auswertbaren Bewertungsfunktionen. Da alle *NPC*-Relationen polynomiell verifizierbar und selbstreduzierbar

[12] Das Orakel spendiert die Antwort sozusagen kostenlos wie eine gute Fee.

sind, ergibt sich für eine breite Klasse von Problemen, dass Entscheiden, Konstruieren und Optimieren ungefähr gleich schwer sind. Dadurch erhalten wir die Berechtigung, uns beim Studium der Problemkomplexität auf Entscheidungsprobleme (formale Sprachen) zurückzuziehen.

Stephen Cook (Universität Toronto) und Richard Karp (Universität Berkeley) waren zwei der westlichen Protagonisten der *NP*-Vollständigkeitstheorie in den siebziger Jahren des vorigen Jahrhunderts. Zur gleichen Zeit wurde die Theorie im Osten von Leonid Levin vorangetrieben. Wir beschließen diesen Abschnitt mit dem von Levin vorgeschlagenen Reduktionstyp.

Definition 3.12 *R heisst* Levin-reduzierbar *auf R' , in Zeichen $R \leq_L R'$, wenn drei in Polynomialzeit berechenbare Abbildungen f, g, h existieren, so dass für alle $x, y, z \in \Sigma^*$ folgendes gilt:*

$$x \in L(R) \quad \Leftrightarrow \quad f(x) \in L(R') \tag{6}$$

$$(x, y) \in R \quad \Leftrightarrow \quad (f(x), g(x, y)) \in R' \tag{7}$$

$$(f(x), y') \in R' \quad \Leftrightarrow \quad (x, h(x, y')) \in R \tag{8}$$

Wir können f als eine *Eingabetransformation* auffassen, die gemäß (6) eine Karp-Reduktion von $L(R)$ auf $L(R')$ repräsentiert. Abbildungen g und h sind *Lösungstransformationen*: zu gegebenem Eingabewort x transformiert g gemäß (7) eine R-zulässige Lösung für x in eine R'-zulässige Lösung für $f(x)$; umgekehrt transformiert h gemäß (8) eine R'-zulässige Lösung für $f(x)$ in eine R-zulässige Lösung für x.

Eine Levin-Reduktion von R auf R' impliziert nicht nur eine Karp-Reduktion von $L(R)$ auf $L(R')$, sondern auch eine Turing-Reduktion von R auf R':

Lemma 3.13 $R \leq_L R' \implies R \leq_T R'$.

Beweis Folgende DOTM mit einem Orakel für R' löst das Suchproblem $\mathcal{S}_R$ in Polynomialzeit:

1. Transformiere x in $f(x)$.
2. Befrage das Orakel für R' nach einer R'-zulässigen Lösung y' für $f(x)$. (Falls diese nicht existiert, so existiert wegen (6) auch keine R-zulässige Lösung für x.)
3. Transformiere (x, y') in $h(x, y')$ und gib $h(x, y')$ aus. (Wegen (8) ist $h(x, y')$ eine R-zulässige Lösung für x.)

•

Es ist nicht schwer, die folgenden Aussagen zu zeigen:

Lemma 3.14 *1. Relation $\leq_L$ ist reflexiv und transitiv.*

2. *Aus $R \leq_L R'$ und der Lösbarkeit von $\mathcal{S}_{R'}$ in Polynomialzeit folgt die Lösbarkeit von $\mathcal{S}_R$ in Polynomialzeit.*

Definition 3.15 *Eine Relation R_0 heißt NP-*hart unter $\leq_L$ *(bzw. NP-*hart unter $\leq_T$*), falls $R \leq_L R_0$ (bzw. $R \leq_T R_0$) für jede NP-Relation R. Ist zusätzlich R_0 selber auch eine NP-Relation, dann heißt sie NP-*vollständig unter $\leq_L$ *(bzw. NP-*vollstandig unter $\leq_T$*). Diese Sprechweise übertragen sich auch auf das von R_0 repräsentierte Suchproblem.*

Wir stellen nun eine erste unter $\leq_L$ *NP*-vollständige Relation vor. Die Relation R_{SAT} enthalte alle Paare der Form (F, a), so dass folgende Bedingungen gelten:

- F repräsentiert eine Kollektion von Booleschen Klauseln, sagen wir über n Booleschen Variablen.
- $a \in \{0,1\}^n$ ist eine Boolesche Belegung dieser Variablen.
- Belegung a erfüllt alle in F enthaltenen Klauseln.

Offensichtlich gilt SAT $= L(R_{SAT})$.

Satz 3.16 *R_{SAT} ist eine unter Levin-Reduktionen NP-vollständige Relation.*

Beweis R_{SAT} ist wegen SAT $\in NP$ eine *NP*-Relation. Es bleibt zu zeigen, dass R_{SAT} unter $\leq_L$ *NP*-hart ist. Sei R eine *NP*-Relation. Somit gilt $L = L(R) \in NP$. Der Beweis des Satzes von Cook liefert eine Karp-Reduktion von $L = L(R)$ auf SAT $= L(R_{SAT})$ durch Angabe einer in Polynomialzeit berechenbaren Eingabetransformation $x \mapsto F_x$. Zum Nachweis von Folgerung 3.16 fehlt also nur die Angabe zweier geeigneter Lösungstransformationen:

1. Es muss in Polynomialzeit möglich sein, ein Paar $(x, y) \in R$ in eine Boolesche Belegung a mit $(F_x, a) \in R_{SAT}$ zu transformieren.
   $(x, y) \in R$ bedeutet, dass y ein Zertifikat für $x \in L$ ist. Im Beweis des Satzes von Cook wurde gezeigt, wie man hieraus eine F_x erfüllende Boolesche Belegung a ableiten kann. Diese Belegung ergab sich im Wesentlichen aus der deterministischen Verifikation von $(x, y) \in R$. Da die Laufzeit der Verifikationsphase polynomiell in $|x| + |y|$ (und somit polynomiell in $|x|$) beschränkt ist, erhalten wir die Transformation $(x, y) \mapsto a$ in Polynomialzeit.

2. Es muss in Polynomialzeit möglich sein, ein Paar (x, a) mit einer F_x erfüllenden Booleschen Belegung a in ein Zertifikat y mit $(x, y) \in R$ zu transformieren.
   Im Beweis des Satzes von Cook wurde demonstriert, wie sich y aus der Belegung der Booleschen Variablen der Form $S(0, -k, b)$ mit $k = 1, \ldots, p(n)$ und $b = 0, 1$ auf einfache Weise (und offensichtlich in Polynomialzeit) ablesen lässt.

Damit hat sich eine Levin-Reduktion von R auf R_{SAT} ergeben. •

4 Selbstreduzierbarkeit aller NPC-Relationen

Wir beginnen mit der Definition der $\mathcal{K}$-Relationen für eine Komplexitätsklasse $\mathcal{K}$.

Definition 4.1 *R heisst $\mathcal{K}$-Relation, falls $L(R) \in \mathcal{K}$.*

Im Spezialfall $\mathcal{K} = NP$ deckt sich diese Definition mit unserer alten Definition der NP-Relationen. Das Hauptresultat dieses Abschnittes ist die folgende Aussage über NPC-Relationen:

Theorem 4.2 *Jede NPC-Relation ist selbstreduzierbar.*

Beweis Wir beweisen zunächst die Selbstreduzierbarkeit von R_{SAT}:

Behauptung $R_{SAT} \leq_T$ SAT.

Sei $F = F(v_1, \ldots, v_n)$ eine Konjunktion von Klauseln über den Booleschen Variablen $v_1, \ldots, v_n$. Mit Hilfe eines SAT-Orakels können wir das Suchproblem zu R_{SAT} lösen wie folgt:

Vorabtest Wir fragen das Orakel, ob F erfüllbar ist. Falls nicht, können wir mit der Meldung, dass keine erfüllende Belegung existiert, abbrechen. Falls doch, dann geht es weiter wie folgt.

Konstruktion einer erfüllenden Belegung Für $i \geq 0$ und eine partielle Belegung $(a_1, \ldots, a_i) \in \{0,1\}^i$ bezeichne $F_i = F(a_1, \ldots, a_i, v_{i+1}, \ldots, v_n)$ die vereinfachte Klauselkonjunktion, die entsteht, wenn wir für $j = 1, \ldots, i$ die Variable v_j durch die Boolesche Konstante a_j ersetzen. [13] Für $i = 0$ ist noch keine Variable belegt, d.h., $F_0 \equiv F$. Für $i = n$ sind alle Variablen belegt, d.h., $F_n \in \{\text{TRUE}, FALSE\}$. Falls $F_n \equiv$ TRUE, dann handelt es sich bei a um eine erfüllende Belegung. Eine partielle Belegung heisse *gut*, wenn sie zu einer erfüllenden Belegung fortgesetzt werden kann. Da der Vorabtest garantiert, dass $F \equiv F_0$ erfüllbar ist, ist die leere Belegung gut. Unsere Strategie besteht darin, für $i = 0, \ldots, n-1$ eine gute partielle Belegung $a_1, \ldots, a_i$ mit Hilfe des SAT-Orakels zu einer guten partiellen Belegung $a_1, \ldots, a_i, a_{i+1}$ zu erweitern. Falls dies gelingt, können wir iterativ die gute leere Belegung zu einer guten vollständigen (und somit F erfüllenden) Belegung fortsetzen. Nehmen wir also an, dass $a_1, \ldots, a_i$ eine gute partielle Belegung ist. Beachte, dass dann $a_1, \ldots, a_i, 0$ oder $a_1, \ldots, a_i, 1$ eine gute partielle Belegung sein muss. Wir fragen das SAT-Orakel, ob $F(a_1, \ldots, a_i, 0, v_{i+2}, \ldots, v_n)$ erfüllbar ist. Falls ja, dann setzen wir $a_{i+1} = 0$. Falls nein, dann muss erzwungenermaßen $F(a_1, \ldots, a_i, 1, v_{i+2}, \ldots, v_n)$ erfüllbar sein, und wir setzen $a_{i+1} = 1$. Auf diese Weise gelangen wir nach n Iterationen zu einer F erfüllenden Belegung $a = (a_1, \ldots, a_n)$.

[13] Vereinfachungen: durch die partielle Belegung bereits erfüllte Klauseln können eliminiert bzw. durch TRUE ersetzt werden; in den verbleibenden Klauseln können alle Literale $v_j, \bar{v}_j$ für $j = 1, \ldots, i$ entfernt bzw. durch FALSE ersetzt werden.

Um Theorem 4.2 zu beweisen haben wir zu zeigen:

Behauptung Sei R eine NPC-Relation. Dann gilt $R \leq_T L(R)$.
Dies ist klar wegen

$$R \leq_L R_{SAT} \leq_T \text{SAT} \leq_{pol} L(R) \ .$$

Dabei haben wir der Reihe nach ausgenutzt, dass R_{SAT} NP-vollständig unter Levin-Reduktionen ist, dass R_{SAT} selbstreduzierbar ist und dass $L(R) \in NPC$ gelten muss. Da Levin- und Karp-Reduktionen spezielle Cook-Reduktionen sind, ergibt sich

$$R \leq_T R_{SAT} \leq_T \text{SAT} \leq_T L(R)$$

und somit, wegen der Transitivität von Cook-Reduzierbarkeit, schließlich $R \leq_T L(R)$. •

Die Klasse NPC ist sehr reichhaltig. Die Eigenschaft der Selbstreduzierbarkeit gilt daher für eine sehr umfangreiche Klasse von Relationen. Für alle diese Relationen ist also das Konstruieren einer zulässigen Lösung nicht wesentlich aufwendiger als das Entscheiden, ob eine solche Lösung existiert. Nichtsdestotrotz gibt es vermutlich $(NP \setminus P)$-Relationen, die nicht selbstreduzierbar sind:

COMPOSITES COMPOSITES bezeichnet die Menge der natürlichen Zahlen $N > 1$ (mit Kodierungslänge $n = \log N$), die keine Primzahlen sind. Da ein Primzahltest in Polynomialzeit (in Abhängigkeit von $n = \log N$) durchgeführt werden kann, gehört COMPOSITES zur Klasse P. Das zugehörige Suchproblem ist das *Faktorisierungsproblem*: zu gegebener Zahl N finde (falls möglich) eine Faktorisierung $N = N_1 \cdot N_2$ mit $N_1, N_2 \geq 2$. Die Frage der Selbstreduzierbarkeit ist offen. Im Falle der Selbstreduzierbarkeit würde ein COMPOSITES-Orakel erlauben, das Faktorisierungsproblem zu lösen. Wegen COMPOSITES $\in P$ könnte dann eine Zahl in Polynomialzeit in ihre Primfaktoren zerlegt werden.[14]

Ebenso gibt es selbstreduzierbare NP-Relationen R, für die vermutlich $L(R) \in NP \setminus (P \cup NPC)$ gilt:

GRAPHENISOMORPHIE Das *Graphenisomorphieproblem* ist die Frage, ob zwei vorgegebene Graphen $G = (V, E), G' = (V', E')$ mit $|V| = |V'|$ *isomorph* sind, d.h., sind beide Graphen bis auf Umbenennung der Knoten gleich? Dieses Problem liegt vermutlich in $NP \setminus (P \cup NPC)$. Das zugehörige Suchproblem verlangt nach Angabe der *Isomorphie*, also der bijektiven Abbildung $h : V \to V'$, so dass $E' = \{(h(v), h(w)) | \ (v, w) \in E\}$.

Die zentralen Ideen zum Nachweis der Selbstreduktion sind wie folgt:

- Eine partielle Lösung des Grpahenisomorphieproblems ist eine injektive Abbildung h einer Teilmenge U von V nach V', die zu einer Isomorphieabbildung fortsetzbar ist. Nehmen wir oBdA an, dass G und G' isomorph sind. Dann ist die *leere Abbildung* mit Definitionsbereich $U = \emptyset$ eine partielle Lösung.

[14] Man vermutet, dass dies nicht möglich ist, und viele Public Key Kryptosysteme basieren auf der (vermuteten) Härte des Faktorisierungsproblems.

- Die Idee ist, den Definitionsbereich einer partiellen Lösung mit Hilfe des GRAPHENISOMORPHIE-Orakels iterativ zu erweitern. Wenn wir testen könnten, ob eine Erweiterung der Form $h(w) = z$ mit $w \in V \setminus U$ immer noch eine partielle Lösung ist, dann könnten wir durch Ausprobieren aller Kandidaten z schließlich eine Fortsetzung der partiellen Lösung ausfindig machen. Schönheitsfehler: Wir verfügen nicht über einen solchen "Fortsetzungstest", sondern nur über ein GRAPHENISOMORPHIE-Orakel.

- Es verbleibt zu zeigen, dass man mit einem GRAPHENISOMORPHIE-Orakel einen virtuellen Fortsetzungstest bereitstellen kann. Sei h bisher auf $U = \{u_1, \ldots, u_i\}$ definiert und $u'_j = h(u_j)$ für $j = 1, \ldots, i$. Wir wollen testen, ob $h(u_{i+1}) = u'_{i+1}$ eine geeignete Fortsetzung ist. Zu diesem Zweck erzeugen wir zwei Hilfsgraphen H und H'. H geht aus G hervor, indem wir für $j = 1, \ldots, i+1$ dem Knoten u_j jn-viele neue Knoten als Nachbarn geben. Analog geht H' geht aus G' hervor, indem wir für $j = 1, \ldots, i+1$ dem Knoten u'_j jn-viele neue Knoten als Nachbarn geben. Es ist nicht schwer zu beweisen, dass h mit erweitertem Definitionsbereich $\{u_1, \ldots, u_i, u_{i+1}\}$ genau dann eine partielle Lösung ist, wenn H und H' isomorph sind. Der Fortsetzungstest wird also implementiert, indem wir das GRAPHENISOMORPHIE-Orakel nach der Isomorphie von H und H' befragen.

Wir fassen die letzte Diskussion zusammen:

Theorem 4.3 *Die Relation zum Graphenisomorphieproblem ist selbstreduzierbar.*

5 Die Grenze zwischen P und NPC

In diesem Abschnitt beziehen wir uns auf Handzettel, die von der Webseite zur Vorlesung runtergeladen werden können.

Eine alte Binsenweisheit besagt, dass Glück und Leid dicht beieinander liegen. Wenn wir unter Leid die *NP*-Härte eines zu lösenden Berechnungsproblems und unter Glück seine Lösbarkeit in Polynomialzeit verstehen, dann trifft die obige Feststellung auch auf den Algorithmenentwurf zu. Handzettel 2 (entnommen aus *Garey & Johnson*) zeigt, wie dicht Probleme aus P und NPC beieinander liegen können:

- Der **kürzeste** Pfad zwischen zwei Knoten in einem Graphen mit positiven Kantengewichten kann (zum Beispiel mit dem Algorithmus von Dijkstra) effizient gefunden werden. Der **längste** Pfad hingegen vermutlich nicht. Das korrespondierende Entscheidungsproblem, LONGEST PATH BETWEEN TWO VERTICES, ist nämlich (sogar eingeschränkt auf Einheitsgewichte für Kanten) *NP*-vollständig.[15]

- VERTEX COVER ist, wie wir bereits wissen, *NP*-vollständig. Es gibt also vermutlich keinen effizienten Algorithmus, der zu einem gegebenen Graphen $G = (V, E)$ die

[15] Dies kann (relativ leicht) mit einer Kette von Karp-Reduktionen nachgewiesen werden, die von HAMILTONIAN CIRCUIT über HAMILTONIAN PATH (auf die offensichtiche Weise definiert) nach LONGEST PATH BETWEEN TWO VERTICES führt.

kleinste Knotenmenge $U \subseteq V$ findet, die von jeder Kante mindestens einen Randknoten enthält. Das duale EDGE COVER Problem, das nach einer kleinsten Kantenmenge fragt, die jeden Knoten überdeckt, ist hingegen mit Hilfe von MAXIMUM MATCHING effizient lösbar.

- Das Problem MINIUMUM EQUIVALENT DIGRAPH fragt nach einer kleinsten Teilmenge $A' \subseteq A$ von Kanten eines gegebenen Digraphen $G = (V, A)$, so dass zwei beliebige $u, v \in V$ genau dann in G durch einen Pfad (von u nach v) verbunden werden können, wenn dies in $G' = (V, A')$ möglich ist. TRANSITIVE REDUCTION unterscheidet sich von MINIUMUM EQUIVALENT DIGRAPH nur dadurch, dass $A' \subseteq V \times V$, d.h., A' muss keine Teilmenge von A sein. TRANSITIVE REDUCTION kann (mit Hilfe von TRANSITIVE CLOSURE und Berechnung von starken Zusammenhangskomponenten) effizient gelöst werden. Das Entscheidungsproblem zu MINIMUM EQUIVALENT DIGRAPH hingegen enthält DIRECTED HAMILTONIAN CIRCUIT[16] als Teilproblem und ist daher *NP*-vollständig.

- Eine Eingabeinstanz zum Problem SCHEDULING WITH INDIVIDUAL DEADLINES besteht aus einer Menge T von Aufgaben (tasks), einer partiellen Ordnung $\prec$ auf T, einem individuellen Schlusstermin (deadline) $d(t)$ für jede Aufgabe $t \in T$ und einer Anzahl m von Prozessoren. Jede Aufgabe t kann in einer Zeiteinheit auf einem Prozessor ausgeführt werden. $t \prec t'$ bedeutet, dass t ausgeführt sein muss, bevor die Ausführung von t' begonnen wird. Die Frage ist, ob man die Aufgaben aus T den m Prozessoren so zuordnen kann, dass alle Schlusstermine eingehalten werden. Eine partielle Ordnung auf T heisst INTREE oder auch *Montagebaum*, wenn jedes $t \in T$ maximal einen unmittelbaren Nachfolger hat und es genau ein $t_0 \in T$ ohne Nachfolger gibt.[17] Analog spricht man von einem OUTTREE oder auch *Sortierbaum*, wenn jedes $t \in T$ maximal einen unmittelbaren Vorgänger hat und es genau einen Knoten $t_0 \in T$ ohne Vorgänger gibt.[18] INTREE SCHEDULING (bzw. OUTTREE SCHEDULING) ist die Spezialisierung von SCHEDULING WITH INDIVIDUAL DEADLINES auf partielle Ordnungen der Form INTREE (bzw. OUTTREE). Wie Brucker, Garey und Johnson 1977 gezeigt haben, kann INTREE SCHEDULING effizient gelöst werden, wohingegen OUTTREE Scheduling *NP*-vollständig ist (Reduktion von VERTEX COVER).

Es gehört zu den wünschenswerten Eigenschaften einer professionellen Informatikerin bzw. eines professionellen Informatikers, das Grenzgebiet zwischen P und NPC gut zu kennen. Der obere Teil von Handzettel 3 zeigt den Grenzverlauf. Die untere Grenzlinie von NPC wird von den am weitesten eingeschränkten *NP*-vollständigen Teilproblemen gebildet. Jede weitere Spezialisierung eines dieser Probleme führt aus NPC heraus. Die obere Grenzlinie

[16] genauer: die Einschränkung von DIRECTED HAMILTONIAN CIRCUIT auf stark zusammenhängende Digraphen, die auch *NP*-vollständig ist (gleiche Reduktion wie in der Vorlesung)

[17] Solche partiellen Ordnungen sind dann als Baum mit Wurzel t_0 darstellbar, wobei die Kanten zur Wurzel hin orientiert sind.

[18] Solche partiellen Ordnungen sind dann als Baum mit Wurzel t_0 darstellbar, wobei die Kanten von der Wurzel weg orientiert sind.

von P wird von den allgemeinsten Teilproblemen gebildet, die noch in Polynomialzeit gelöst werden können. Jede weitere Generalisierung eines dieser Probleme führt aus P heraus. Je dichter diese beiden Grenzlinien aneinanderliegen, desto kleiner ist das unbekannte Terrain der Probleme mit einem offenen Status. Die voranschreitende Forschung schiebt die Grenzlinien tendenziell aufeinander zu, da sowohl der Fundus der NP-vollständigen Probleme als auch der Fundus der effizienten Algorithmen ständig erweitert wird. Anschaulich gesprochen gibt es *kleine Engelchen*, die das bekannte Territorium von P auf immer allgemeinere Berechnungsprobleme ausdehnen, und *kleine Teufelchen*, die von immer spezielleren Problemen die NP-Vollständigkeit nachweisen. Es ist aus der strukturellen Komplexitätstheorie bekannt, dass (unter der Vorausetzung $P \neq NP$) die Klasse $NPC \setminus P$ nicht leer ist. So sehr also Engelchen und Teufelchen sich mühen, sie werden sich niemals auf einer scharfen Grenzlinie begegnen.

Wenden wir doch dieses allgemeine Schema einmal auf eine konkrete Problemhierachie an. PRECEDENCE CONSTRAINED SCHEDULING ist der Spezialfall von SCHEDULING WITH INDIVIDUAL DEADLINES bei dem die individuellen Schlusstermine alle identisch sind zu einem gemeinsamen Schlusstermin D. Mit einer von CLIQUE ausgehenden Karp-Reduktion kann man die NP-Vollständigkeit von PRECEDENCE CONSTRAINED SCHEDULING nachweisen. Aus dem unteren Teil von Handzettel 3 geht hervor, dass die Spezialfälle, bei denen die partielle Ordnung auf T leer oder ein Baum (Montage- oder Sortierbaum)[19] ist, in Polynomialzeit gelöst werden kann. Ebenso für allgemeine partielle Ordnung auf T und eine feste Anzahl 1, 2 oder 3 von Prozessoren (fleissige Engelchen). Für eine feste Anzahl von 4 oder mehr Prozessoren ist der Status des Problems offen.

Die Exploration des Grenzgebietes zwischen NPC und P geschieht in der Praxis meist für eine konkrete Problemhierarchie (wie zum Beispiel PRECEDENCE CONSTRAINED SCHEDULING).

Bei Graphenproblemen ergibt sich eine Problemhierarchie auf natürliche Weise, indem man von der Klasse aller (endlicher) Graphen zu speziellen Graphklassen übergeht. Zum Beispiel:

- Bäume
- planare Graphen
- Graphen mit beschränktem Knotengrad

Eine weitere Spezialisierung ergibt sich, indem einige der Eingabevariablen zu Konstanten "gefriert". Wir werden in Abschnitt 6 exemplarisch die Problemhierarchie zum Gra-

[19] Für Montagebäume folgt dies aus dem oben zitierten allgemeineren Resultat für INTREE SCHEDULING (individuelle Schlusstermine). Für Sortierbäume kann man im Falle eines einheitlichen Schlusstermines D den Algorithmus für Montagebäume zusammen mit einem Symmetrieargument verwenden: Erstens, kehre im Sortierbaum SB die Orientierung aller Kanten um und erhalte einen Montagebaum MB. Zweitens, wende das Verfahren für Montagebäume an und erhalte einen optimalen Plan, der alle Aufgaben auf m Prozessoren in $T \leq D$ Zeiteinheiten $1, \ldots, T$ unter Einhaltung der (spiegelverkehrten) partiellen Ordnung MB ausführt. Drittens, spiegele den Plan, d.h., durchlaufe ihn in der zeitlichen Reihenfolge $T, \ldots, 1$. Dann wird die partielle Ordnung SB respektiert.

phenfärbungsproblem diskutieren.

Für ein Zahlenproblem erhalten wir ein natürliches Teilproblem, indem wir verlangen, dass der Maximalbetrag eines Zahlparameters der Eingabe polynomiell durch die Eingabelänge beschränkt ist. Dies führt zu der Frage, ob ein Zahlenproblem nur darum nicht zu P gehört, weil man mit wenigen Bits "riesige Zahlen" spezifizieren kann, oder ob es auch schon für "moderate Zahlen" NP-vollständig ist. Probleme der ersten Kategorie heißen *pseudopolynomiell lösbar*, Probleme der letzteren Kategorie heißen *stark NP-vollständig*. Wir werden später sehen, dass zum Beispiel PARTITION pseudopolynomiell lösbar, aber eine Verallgemeinerung davon (3-PARTITION) stark NP-vollständig ist.

6 Analyse von eingeschränkten Graphproblemen

Eine *k-Färbung* eines Graphen $G = (V, E)$ ist eine Färbung der Knoten, die monochromatische Kanten vermeidet, d.h., eine Abbildung $f : V \to \{1, \ldots, k\}$ mit $f(v) \neq f(w)$ für alle $\{v, w\} \in E$. Zu gegebener Färbung heisst die Menge aller Knoten der gleichen Farbe i die *Farbklasse* zu i. Die *chromatische Zahl* $\chi(G)$ ist die kleinste Anzahl k von Farben, die eine zulässige Färbung von G ermöglicht. Das *Graphenfärbungsproblem* (als Optimierungsproblem) besteht darin, zu einem gegebenen Graphen G eine $\chi(G)$-Färbung anzugeben. COLORABILITY sei das Problem zu gegebenem Graphen G und gegebener Kostenschranke k zu entscheiden, ob G k-färbbar ist. k-COLORABILITY sei das Teilproblem, bei dem k nicht Teil der Eingabe ist (Gefrieren einer Variable zu einer Konstanten).

Ein natürliches Anwendungsszenario für Graphenfärbung ist die Durchführung von Prozessen mit gemeinsamen Ressourcen. Zwei Prozesse stehen im Konflikt zueinander, wenn sie die selbe Ressource benötigen. Zwei solche Prozesse können nicht im gleichen Zeitabschnitt ausgeführt werden. Prozesse und ihre Ressourcenkonflikte können durch einen Konfliktgraphen modelliert werden, dessen Knoten Prozesse und dessen Kanten Ressourcenkonflikte repräsentieren. Die chromatische Zahl des Konfliktgraphen gibt die minimale Anzahl von Zeitabschnitten an, die eine konfliktfreie Ausführung zulässt.

Es ist bekannt, dass 2-COLORABILTY zu P gehört. Wir skizzieren kurz die Grundidee. Eine Teilmenge U der Knotenmenge V eines Graphen $G = (V, E)$ heisst *unabhängig (Independent Set)*, wenn die Knoten aus U paarweise in G nicht benachbart sind. Man beachte, dass jede Farbklasse eine unabhängige Menge ist. G heisst *bipartit*, wenn V sich in zwei unabhängige Mengen zerlegen lässt. Das folgende Resultat ist nicht schwer zu zeigen:

Lemma 6.1 *Die folgenden Aussagen sind äquivalent:*

1. *G ist 2-färbbar.*
2. *G ist bipartit.*
3. *G enthält keine Kreise ungerader Länge.*

Da man im Rahmen einer systematischen Graphexploration (wie DFS oder BFS) die Existenz von Kreisen ungerader Länge leicht testen kann, erhalten wir die

Folgerung 6.2 2-*COLORABILITY*$\in P$.

Es bezeichne $\Delta(G)$ den maximalen Grad eines Knotens in G. Ein naives Verfahren zum Färben von G liefert der folgende gierige Algorithmus. Sei $\Delta := \Delta(G)$. Färbe einen Knoten nach dem anderen mit Farben aus $C = \{1, \ldots, \Delta+1\}$ gemäß folgender Strategie: der aktuelle Knoten v erhält die kleinste Farbe i aus C, die noch nicht für einen seiner Nachbarn verwendet wurde.
Dieses naive Verfahren ist effizient durchführbar und kommt mit $1+\Delta(G)$ Farben aus. Somit gilt $\chi(G) \leq \Delta(G) + 1$. Für den vollständigen Graphen[20] mit n Knoten ist diese Schranke scharf, da seine chromatische Zahl n und sein maximaler Knotengrad $n-1$ ist. Darüberhinaus gilt:

Lemma 6.3 (Satz von Brooks) — *ohne Beweis* —
Es sei G ein zusammenhängender nicht vollständiger Graph. Dann gilt $\chi(G) \leq \Delta(G)$.

Folgerung 6.4 *COLORABILITY eingeschränkt auf Graphen vom Maximalgrad* 3 *gehört zu P.*

Beweis Wir zeigen, dass $\chi(G)$ effizient berechenbar ist. Da man die Zusammenhangskomponenten eines Graphen unabhängig voneinander färben kann, dürfen wir oBdA annehmen, dass der gegebene Graph zusammenhängend ist. Falls G nicht aus einem einzigen Knoten besteht, benötigen wir mindestens zwei Farben. Wie oben erwähnt können wir in Polynomialzeit testen, ob G 2-färbbar ist. Nehmen wir zu unseren Ungunsten an, dass wir mindestens 3 Farben brauchen. Da $\Delta(G) \leq 3$, brauchen wir maximal 4 Farben. Falls G eine 4-Clique ist, benötigen wir genau 4 Farben. Andernfalls garantiert der Satz von Brooks, dass drei Farben ausreichen. •

Insgesamt hat sich also gezeigt, dass COLORABILTY eingeschränkt auf zwei Farben oder auf maximalen Knotengrad 3 zu P gehört. Soweit die Nachrichten aus den himmlischen Sphären. Hören wir uns an, was die bocksbeinigen Kerlchen aus der unteren Etage zu dem Thema zu sagen haben.

Theorem 6.5 3-*COLORABILITY ist NP-vollständig. Dies gilt sogar dann, wenn wir die zulässigen Eingabeinstanzen auf planare Graphen vom maximalen Knotengrad* 4 *einschränken.*

Beweis Mitgliedschaft zu *NP* ist offensichtlich. Wir haben die *NP*-Härte nachzuweisen. Der Beweis vollzieht sich in drei Stufen.

Stufe 1 3-SAT$\leq_{pol}$ 3-COLORABILTY.

Die Reduktion erfolgt mit der „Methode der verbundenen Komponenten“. Abbildung 4 zeigt eine Übersicht. Das Herzstück der Reduktion ist die Klauselkomponente. Sie ist so kunstvoll entworfen, dass zulässige 3-Färbungen (mit den Farben $0, 1, 2$) der Klauselkomponente implizit die korrespondierende Klausel “korrekt auswerten”, wobei die Farben $0, 1$ die Booleschen Wahrheitswerte repräsentieren. Genauer gesagt gilt folgendes unter der Voraussetzung, dass für A, B, C nur die Farben $0, 1$ zur Verfügung stehen:

[20] Ein Graph heißt *vollständig*, wenn jeder Knoten zu jedem anderen Knoten benachbart ist.

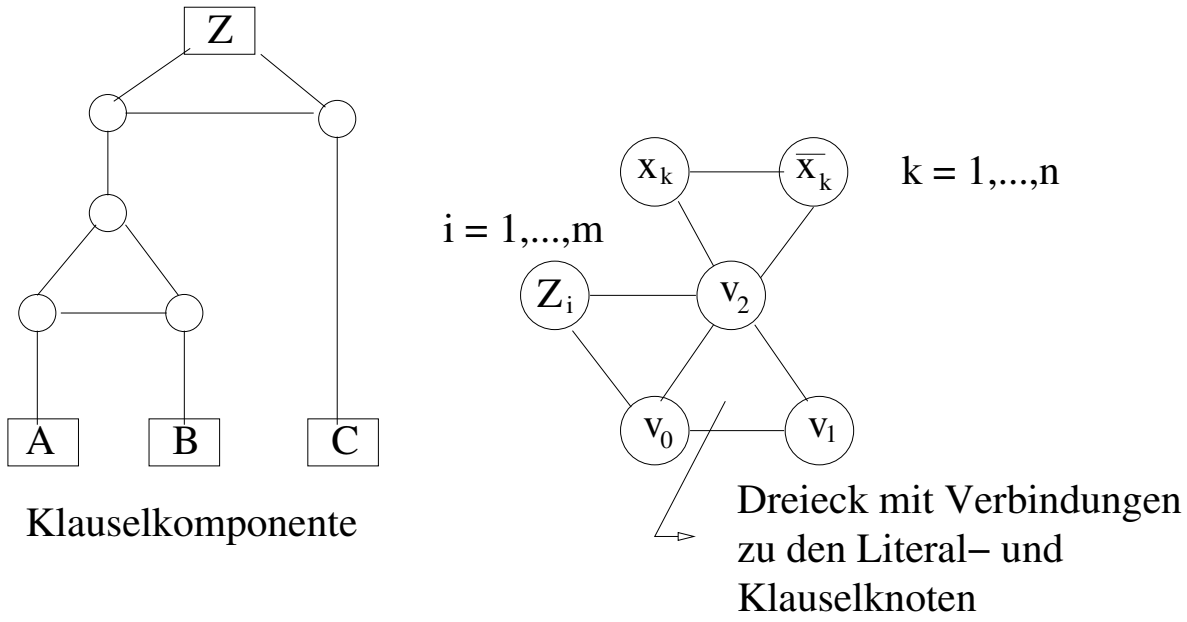


Abbildung 4: Ein Ausschnitt aus der Belegungskomponente im Falle $m = 3$: die Tripel aus T_k sind schraffiert, die Tripel aus F_k unschraffiert.

- Wenn A, B, C Farbe 0 haben, dann muss auch Z mit 0 gefärbt sein.
- Wenn einer der Knoten A, B, C die Farbe 1 hat, dann kann auch Z mit 1 gefärbt werden.

Die weiteren Kernbeobachtungen zu der Reduktion sind wie folgt:

- Das von v_0, v_1, v_2 gebildete Dreieck sorgt dafür, dass die Knoten v_0, v_1, v_2 drei verschiedene Farben erhalten: oBdA die Farben $0, 1, 2$.
- Das von $v_2, x_k, \bar{x}_k$ gebildete Dreieck sorgt dafür, dass die Farben von $x_k, \bar{x}_k$ entweder $0, 1$ oder $1, 0$ sind. Die Farben von $x_k, \bar{x}_k$ entsprechen somit einer Belegung der Booleschen Variablen x_k mit entweder 0 oder 1.
- Das von v_0, v_2, Z_i gebildete Dreieck sorgt dafür, dass Z_i mit 1 gefärbt werden muss. Intuitiv steht Z_i für den Wahrheitswert der i-ten Klausel.
- Zu jeder Klausel der Form $C_i = z_{i1} \vee z_{i2} \vee z_{i3}$ mit $z_{ij} \in \{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$ konstruieren wir eine Klauselkomponente mit z_{i1}, z_{i2}, z_{i3} in den Rollen von A, B, C und Z_i in der Rolle von Z.

Aus diesen Beobachtungen ergibt sich leicht, dass gegebene 3-Klauseln $C_1, \ldots, C_m$ genau dann erfüllbar sind, wenn der zugehörige Graph 3-färbbar ist. Ergebnis von Stufe 1: 3-COLORABILITY ist *NP*-hart.

Stufe 2 Entwurf des "Crossover-Gadget"

Wir erinnern daran, dass ein Graph planar ist, wenn man die Knoten und Kanten

so in die Ebene einbetten kann, dass Kanten sich nur an gemeinsamen Randpunkten berühren. Die in Stufe 1 benutzte Reduktion erzeugt nicht notwendig einen planaren Graphen. Wir können ihn aber mit Hilfe eines "Crossover-Gadgets" (s. oberen Teil von Handzettel 4) in einen planaren Graphen transformieren. Durch „Spielen" mit dem Gadget findet man leicht heraus, dass bei zulässigen Färbungen x, x' (bzw. y, y') die selbe Farbe haben **können** (d.h. es existiert eine zulässige 3-Färbung f mit $f(x) = f(x')$ und $f(y) = f(y')$) und auch **müssen** (d.h. für jede zulässige 3-Färbung f gilt $f(x) = f(x')$ und $f(y) = f(y')$). Dabei hat man (was das Gadget betrifft) die freie Wahl, ob x, y die selbe oder verschiedene Farben haben. Wie im mittleren Teil von Handzettel 4 zu sehen ist, können wir alle durch eine Kante (u, v) verlaufenden Kreuzungen durch verkettete Duplikate des (planaren!) Gadgets substituieren. Die Randbedingung $f(u) \neq f(v)$ für zulässige Färbungen bleibt dabei erhalten, weil die Farbe von u sich gewissermaßen durch die Kette der Gadgets bis zum Nachbarknoten von v durchpropagiert. Wenn wir jede Kante von G auf diese Weise kreuzungsfrei machen, erhalten wir einen planaren Graphen, der genau dann 3-färbbar ist, wenn der ursprüngliche Graph es war. Ergebnis von Stufe 2: 3-COLORABILITY eingeschränkt auf planare Graphen ist *NP*-hart.

Stufe 3 Entwurf des "Portal-Gadgets"

Die in Stufen 1 und 2 skizzierte Reduktion erzeugt einen Graphen mit evtl. hohem maximalen Knotengrad. Wir setzen die Reduktion so fort, dass sie den maximalen Knotengrad auf 4 reduziert. Dies kann mit Hilfe einer lokalen Ersetzung geschehen (s. den unteren Teil von Handzettel 4). Dabei wird ein Knoten v mit k Nachbarn durch ein Portal-Gadget H_k ersetzt. H_k ist ein Untergraph mit inneren Knoten (die nicht durch Kanten mit Knoten ausserhalb H_k verbunden sind) und k Portalen. Jedes der Portale ist über eine Kante mit genau einem Nachbarn von v verbunden. Die Portale haben innerhalb H_k den Grad 2, also insgesamt den Grad 3. Die inneren Knoten von H_k haben Grad 4. H_k ist so kunstvoll konstruiert, dass die Portale alle die gleiche Farbe haben können und müssen. Man hat also beim Färben des neuen Graphen die gleichen Freiheitsgrade wie zuvor.

Stufe 3 schließt den Beweis des Theorems ab. •

Die Grenze zwischen P und NP hat sich beim Graphenfärbungsproblem somit (einigermaßen) präzise bestimmen lassen.

7 Starke NP-Vollständigkeit

7.1 Beispiele für pseudopolynomielle Algorithmen

Anna und Bert lassen sich durch die neuesten Nachrichten zur NP-Vollständigkeit nicht entmutigen. Anna hat eine Idee, um das PARTITION-Problem zu lösen. Seien $w_1, \ldots, w_n \geq 0$ die gegebenen Zahlen und $S = w_1 + \cdots + w_n$ ihre Summe. Nehmen wir oBdA an, dass S

eine gerade Zahl ist. Anna will eine $n \times (1 + S/2)$-Tabelle $T = (t_{r,s})_{1 \le r \le n, 0 \le s \le S/2}$ mit den Booleschen Einträgen

$$t_{r,s} = \left(\exists I \subseteq \{1, \ldots, r\} : \sum_{i \in I} w_i = s \right) \tag{9}$$

berechnen. In Worten: die Indikatorvariable $t_{r,s}$ hat genau dann den Wahrheitswert 1, wenn sich aus den Zahlen $w_1, \ldots, w_r$ eine Auswahl mit Summenwert s treffen lässt. Die erste Zeile der Tabelle ist leicht auszurechnen:

$$t_{1,s} = ((s = 0) \vee (w_1 = s)) \,. \tag{10}$$

Gegeben Zeile $r-1$, dann lässt sich auch Zeile r mit Hilfe einer einfachen Vorschrift ausfüllen:

$$t_{r,s} = (t_{r-1,s} \vee ((s \ge w_r) \wedge t_{r-1,s-w_r})) \,.$$

Denn Summenwert s lässt sich genau dann mit einer Auswahl aus $w_1, \ldots, w_r$ erhalten, wenn er sich durch eine solche Auswahl **ohne** Teilnahme von w_r oder durch eine solche Auswahl **mit** Teilnahme von w_r erhalten lässt. Der erste Fall liegt genau dann vor, wenn man Summenwert S bereits durch eine Auswahl der Zahlen $w_1, \ldots, w_{r-1}$ erhalten kann. Der zweite Fall liegt genau dann vor, wenn $s \ge w_r$ ist und man Summenwert $s - w_r$ durch eine Auswahl der Zahlen $w_1, \ldots, w_{r-1}$ erhalten kann.
Anna kann die Rechenvorschriften (9) und (10) benutzen, um die Tabelle T zeilenweise auszufüllen. Offensichtlich gehört die Eingabeinstanz $w_1, \ldots, w_n$ genau dann zur Sprache PARTITION, wenn $t_{n,S/2}$ den Wahrheitswert 1 hat. Der Eintrag in der rechten unteren Ecke von Tabelle T verrät uns also die Lösung.
Das ist nicht übel, sagt Bert. *Ich habe übrigens eine ähnliche Methode zum Lösen von KNAPSACK (KP). Meine Tabelle ist auch 2-dimensional und hat Einträge $t_{r,s} \in \mathbb{N}_0$, die den maximalen Nutzen angeben, der sich mit den Objekten $1, \ldots, r$ unter Beachtung von Gewichtsschranke s erzielen lässt. Ich kann diese Tabelle mit einer ähnlich einfachen Rechenvorschrift systematisch ausfüllen wie Du Deine. Schon gut*, sagt Anna. *Ich kann mir vorstellen wie Du das machst.*[21] *Wir haben also beide ein NP-vollständiges Problem gelöst.*

Um die vorangehenden Überlegungen zu klaren Aussagen zusammenzufassen, benötigen wir die folgende

Definition 7.1 *Bei einem Zahlenproblem (formale Sprache) L assoziieren wir zu einer Eingabeinstanz I neben der Eingabelänge $n(I)$ die maximale Größe $M(I)$ einer der in I gegebenen Zahlparameter.*[22] *Zu einem festen Polynom p bezeichne L_p die Einschränkung von L auf Eingabeinstanzen I mit $M(I) \le p(n(I))$. L heißt* pseudopolynomiell entscheidbar*, wenn $L_p \in P$ für jedes Polynom p. Ein* pseudopolynomieller *Algorithmus für L ist eine DTM, die das Mitgliedschaftsproblem für L löst und deren Laufzeit polynomiell in $n(I)$ und $M(I)$ beschränkt ist.*

[21] die Hörer und Hörerinnen der Vorlesung hoffentlich ebenfalls (s. Übung)

[22] Wir vereinbaren, dass $M(I) = 1$, falls I keine Zahlparameter enthält. In diesem Fall sprechen wir von einem „rein kombinatorischen“ Problem.

Aus der Existenz eines pseudopolynomiellen Algorithmus für L folgt offensichtlich, dass L pseudopolynomiell entscheidbar ist. Anna's und Bert's Verfahren ergeben also folgendes

Theorem 7.2 *PARTITION und KP sind mit pseudopolynomiellen Algorithmen lösbar.*

Da SUBSET SUM ein Teilproblem von KP ist, überträgt sich dieses Resultat auf SUBSET SUM.

7.2 Erste Beispiele für stark NP-vollständige Probleme

Kann man vielleicht alle Zahlenprobleme pseudopolynomiell lösen? Zum Beispiel auch BP? Oder gibt es Zahlenprobleme, bei denen alle pseudopolynomiellen Lösungsversuche an unüberwindliche Barrieren stoßen?

Die folgende Definition liefert ein wichtiges Werkzeug zur Beantwortung dieser Frage:

Definition 7.3 *Ein Zahlenproblem (formale Sprache) L heißt* stark *NP*-hart*, wenn es ein Polynom p gibt, so dass L_p NP-hart ist. Gilt zusätzlich $L \in NP$, dann heißt L* stark *NP*-vollständig.

Wir merken an, dass rein kombinatorische Probleme (wie zum Beispiel SAT, 3-SAT, CLIQUE, DHC, HC) ohne (echte) Zahlparameter automatisch stark *NP*-vollständig sind, wenn sie *NP*-vollständig sind. Der neue Begriff macht nur Sinn bei echten Zahlenproblemen, deren Eingabeinstanzen potenziell superpolynomiell in $n(I)$ große Zahlen enthalten können. Das folgende Resultat ist offensichtlich:

Theorem 7.4 *Falls $P \neq NP$, dann kann es zu einem stark NP-vollständigen Problem keinen pseudopolynomiellen Algorithmus geben.*

Ein erstes stark *NP*-vollständiges Zahlenproblem kennen wir schon:

Theorem 7.5 *TSP ist stark NP-vollständig.*

Beweis Die Reduktion von HC auf TSP, die zum Nachweis der *NP*-Vollständigkeit von TSP verwendet verwendet wird, bildet einen Graphen $G = (V, E)$ auf eine Kostenmatrix $D = (d_{i,j})$ ab, mit $d_{i,j} = 1$ falls $\{i, j\} \in E$ und $d_{i,j} = 2$ sonst. Dann existiert nämlich ein Hamiltonscher Kreis in G genau dann, wenn D eine Rundreise mit Kosten von maximal n erlaubt. Es treten also keine Zahlparameter auf, die superpolynomiell in der Kodierungslänge der TSP-Eingabeinstanz sind. •

Mit TSP kennen wir ein stark *NP*-vollständiges Anordnungsproblem (mit Zahlparametern). Wir begeben uns nun auf die Jagd nach einem stark *NP*-vollständigen Zerlegungsproblem (mit Zahlparametern). Objekt unserer Begierde sind die folgenden Probleme:

3-DM Perfektes 3-Dimensionaless Matching

Eingabe Drei disjunkte gleichmächtige Mengen $X = \{x_1, \ldots, x_q\}$, $Y = \{y_1, \ldots, y_q\}$, $Z = \{z_1, \ldots, z_q\}$ sowie eine Menge $M \subseteq X \times Y \times Z$ von Tripeln. Hierbei reicht es, M in der Eingabe zu spezifizieren, da X, Y, Z implizit durch die Komponenten der Tripel aus M gegeben sind.

Frage Gibt es eine Auswahl $T \subseteq M$ von q Tripeln aus M, so dass jedes Element aus $X \cup Y \cup Z$ in genau einem Tripel vorkommt?

3-PARTITION Partition in m Tripel mit gleichen Teilsummen

Eingabe $n = 3m$ Zahlen $a_1, \ldots, a_n \in \mathbb{N}$ mit einer Gesamtsumme $B = a_1 + \cdots + a_n$ der Form $B = mb$, $b \in \mathbb{N}$. Für $i = 1, \ldots, n$ gelte $b/4 < a_i < b/2$.

Frage Kann man diese Zahlen in m gleich große Teilsummen zerlegen, d.h., existiert eine Zerlegung von $\{1, \ldots, n\}$ in Mengen $I_1, \ldots, I_m$ mit

$$\forall j = 1, \ldots, m : \sum_{i \in I_j} a_i = b? \tag{11}$$

Beachte, dass wegen $b/4 < a_i < b/2$ die Bedingung (11) höchstens dann erfüllbar ist, wenn jedes I_j dreielementig ist.

4-Partition Partition in m Quadrupel mit gleichen Teilsummen

Eingabe $n = 4m$ Zahlen $a_1, \ldots, a_n \in \mathbb{N}$ mit einer Gesamtsumme $B = a_1 + \cdots + a_n$ der Form $B = mb$, $b \in \mathbb{N}$. Für $i = 1, \ldots, n$ gelte $b/5 < a_i < b/3$.

Frage Kann man diese Zahlen in m gleich große Teilsummen zerlegen, d.h., existiert eine Zerlegung von $\{1, \ldots, n\}$ in Mengen $I_1, \ldots, I_m$ mit

$$\forall j = 1, \ldots, m : \sum_{i \in I_j} a_i = b? \tag{12}$$

Beachte, dass wegen $b/5 < a_i < b/3$ die Bedingung (12) höchstens dann erfüllbar ist, wenn jedes I_j vierelementig ist.

Wir merken kurz an, dass 2-DM (Perfektes 2-Dimensionales Matching) das zu 3-DM analoge Problem mit Paaren anstelle von Tripeln ist. Bei 2-DM geht es dann um die Frage, ob eine Auswahl $T \subseteq M$ von q Paaren aus M existiert, so dass jedes Element in $X \cup Y$ in genau einem Paar vorkommt. Da man sich die Elemente aus X als *Frauen*, die Elemente aus Y als *Männer*, die Paare aus M als *verträgliche Paarbildungen* und die Auswahl $T \subseteq M$ als ein *perfektes Heiratssystem* vorstellen kann, ist 2-DM auch unter dem Namen *Heiratsproblem* bekannt.[23] Mit Maximum Matching Techniken kann man zeigen (s. Vorlesung *Effiziente Algorithmen*):

[23] 3-DM ist das Heiratsproblem für eine Gesellschaft, in welcher sich neben *männlich* und *weiblich* ein dritter Sextypus etabliert hat. Es wird Traditionalisten zutiefst befriedigen, dass (wie sich im Verlaufe dieses Abschnittes noch zeigen wird) 3-DM *NP*-vollständig ist.

Lemma 7.6 *2-DM* $\in P$.

Wir beabsichtigen, die folgende Kette

$$\text{3-SAT} \leq_{pol} \text{3-DM} \leq_{pol} \text{4-PARTITION}_p \tag{13}$$

von Reduktionen zu entwerfen, wobei p ein Polynom ist. Damit hätten wir 4-PARTITION als stark *NP*-vollständiges Zahlenproblem etabliert.

Lemma 7.7 *3-SAT*$\leq_{pol}$*3-DM.*

Beweis Sei $C = (C_0, \ldots, C_{m-1})$ mit $C_i = z_{i1} \vee z_{i2} \vee z_{i3}$ und $z_{ij} \in \{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$ eine Eingabeinstanz für 3-SAT. Wir geben eine Eingabetransformation $C \mapsto M$ an, die C eine Tripelmenge M zuordnet. C soll genau dann erfüllbar sein, wenn wir für M ein perfektes 3-dimensionales Matching $T \subseteq M$ finden. Die Tripelmenge M besteht aus der *Belegungskomponente* M^B, der *Klauselerfüllungskomponente* M^K und der *Ergänzungskomponente* M^E:

Belegungskomponente Zu jeder Variable x_k, $k = 1, \ldots, n$, assoziieren wir die folgenden Tripel:

$$\begin{aligned} T_k &= \{(\bar{x}_{k,i}, a_{k,i}, b_{k,i}) |\ i = 0, \ldots, m-1\} \\ F_k &= \{(x_{k,i}, a_{k,i+1 \bmod m}, b_{k,i}) |\ i = 0, \ldots, m-1\} \end{aligned}$$

Wir setzen $M_k^B = T_k \cup F_k$ und $M^B = \cup_{k=1}^n M_k^B$. Abbildung 5 illustriert diese Konstruktion. Die Elemente $x_{k,i}, \bar{x}_{k,i}$ nennen wir *Literalelemente*. Wir werden sehen, dass sie auch noch in anderen Komponenten vorkommen. Die Elemente $a_{k,i}$ und $b_{k,i}$ sind *interne Elemente* von M_k^B, d.h., sie kommen in keinen Tripeln außerhalb M_k^B vor. Man überlegt sich leicht, dass es zu einer Zerlegung der internen Elemente von M_k^B in Tripel nur zwei Möglichkeiten gibt: **entweder** alle Tripel aus T_k **oder** alle Tripel aus F_k. Intuitiv verknüpfen wir mit der Entscheidung für T_k die Vorstellung, x_k mit 1 zu belegen, und mit der Entscheidung für F_k die Vorstellung, x_k mit 0 zu belegen.

Klauselerfüllungskomponente Zu jeder Klausel C_i, $i = 0, \ldots, m-1$, assoziieren wir die folgenden Tripel:

$$\begin{aligned} S_i^+ &= \{(x_{k,i}, c_i, d_i) |\ 1 \leq k \leq n, x_k \in C_i\} \\ S_i^- &= \{(\bar{x}_{k,i}, c_i, d_i) |\ 1 \leq k \leq n, \bar{x}_k \in C_i\} \end{aligned}$$

Wir setzen $M_i^K = S_i^+ \cup S_i^-$ und $M^K = \cup_{i=0}^{m-1} M_i^K$. Die Elemente c_i und d_i sind *interne Elemente* von M_i^K, d.h., sie kommen in keinen Tripeln außerhalb M_i^K vor. Um diese Elemente in die Tripelzerlegung mit einzubeziehen, sind wir gezwungen, ein Tripel aus M_i^K auszuwählen. Intuitiv verbinden wir damit die Auswahl eines C_i erfüllenden Literals.

Ergänzungskomponente Die Ergänzungskomponente wird dazu dienen, noch unüberdeckte Literalelemente in die Tripelzerlegung einzubeziehen. Die internen Elemente von M^B und M^K werden durch $nm + m$ Tripel exakt überdeckt werden. Damit sind auch bereits $nm + m$ Literalelemente überdeckt. Von den insgesamt $2nm$ Literalelementen wären dann noch $(n-1)m$ unüberdeckt. Dies motiviert die folgende Definition der Ergänzungskomponente:

$$M^E = \{(x_{k,i}, e_l, f_l), (\bar{x}_{k,i}, e_l, f_l |\ 1 \leq k \leq n, 0 \leq i \leq m-1, 1 \leq l \leq (n-1)m\}\ .$$

e_l, f_l sind die *internen Elemente* von M^E.

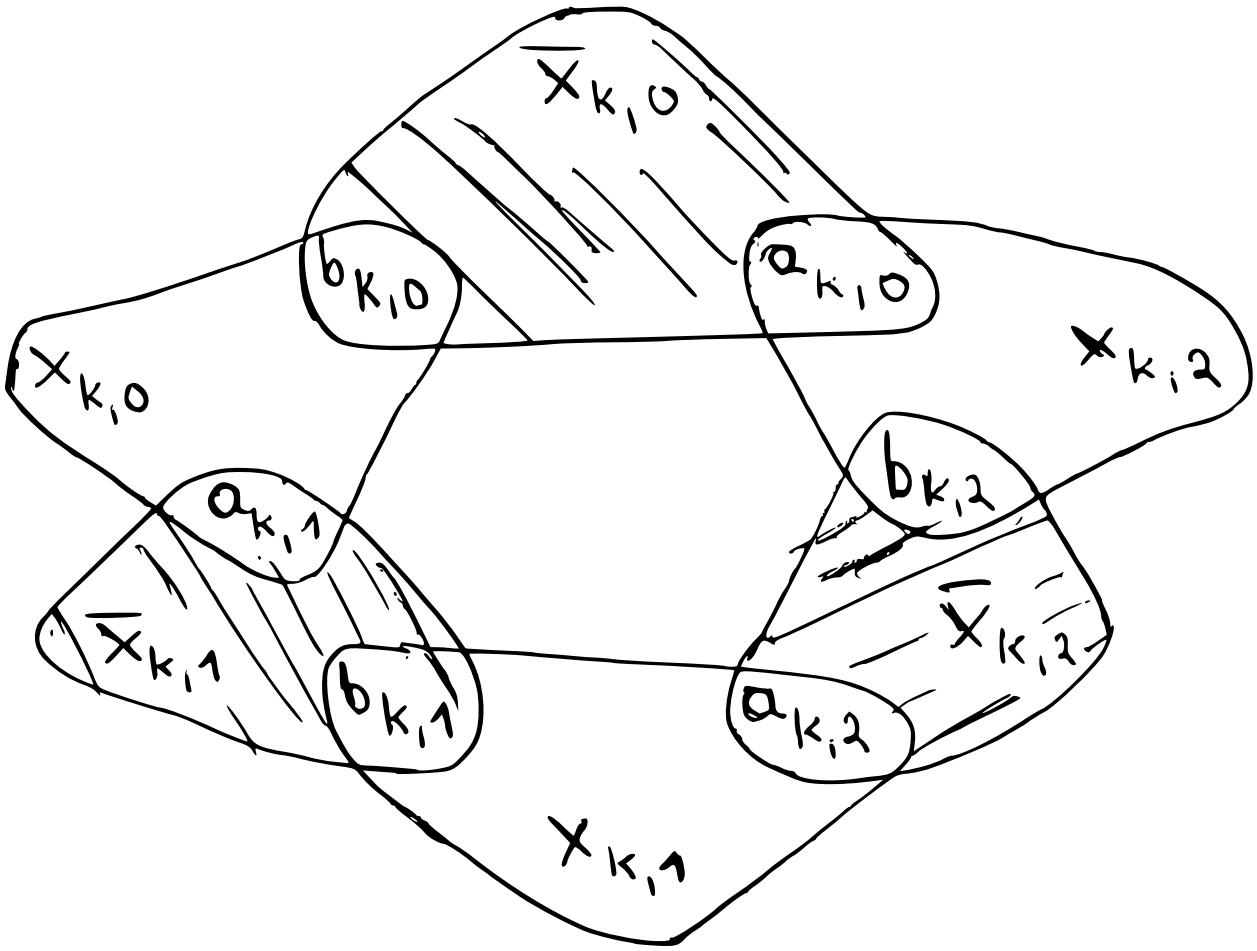


Abbildung 5: Ein Ausschnitt aus der Belegungskomponente im Falle $m = 3$: die Tripel aus T_k sind schraffiert, die Tripel aus F_k unschraffiert.

Die Transformation $C \mapsto M$ mit $M = M^B \cup M^K \cup M^E$ ist offensichtlich in Polynomialzeit berechenbar. Der Beweis wird abgeschlossen durch die

Behauptung C ist genau dann erfüllbar, wenn für M ein perfektes 3-dimensionales Matching existiert.

Nehmen wir zunächst an, dass eine C erfüllende Belegung gegeben ist. Wir bilden ein perfektes 3-dimensionales Matching T für M nach folgender Strategie:

1. Für $k = 1, \ldots, n$ nimm alle Tripel aus T_k in T auf, falls $x_k = 1$, und nimm alle Tripel aus F_k in T auf, falls $x_k = 0$.
   Effekt Alle internen Elemente von M^B sind exakt überdeckt; ebenso alle Literalelemente $x_{k,i}$ mit $x_k = 0$ bzw. alle Literalelemente $\bar{x}_{k,i}$ mit $x_k = 1$. Die anderen Literalelemente (die zu erfüllten Literalen korrespondieren) sind noch unüberdeckt.

2. Fixiere zu jeder Klausel C_i, $0 \leq i \leq m-1$, ein erfülltes Literal $z_{i,j}$. Es existiert ein k, so dass $z_{i,j} \in \{x_k, \bar{x}_k\}$. Falls $z_{i,j} = x_k$, dann nimm das Tripel $(x_{k,i}, c_i, d_i) \in S_i^+$ in T auf. Falls $z_{i,j} = \bar{x}_k$, dann nimm das Tripel $(\bar{x}_{k,i}, c_i, d_i) \in S_i^-$ in T auf. Beachte, dass in beiden Fällen ein noch unüberdecktes Literalelement überdeckt wurde.
   Effekt Alle internen Elemente von M^K sind exakt überdeckt. Insgesamt $nm + m$ Literalelemente sind nun exakt überdeckt. Somit sind $(n-1)m$ Literalelemente noch unüberdeckt.

3. Benutze — in der offensichtlichen Weise — $(n-1)m$ Tripel aus M^E, um die noch unüberdeckten Literalelemente sowie die internen Elemente von M^E exakt zu überdecken.

Nehmen wir nun umgekehrt an, dass ein perfektes 3-dimensionales Matching T für M gegeben ist. Wie bereits oben angemerkt, muss T für jedes k entweder alle Tripel aus T_k oder alle Tripel aus F_k enthalten. Im ersten Fall setzen wir $x_k = 1$, im zweiten Fall $x_k = 0$. Wir zeigen, dass die so erhaltene Belegung in jeder Klausel mindestens 1 Literal erfüllt. Die Tripel in $T \cap M^B$ lassen genau die Literalelemente unüberdeckt, die zu erfüllten Literalen korrespondieren. Da die Tripel aus $T \cap M^K$ alle internen Elemente von M^K überdecken, muss es zu jeder Klausel ein erfülltes Literal geben: die erste Komponente des c_i, d_i überdeckenden Tripels ist nämlich ein von $T \cap M^B$ noch unüberdecktes Literalelement, das zu einem C_i erfüllenden Literal korrespondiert. •

Lemma 7.8 *Es existiert ein Polynom p mit 3-DM$\leq_{pol}$4-PARTITION$_p$.*

Beweis Sei $M \subseteq X \times Y \times Z$ mit $X = \{x_1, \ldots, x_q\}$, $Y = \{y_1, \ldots, y_q\}$ und $Z = \{z_1, \ldots, z_q\}$ eine gegebene Tripelmenge. Zu jedem $w \in W = X \cup Y \cup Z$ bezeichne $L(w)$ die Anzahl der Tripel, in denen w vorkommt. Sei weiter $m = |M|$ und $r = 32q$. Wir können oBdA $m \geq q$ voraussetzen, da kein perfektes 3-dimensionales Matching für M mit weniger als q Tripeln existiert. Die zu M korrespondierende Eingabeinstanz für 4-PARTITION bestehe aus den folgenden Zahlen:

Elementzahlen Zu jedem $w \in W$ assoziieren wir die *Hauptzahl* $a_1(w)$ und die *Nebenzahlen*

$a_l(w)$ gemäß folgender Definition:

$$\begin{aligned}
a_1(x_i) &= 10r^4 + ir + 1,\ 1 \le i \le q \\
a_l(x_i) &= 11r^4 + ir + 1,\ 1 \le i \le q,\ 2 \le l \le L(x_i) \\
a_1(y_j) &= 10r^4 + jr^2 + 2,\ 1 \le j \le q \\
a_l(y_j) &= 11r^4 + jr^2 + 2,\ 1 \le j \le q,\ 2 \le l \le L(y_j) \\
a_1(z_k) &= 10r^4 + kr^3 + 4,\ 1 \le k \le q \\
a_l(z_k) &= 8r^4 + kr^3 + 4,\ 1 \le k \le q,\ 2 \le l \le L(z_k)
\end{aligned}$$

Tripelzahlen Zu jedem $(x_i, y_j, z_k) \in M$ assoziieren wir die Zahl

$$a(x_i, y_j, z_k) = 10r^4 - kr^3 - jr^2 - ir + 8.$$

Wir erhalten insgesamt eine Menge $\mathcal{A}$ bestehend aus $\sum_{w \in W} L(w) = 3m$ Elementzahlen und m Tripelzahlen. Es wird sich weiter unten implizit ergeben, dass die Gesamtsumme aller $4m$ Zahlen genau mb beträgt, wobei

$$b = 40r^4 + 15.$$

Man verifiziert leicht, dass jede Zahl aus $\mathcal{A}$ größer als $b/5$ und kleiner als $b/3$ ist. (Daher können nur Zahlquadrupel aus $\mathcal{A}$ die Teilsumme b ergeben.) Die Transformation $M \mapsto \mathcal{A}$ ist offensichtlich in Polynomialzeit berechenbar. Da alle Zahlen in $\mathcal{A}$ kleiner als $12r^4 = 12 \cdot (32q)^4 \le 12 \cdot (32m)^4 \le 12 \cdot (8|\mathcal{A}|)^4$ sind, ist der maximale Zahlparameter in $\mathcal{A}$ polynomiell in Eingabelänge $n(\mathcal{A}) \ge |\mathcal{A}|$ beschränkt. Der Beweis wird daher abgeschlossen durch folgende **Behauptung** Es existiert genau dann ein perfektes 3-dimensionales Matching für M, wenn $\mathcal{A}$ sich in m Quadrupel (vierelementige Teilmengen) zerlegen lässt, so dass jedes Quadrupel die Teilsumme b liefert.

Nehmen wir zunächst an, dass ein perfektes 3-dimensionales Matching T für M vorliegt. Wir bilden für jedes der m Tripel aus M ein Zahlenquadrupel nach der folgenden Strategie:

Fall 1 Sei $(x_i, y_j, z_k) \in T$. Dann bildet $a(x_i, y_j, z_k)$ zusammen mit $a_1(x_i), a_1(y_j), a_1(z_k)$ ein Quadrupel. Die Summe dieser vier Zahlen ist offensichtlich b.

Fall 2 Sei $(x_i, y_j, z_k) \in M \setminus T$. Dann bildet $a(x_i, y_j, z_k)$ zusammen mit $a_{l_1}(x_i), a_{l_2}(y_j), a_{l_3}(z_k)$ ein Quadrupel. Hierbei werden $l_1, l_2, l_3 \ge 2$ so gewählt, dass keine Nebenzahl mehrfach verwendet wird. (Der Vorrat an Nebenzahlen ist genau passend bemessen.) Die Summe dieser vier Zahlen ist wiederum b.

Nehmen wir jetzt umgekehrt an, dass eine erfolgreiche Zerlegung von $\mathcal{A}$ in Zahlenquadrupel vorgegeben ist. Sei (A_1, A_2, A_3, A) eines dieser Quadrupel. Aus $A_1 + A_2 + A_3 + A = b$ folgt

$$(A_1 + A_2 + A_3 + A \bmod r) = (b \bmod r) = 15.$$

Diese Gleichung kann nur eingehalten werden, wenn Indizes $i, j, k, i', j', k', l_1, l_2, l_3$ existieren, so dass die Komponenten des Quadrupels die Form

$$A = a(x_i, y_j, z_k),\ A_1 = a_{l_1}(x_{i'}),\ A_2 = a_{l_2}(y_{j'}),\ A_3 = a_{l_3}(z_{k'})$$

haben. Die Rechnung modulo r^2 ergibt dann

$$(i'r + 1) + 2 + 4 + (-ir + 8) = (A_1 + A_2 + A_3 + A \bmod r^2) = (b \bmod r^2) = 15.$$

Hieraus können wir $i' = i$ ableiten. Die Rechnung modulo r^3 liefert

$$(ir + 1) + (j'r^2 + 2) + 4 + (-jr^2 - ir + 8) = (A_1 + A_2 + A_3 + A \bmod r^3) = (b \bmod r^3) = 15.$$

Hieraus können wir $j' = j$ ableiten. Die Rechnung modulo r^4 ergibt

$$\begin{aligned}(ir + 1) + (jr^2 + 2) + (k'r^3 + 4) + (-kr^3 - jr^2 - ir + 8) \quad &= \\ (A_1 + A_2 + A_3 + A \bmod r^3) = (b \bmod r^3) &= 15.\end{aligned}$$

Es folgt $k' = k$. Der Koeffizient von r^4 bei der Summenzahl $b = 40r^4 + 15$ hat den Wert 40. Die Summe von A_1, A_2, A_3, A muss daher auch Koeffizient 40 vor dem Term r^4 liefern. Eine Inspektion der Elementzahlen ergibt, dass A_1, A_2, A_3 **entweder** drei Hauptzahlen **oder** drei Nebenzahlen sein müssen. (Im ersten Fall ergibt sich Koeffizient 40 als eine Summe der Form $10 + 10 + 10 + 10$, im zweiten als Summe der Form $11 + 11 + 8 + 10$.) Man überzeugt sich nun leicht, dass die Hauptzahl-Quadrupel ein perfektes 3-dimensionales Matching für M repräsentieren. •

Folgerung 7.9 *4-PARTITION ist stark NP-vollständig.*

7.3 Weitere stark NP-vollständige Probleme

NP-Härte vererbt sich entlang von Karp-Reduktionsketten. Reduktionen, die starke *NP*-Härte erhalten, sind der Gegenstand der folgenden

Definition 7.10 *Seien $L, L' \subseteq \Sigma^*$. Eine* pseudopolynomielle Reduktion *von L auf L' ist eine Abbildung $f : \Sigma^* \to \Sigma^*$ mit den folgenden Eigenschaften:*

1. *$f(x)$ kann in poly$(|x|, M(x))$ Schritten berechnet werden.*[24]
2. *Für alle $x \in \Sigma^*$: $x \in L \Leftrightarrow f(x) \in L'$.*
3. *$|x| \leq$ poly$(|f(x)|)$ und $M(f(x)) \leq$ poly$(|x|, M(x))$.*

Das folgende Resultat ist offensichtlich:

Theorem 7.11 *Seien L und L' Zahlenprobleme und sei L pseudopolynomiell Karp-reduzierbar auf L'. Falls ein pseudopolynomieller Algorithmus für L' existiert, dann existiert auch ein pseudopolynomieller Algorithmus für L. Falls andererseits L stark NP-hart ist, dann ist auch L' stark NP-hart.*

[24] $M(x)$ ist hierbei das Maximum der Zahlparameter, die in der Eingabe mit Codewort x auftreten. Falls die Eingabeinstanzen zur Sprache L keine Zahlparameter enthalten, kann $M(x)$ als Argument von poly gestrichen werden.

Mit Hilfe pseudopolynomieller Reduktionen können wir ausgehend von 4-PARTITION weitere stark *NP*-vollständige Zahlenprobleme ausfindig machen:

Lemma 7.12 *(ohne Beweis)*
4-PARTITION ist pseudopolynomiell Karp-reduzierbar auf 3-PARTITION.

Lemma 7.13 *4-PARTITION und 3-PARTITION sind pseudopolynomiell Karp-reduzierbar auf BP.*[25]

Beweis Wir führen den Beweis für 3-PARTITION. (Der Beweis für 4-PARTITION ergibt sich mit einem analogen Argument.) Es handelt sich um eine Transformation mit Spezialisierung: 3-PARTITION (so wie in Abschnitt 7.2 definiert) kann als ein Teilproblem von BP betrachtet werden, bei welchem die Binkapazität auf b und die Kostenschranke (Schranke für die erlaubte Anzahl von Bins) auf m gesetzt wird. m Bins der Kapazität b reichen nämlich genau dann aus, wenn sich die gegebene Zahlenkollektion (mit Gesamtsumme mb) in m Tripel mit Teilsumme b zerlegen lässt. •

Folgerung 7.14 *3-PARTITION und BP sind stark NP-vollständig.*

Wir bemerken abschließend, dass 3-PARTITION für pseudopolynomielle Karp-Reduktionen ein ähnlich beliebtes Quellproblem ist wie 3-SAT für (gewöhnliche) Karp-Reduktionen.

8 Polynomielle versus NP-harte Approximation

Ein *Approximationsalgorithmus* (kurz: PTAA[26]) zu einem Optimierungsproblem Π ist ein polynomieller Algorithmus A, der zu einer Eingabeinstanz I eine "zulässige Lösung" σ ausgibt, die eine "mathematische Gütegarantie" besitzt. Dabei sagen wir A hat *Güte* $k \geq 1$, wenn A zu jeder Eingabeinstanz I eine Lösung σ konstruiert, deren Wert vom Wert der optimalen Lösung multiplikativ maximal um Faktor k abweicht. Ein PTAA der Güte 2 für ein Maximierungsproblem würde also stets mindestens die Hälfte des optimalen Profites erbeuten. Ein PTAA der Güte 2 für ein Minimierungsproblem würde höchstens doppelt soviel Kosten wie nötig verursachen. Es ist naheliegend zu versuchen, ein *NP*-hartes Optimierungsproblem fast-optimal mit einem PTAA zu lösen, dessen Güte möglichst nahe bei 1 liegen sollte. I.A. wird es eine Barriere k_0 geben, so dass PTAAs A_k einer Güte $k > k_0$ existieren, nicht aber PTAAs einer Güte $k < k_0$ (unter der $P \neq NP$ Voraussetzung). Grenzfälle sind (formuliert für Minimierungsprobleme):

Keine konstante Gütegarantie $k_0 = \infty$.
Jeder PTAA produziert Lösungen, deren Kosten die minimalen Kosten um einen beliebig großen Faktor überschreiten können.

[25] Zur Definition von BP (Bin Packing) s. die Problemliste im Anhang zum Skript.
[26] = Polynomial Time Approximation Algorithm

Approximationsschema $k_0 = 1$.
Für jedes $k > 0$ existiert ein PTAA A_k der Güte $1 + 1/k$.

Wir behandeln das Thema der PTAAs in diesem Abschnitt nur am Beispiel des Problems des Handelsreisenden und am Beispiel des Rucksackproblems. Es wird sich folgendes Bild ergeben:

- TSP besitzt keine konstante Gütegarantie (sofern $P \neq NP$).
- Für „Metrisches TSP“ (ein Teilproblem von TSP) gibt es hingegen einen PTAA der Güte 1.5.
- Für KNAPSACK gibt es ein Approximationsschema.

Am Ende des Abschnittes diskutieren wir kurz grundsätzliche Barrieren beim Design von PTAAs in Form von Resultaten zur NP-harten Approximation.

8.1 Approximierbarkeit von TSP

Für TSP (in seiner allgemeinen Form) kann man sich klar machen, dass keine konstante Gütegarantie existiert (außer wenn $P = NP$). Wir reduzieren zu diesem Zweck das Problem des Hamiltonschen Kreises (HC) in geeigneter Weise auf das Problem des Handelsreisenden (TSP). Sei $G = (V, E)$ der Eingabegraph zu HC. Knotenmenge V bestehe aus $n \geq 2$ Knoten, die wir mit den Nummern von 1 bis n identifizieren. Will man lediglich HC $\leq_{pol}$ TSP nachweisen, genügt die uns bereits bekannte Reduktion:
Setze in der $(n \times n)$-Kostenmatrix D den Eintrag $d_{i,j}$ auf 1 falls $\{i, j\} \in E$, und andernfalls auf 2.
Hieraus ergibt sich zwar die NP-Härte von TSP, aber es ist noch nicht ausgeschlossen, dass vernünftige PTAAs existieren. Betrachten wir aber nun die folgende leichte Modifikation der alten Reduktion:
Setze in der $(n \times n)$-Kostenmatrix D den Eintrag $d_{i,j}$ auf 1 falls $\{i, j\} \in E$, und andernfalls auf kn.
Es folgt, dass bez. D genau dann eine Rundreise mit Kosten n existiert, wenn G einen Hamiltonschen Kreis enthält. Falls jedoch in G kein Hamiltonscher Kreis existiert, muss die Rundreise mindestens eine Kante außerhalb von E verwenden. In diesem Fall betragen die Kosten mindestens $n - 1 + kn$. Gäbe es einen PTAA für TSP der Güte k, so würde im Falle der Existenz eines Hamiltonschen Kreises in G eine Rundreise mit Kosten höchstens kn produziert, andernfalls jedoch eine Rundreise mit Kosten mindestens $kn + n - 1 > kn$. Mit anderen Worten: mit Hilfe der approximativen Lösung des TSP könnten wir HC exakt lösen. Falls $P \neq NP$, ist dies jedoch nicht möglich. Somit gibt es keine garantierte Güte k für PTAAs zu TSP (außer wenn $P = NP$).

Die Theorie der NP-Vollständigkeit hat uns signalisiert, dass es vermutlich Zeitverschwendung ist, nach einem PTAA für TSP zu suchen. Betrachten wir nun aber die Einschränkung von TSP auf Kostenmatrizen D, die symmetrisch sind, d.h.,

$$\forall 1 \leq i < j \leq n : \; d_{ij} = d_{ji},$$

und die Dreiecksungleichung erfüllen, d.h.,

$$\forall 1 \leq i, j, k \leq n : \; d_{ik} \leq d_{ij} + d_{jk}.$$

Der Parameter $d_{i,j}$ lässt sich dann als Distanz zwischen i und j auffassen. Statt von „Kosten einer Rundreise“ sprechen wir beim metrischen TSP auch von der „Länge einer Rundreise“. Die Bedingungen der Symmetrie und der Dreiecksungleichung sind für praktische Anwendungen durchaus vernünftig, denn sie besagen in salopper Formulierung:

Symmetrie Von A nach B ist es so weit wie von B nach A.

Dreiecksungleichung Von A nach C kann es nicht weiter sein als von A über B nach C.

Das durch Symmetriebedingung und Dreiecksungleichung eingeschränkte TSP-Problem wird als *Metrisches TSP* bezeichnet. Eine Eingabeinstanz des metrischen TSP lässt sich auf die offensichtliche Weise als vollständiger, ungerichteter Graph mit Kantengewichten $d_{i,j}$ auffassen.

Wir stellen zur Bequemlichkeit ein paar (teilweise schon bekannte) graphentheoretische Konzepte bereit, die beim Entwurf eines PTAAs für das metrische TSP eine Rolle spielen. Sei G ein ungerichteter Graph. Ein *Weg* in G ist eine Folge $v_1, \ldots, v_r$ von $r \geq 1$ Knoten, wobei für alle $i = 1, \ldots, r-1$ Knoten v_i und v_{i+1} durch eine Kante verbunden sein müssen. (Ein Grenzfall ist der aus nur einem Knoten bestehende “Punktweg”.) Falls $r \geq 2$ und $v_1 = v_r$, dann heisst der Weg auch *geschlossener Weg* oder *Kreis*. Eine *Euler-Tour* in G ist ein Kreis, der jede Kante in G genau einmal durchläuft. G heisst *zusammenhängend*, wenn zwei Knoten sich stets durch einen Pfad miteinander verbinden lassen. Ein *ungerichteter Baum* ist ein zusammenhängender, kreisloser ungerichteter Graph. Wenn man aus einem Baum eine Kante entfernt (ohne die Randknoten der Kante dabei mitzuentfernen), zerfällt er in zwei Teile. Ein Baum ist gewissermaßen die ökonomischste Art, alle Knoten durch Pfade miteinander zu verbinden. Ein *Untergraph* von $G = (V, E)$ ist gegeben durch eine Knotenmenge $V' \subseteq V$ und alle Kanten aus E, die Knoten aus V' miteinander verbinden. Man spricht auch von dem *durch V' in G induzierten Untergraphen*. Ein *Teilgraph* von $G = (V, E)$ ist ein Graph $G' = (V', E')$ mit $V' \subseteq V$ und $E' \subseteq E$. Ein *Spannbaum (spanning tree)* von G ist ein Teilgraph der Form $T = (V, E')$, der ein Baum ist. Er enthält also alle Knoten von G und verbindet diese baumartig. Ein solcher Spannbaum kann natürlich nur dann existieren, wenn G zusammenhängend ist. Im Falle von Kantengewichten können wir $T = (V, E')$ die Kosten

$$c(T) = \sum_{e \in E'} w(e)$$

zuordnen. Ein *minimaler Spannbaum (Minimum Spanning Tree) oder kurz MST)* ist ein Spannbaum minimaler Kosten. Es ist bekannt, dass minimale Spannbäume effizient berechnet werden können (zum Beispiel durch den Algorithmus von Kruskal).

Mit diesem Wissen ausgestattet ist es nun leicht, einen PTAA der Güte 2 für das metrische TSP zu skizzieren. Es sei $G = (V, E)$ der vollständige ungerichtete Graph mit n Knoten $1, \ldots, n$. Kante $\{i, j\}$ erhält als Gewicht die Distanz $d_{i,j}$ zwischen i und j. Wir berechnen

einen minimalen Spannbaum $T = (V, E')$ von G (zum Beispiel mit dem Algorithmus von Kruskal). Seien c die Gesamtkosten von T. Im Beispiel aus Abbildung 6 erhalten wir die Rundreise

$$R(T) = 3, 5, 8, 5, 1, 5, 7, 10, 7, 5, 3, 2, 9, 4, 9, 6, 9, 2, 3,$$

wenn wir (wie in Abbildung 6 angedeutet) einmal um T herumlaufen und dabei die angetroffenen Knoten der Reihe nach auflisten. Da Rundreise $R(T)$ jede Kante von T zweimal durchläuft, hat sie Länge $2c$. $R(T)$ hat noch einen Schönheitsfehler: die Knoten werden i.A. mehrfach besucht. Wir erhalten eine zulässige Lösung von TSP — also eine Permutation $P(T)$ von $1, \ldots, n$ —, wenn wir in $R(T)$ alle Vorkommen von Knoten außer dem ersten (also alle Duplikate) streichen. In unserem Beispiel führt dies zu

$$P(T) = 3, 5, 8, 1, 7, 10, 2, 9, 4, 6.$$

Wegen der Dreiecksungleichung kann $P(T)$ nicht länger als $R(T)$ sein. Die Kosten von $P(T)$ sind also maximal $2c$.

Wie verhält es sich nun mit einer optimalen Rundreise? Im Graphen G formt diese einen Hamiltonschen Kreis C. Entfernen wir aus C (irgend-)eine Kante, entsteht ein (sehr spezieller) Spannbaum $T(C)$ (s. Abbildung 7). Die Länge von Rundreise C ist nicht kleiner als die Gesamtkosten von $T(C)$. Diese wiederum betragen mindestens c. Also hat C mindestens die Länge c und der skizzierte PTAA hat die Güte 2. Es lässt sich **zeigen**, dass diese Analyse scharf ist, d.h., der MST-basierte Algorithmus hat keine Güte unterhalb von 2. **Übg.**

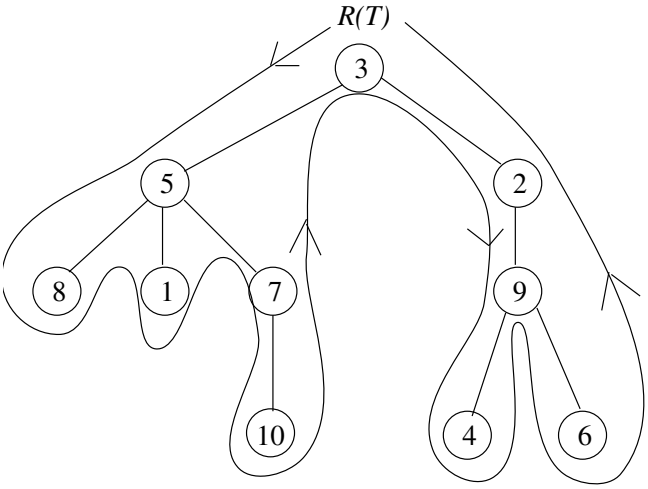


Abbildung 6: Die aus einem Minimum Spanning Tree abgeleitete Rundreise.

Vom bisherigen Erfolg berauscht stecken wir uns jetzt ein ehrgeizigeres Ziel: Entwurf eines PTAAs der Güte 1.5. Zu diesem Zweck muss unser bisheriger PTAA noch etwas "aufgepeppt" werden. Der Faktor 2, der uns vom Optimum trennt, resultiert daraus, dass die konstruierte Rundtour jede Kante des minimalen Spannbaumes T implizit 2-mal durchläuft. Wenn wir eine Euler-Tour durch T zur Verfügung hätten, würden wir jede Kante nur 1-mal durchlaufen.

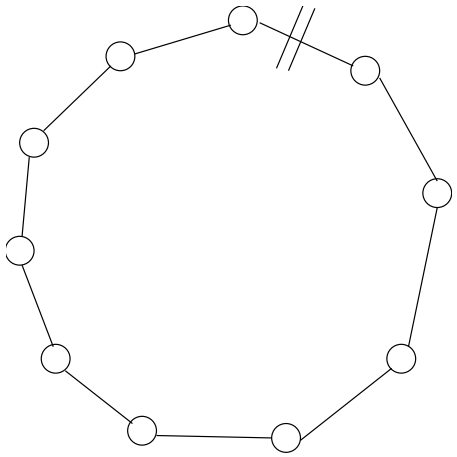


Abbildung 7: Der aus einer Rundreise abgeleitete Spanning Tree.

Dies wirft die Frage auf, für welche Graphen Euler-Touren existieren. Die Antwort liefert folgendes

Lemma 8.1 *Ein zusammenhängender Graph (Mehrfachkanten zugelassen!) besitzt genau dann eine Euler-Tour, wenn jeder Knoten einen geraden Grad (also geradzahlig viele Nachbarn) besitzt.*[27]

Beweis Wenn ein Knoten v mit ungeradem Grad, sagen wir Grad $2d+1$ existiert, dann kann es keine Euler-Tour geben: wenn der Kreis den Knoten v zum $d+1$-mal betritt, kann er ihn nicht mehr verlassen (Sackgassenargument).
Wenn ein zusammenhängender Graph G nur Knoten mit geradem Grad besitzt, dann kann eine Euler-Tour (effizient) konstruiert werden wie folgt:

1. Konstruiere einen ersten Kreis C, indem Du solange auf G spazieren gehst (ohne eine Kante doppelt zu laufen), bis Du zum Ausgangspunkt zurückkehrst.[28]

2. Wenn C bereits eine Euler-Tour darstellt, dann gib C aus. Andernfalls mache weiter mit Schritt 3.

3. Wähle einen Knoten v auf C, der noch unbenutzte Ausgangskanten hat.[29] Konstruiere von v aus einen zweiten Kreis C' (wieder durch „spazieren gehen“). Verschmelze[30] C und C' zu einem neuen Kreis und nenne diesen wieder C. Gehe zurück zu Schritt 2.

[27] In diesem Fall spricht man auch von einem *Euler'schen Graphen.*

[28] Da jeder Knoten geraden Grad hat, kann man den Spaziergang stets fortsetzen, solange man noch nicht den Ausgangspunkt erreicht hat.

[29] Da G zusammenhängend ist, muss es einen solchen Knoten v auf C geben.

[30] Laufe durch C bis zum Erreichen von v, dann durchlaufe C' bis zum erneuten Erreichen von v und durchlaufe schließlich den Rest des temporär unterbrochenen Kreises C.

•

Aus dem Beweis ergibt sich die

Folgerung 8.2 *Es kann in Polynomialzeit getestet werden, ob ein Graph G eine Euler-Tour enthält. Gegebenenfalls kann diese auch in Polynomialzeit*[31] *konstruiert werden.*

Zurück zur Frage, ob es eine Euler-Tour durch den minimalen Spannbaum T gibt? Leider nein! In seiner Eigenschaft als Baum muss T Knoten ungeraden Grades besitzen (zum Beispiel alle Blätter). Es gilt aber immerhin das folgende

Lemma 8.3 *Jeder Graph G hat geradzahlig viele Knoten ungeraden Grades.*

Beweis Wenn wir die Knotengrade $d_1, \ldots, d_n$ addieren zählen wir jede der m Kanten zweimal (da jede Kante zwei Randknoten besitzt):

$$\sum_{i=1}^{n} d_i = 2m \ .$$

Da die rechte Seite der Gleichung eine gerade Zahl ist, muss es auf der linken Seite geradzahlig viele ungeradzahlige Terme geben. •

Idee Es seien $u_1, \ldots, u_{2k}$ die Knoten ungeraden Grades in T. Ergänze T zu einem Euler'schen Graphen, indem Du k „Heiratskanten" hinzufügst, die die Knoten u_i „perfekt verheiraten". Wähle hierzu k Heiratskanten mit minimalem Gesamtgewicht (ein sogenanntes „Minimum Weight Perfect Matching", welches in Polynomialzeit berechenbar ist).

Hieraus ergibt sich der folgende von Christofides vorgeschlagene PTAA:

1. Gegeben die n-Clique (bestehend aus den n Knoten) mit Kantengewichten $d_{i,j}$, berechne einen minimalen Spannbaum T.
2. Ergänze T (wie soeben beschrieben) durch ein „Minimum Weight Perfect Matching" für seine Knoten ungeraden Grades zu einem Euler'schen Graphen T'.
3. Konstruiere eine Euler-Tour durch T' und gib diese, nach Streichung aller Duplikate, als Rundreise aus.

Beispiel 8.4 *Der Baum T aus Abbildung 6 hat* 6 *Knoten ungeraden Grades, nämlich die Knoten 1,4,6,8,9,10. Wenn wir zum Beispiel die Matchingkanten* $\{1,8\},\{4,10\},\{6,9\}$ *zu T hinzufügen*[32]*, entsteht ein Euler'scher Graph. Eine Euler-Tour wäre zum Beispiel*

$$3,5,8,1,5,7,10,4,9,6,9,2,3 \quad \textit{bzw.} \quad 3,5,8,1,7,10,4,9,6,2$$

nach Streichung von Duplikaten.

[31] Auf einer „Random Access Maschine" können wir „Polynomialzeit" durch „Linearzeit" präzisieren.

[32] Die Kante $\{6,9\}$ gibt es jetzt zweimal, nämlich einmal als Baum- und ein weiteres Mal als Matchingkante. Mehrfachkanten sind hier zugelassen!

Satz 8.5 *Der von Christofides konstruierte PTAA für TSP hat die Güte* 1.5.

Beweis Die (im Folgenden näher ausgeführte) zentrale Beweisidee ist in Abbildung 8 veranschaulicht. Es sei R_* eine optimale Rundreise der Länge $c(R_*)$. Wir wissen bereits, dass $c(R_*) \geq c(T)$. Der Algorithmus von Christofides konstruiert eine Rundreise R der Länge $c(T)+c(M)$, wobei M das „Minimum Weight Perfect Matching“ bezeichnet und $c(M)$ das Gesamtgewicht der dabei beteiligten Kanten. Die Güte 1.5 ergibt sich, wenn wir $c(R_*) \geq c(M)/2$ nachweisen können. Zu diesem Zweck sei R'_* sie Subtour von R_*, die resultiert, wenn wir Knoten geraden Grades in R_* auslassen. R_* enthält $2k$ Kanten, sagen wir $e_1, \ldots, e_{2k}$. Offensichtlich bildet sowohl $\{e_1, e_3, \ldots, e_{2k-1}\}$ als auch $\{e_2, e_4, \ldots, e_{2k}\}$ ein perfektes Heiratssystem für die Knoten $u_1, u_2, \ldots, u_{2k}$. Somit gilt

$$c(R_*) \geq c(R'_*) \geq 2c(M) \ ,$$

was den Beweis abschließt. •

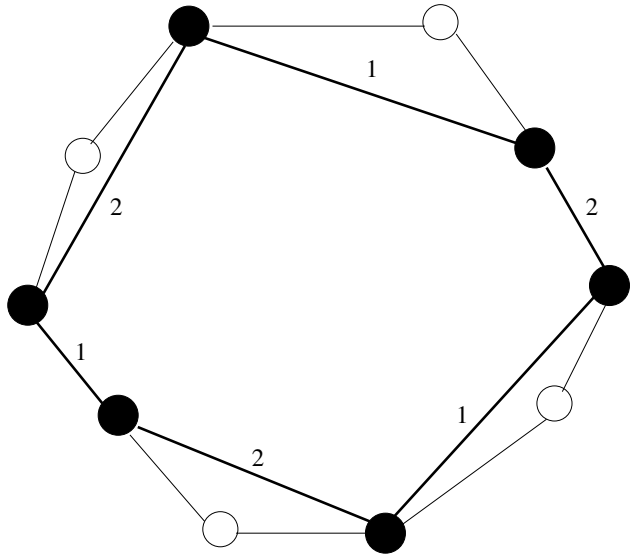


Abbildung 8: Die zwei in der Subtour von R^* enthaltenen perfekten Heiratssysteme: Knoten ungeraden Grades in T sind geschwärzt; Matchingkanten sind im Fettdruck gezeichnet)

8.2 Approximierbarkeit von „Vertex Cover“

Vertex Cover (VC) ist folgendes Problem:

Eingabeinstanz: ein Graph $G = (V, E)$

Frage: Ermittle eine möglichst kleine Knotenmenge $C \subseteq V$ (genannt „Vertex Cover“), mit der sich alle Kanten kontrollieren lassen, d.h., jede Kante des Graphen hat mindestens einen ihrer beiden Randknoten in C.

Beim entsprechenden Entscheidungsproblem ist zusätzlich eine Schranke $k \in \mathbb{N}$ gegeben und es wird gefragt, ob sich alle Kanten in G mit k (oder weniger) Knoten kontrollieren lassen.

Theorem 8.6 *Es gibt einen PTAA der Güte* 2 *für VC.*

Proof Betrachte folgenden PTAA:

1. Berechne ein inklusionsmaximales Matching $M \subseteq E$ in G, d.h., $M \subseteq E$ ist eine Kollektion knotendisjunkter Kanten, aber jede Kante in $E \setminus M$ hat einen Randknoten mit einer Kante aus M gemeinsam (so dass sich das Matching M nicht erweitern lässt).

2. Sei C die Menge der Randknoten der Kanten aus M. Gib C aus und stoppe.

Die folgendenBeobachtungen zeigen, dass tatsächlich ein PTAA der Güte 2 vorliegt:

1. Jede Kante aus M wird von zwei Knoten aus C kontrolliert. Da M inklusionsmaximal ist, wird jede Kante aus $E \setminus M$ von einem Knoten aus C kontrolliert.

2. Da M ein Matching ist, gilt $|C| = 2|M|$.

3. Da M ein Matching ist, kann kein Knoten mehr als eine Kante in M kontrollieren. Daher besteht ein kleinstmögliches Vertex Cover für G aus mindestens $|M|$ Knoten.

Die erste Beobachtung zeigt, dass C ein Vertex Cover für G ist. Die letzten beiden Beobachtungen zeigen, dass der skizzierte PTAA die Güte 2 besitzt. •

8.3 Approximierbarkeit von KNAPSACK

Sahni hat 1975 folgende PTAAs A_k für KNAPSACK (Eingabe: Gewichte $w_1, \ldots, w_n$, Profite $p_1, \ldots, p_n$ und Gewichtsschranke W) vorgeschlagen:

1. Sortiere die n Objekte nach ihrer „Profitrate“, so dass $p_1/w_1 \geq \cdots \geq p_n/w_n$.

2. Zu jeder Teilmenge $I \subseteq \{1, \ldots, n\}$ mit $|I| \leq k$ bestimme ihr Gesamtgewicht $W(I) = \sum_{i \in I} w_i$ und den durch sie realisierten Profit $P(I) = \sum_{i \in I} p_i$.

3. Für jedes I mit mit $|I| \leq k$ und $W(I) \leq W$ bestimme einen „Kandidatenrucksack“ $R(I)$, in den zunächst die durch I bestimmten Objekte aufgenommen werden, um ihn danach (unter Beachtung der Gewichtsschranke) mit anderen Objekten aufzufüllen. Beim Auffüllen erhalten Objekte mit höherer Profitrate höhere Priorität.

4. Wähle schließlich den profitabelsten Kandidatenrucksack.

Beispiel 8.7 *Wir betrachten eine Eingabeinstanz für KNAPSACK mit* $n = 8$ *Objekten und Gewichtsschranke* $W = 110$*. Sowohl die weiteren Eingabeparameter als auch der Beispiellauf von* A_0 *sind aus folgender Tabelle ersichtlich:*

i	1	2	3	4	5	6	7	8
p_i	11	21	31	33	43	53	55	65
w_i	1	11	21	23	33	43	45	55
b_i	1	1	1	1	1	0	0	0
W_Σ	1	12	33	56	89	89	89	89
P_Σ								139

Hierbei ist folgendes zu beachten:

- *Die Eingabeparameter sind bereits nach absteigender Profitrate sortiert.*
- *b_i bezeichnet ein Indikatorbit, welches mit dem Wert 1 anzeigt, dass Objekt i in den Rucksack gesteckt wurde.*
- *W_Σ ist eine dynamische Variable, die on-line das bisher akkumulierte Gewicht mitzählt. Mit Hilfe von W_Σ kann entschieden werden, ob das nächste inspizierte Objekt ohne Überschreitung der Gewichtsschranke $W = 110$ in den Rucksack gesteckt werden kann.*
- *P_Σ bezeichnet in entsprechender Weise den akkumulierten Profit. Da uns hier nur der endgültige Wert interessiert, haben wir die Zwischenergebnisse nicht angegeben.*

Wir halten als Ergebnis fest, dass A_0 die Lösung $I_0 = \{1, 2, 3, 4, 5\}$ mit Profit 139 (und Gewicht 89) berechnet.

Betrachten wir nun einen Beispiellauf von A_1. Es sei daran erinnert, dass I mit $|I| = k$ die Menge der Objekte (bestehend aus 1 Objekt im Falle $k = 1$) bezeichnet, die vorab in den Rucksack gesteckt werden. Da A_0 die Objekte $1, 2, 3, 4, 5$ ausgewählt hat, würden die Beispielläufe mit $I = \{1\}, \{2\}, \{3\}, \{4\}, \{5\}$ nur das Ergebnis $I_0 = \{1, 2, 3, 4, 5\}$ reproduzieren. Wir können uns also auf die Kandatenrucksäcke $R(I)$ mit $I = \{6\}, \{7\}, \{8\}$ beschränken. Diese drei Beispielläufe sind in den folgenden Tabellen zu sehen (wobei die Eintragungen für das vorab in $R(I)$ aufgenommene Objekt zur besseren Kenntlichkeit **fett** *gedruckt sind):*

i	**6**	1	2	3	4	5	7	8
p_i	**53**	11	21	31	33	43	55	65
w_i	**43**	1	11	21	23	33	45	55
b_i	**1**	1	1	1	1	0	0	0
W_Σ	**43**	44	55	76	99	99	99	99
P_Σ								149

i	**7**	1	2	3	4	5	6	8
p_i	**55**	11	21	31	33	43	53	55
w_i	**45**	1	11	21	23	33	43	55
b_i	**1**	1	1	1	1	0	0	0
W_Σ	**45**	46	57	78	101	101	101	101
P_Σ								151

i	**8**	*1*	*2*	*3*	*4*	*5*	*6*	*8*
p_i	**65**	*11*	*21*	*31*	*33*	*43*	*53*	*55*
w_i	**55**	*1*	*11*	*21*	*23*	*33*	*43*	*45*
b_i	**1**	*1*	*1*	*1*	*0*	*0*	*0*	*0*
W_Σ	**55**	*66*	*67*	*88*	*88*	*88*	*88*	*88*
P_Σ								*118*

Der beste Kandidatenrucksack (und somit die Ausgabe von A_1) ist $R(\{7\})$. Er enthält die Objekte $1, 2, 3, 4, 7$ und erzielt einen Profit von 151 (bei einem Gewicht von 101). Man überlegt sich leicht, dass die optimale Lösung die Objekte $1, 2, 3, 5, 6$ in den Rucksack steckt. Sie erzielt einen Profit von 159 (bei einem Gewicht von 109). Algorithmus A_2 hätte die optimale Bepackung des Rucksackes beim Beispiellauf mit $I = \{5, 6\}$ aufgespürt.

Der Algorithmus A_0 geht im Prinzip nur Profitraten-basiert vor (da er vorab in den Rucksack nur „die leere Menge packt“ und ihn danach Profitraten-basiert auffüllt). Anhand von „teuflisch“ ausgewählten Eingabeinstanzen **lässt sich zeigen**, dass A_0 keine konstante Güte c mit $c < \infty$ besitzt. Wenn allerdings „kleine Engelchen“ die Eingabeinstanzen auswählen, dann ist A_0 gar nicht so übel: **Übg.**

Lemma 8.8 *Die Objekte $1, \ldots, n$ seien absteigend nach Profitrate sortiert. Falls die Eingabeinstanz die Bedingung*

$$\exists j \in \{1, \ldots, n\} : \sum_{i=1}^{j} w_i = W \tag{14}$$

erfüllt, dann packt A_0 einen optimalen Rucksack.

Beweis Wir dürfen oBdA annehmen, dass alle Gewichte echt positiv sind. Dann ist der Index j mit $\sum_{i=1}^{j} w_i = W$ eindeutig bestimmt und A_0 packt den Rucksack R^*, der genau die Objekte $1, \ldots, j$ enthält. Setze $r = p_j/w_j$. Für $i = 1, \ldots, n$ setze $p'_i = r \cdot w_i$. Offensichtlich gilt $p'_i \leq p_i$ für alle $i \leq j$, $p'_j = p_j$ und $p'_i \geq p_i$ für alle $i > j$. Wir zerlegen p_i gemäß $p_i = p'_i + (p_i - p'_i)$ und betrachten p'_i als den „Basisprofitwert“ und $p_i - p'_i$ als den „Extraprofitwert“ des Objektes i. Für $i \leq j$ sind die Extrapofitwerte nicht-negativ; für $i > j$ sind sie nicht-positiv. Die Basisprofitraten p'_i/w_i sind für alle $i \in [n]$ identisch zu r. Daher erzielt ein Rucksack R mit Gesamtgewicht $W(R) \leq W$ den Basisprofit $rW(R) \leq rW$. Damit ist rW der maximal erzielbare Basisprofit. Offensichtlich gilt: der von A_0 gepackte Rucksack R^* erzielt den maximalen Basisprofit rW und sein Extraprofit ist identisch mit der Summe aller nicht-negativen Extraprofitwerte. Hieraus folgt umittelbar, dass R^* ein optimal gepackter Rucksack ist. • warum?

PTAA A_k mit $k \geq 1$ kommt auch mit Eingabeinstanzen ganz gut zurecht, die sich ein „Teufel“ ausgedacht hat:

Satz 8.9 *A_k ist ein PTAA für KNAPSACK der Güte $1 + 1/k$.*

Beweis Es bezeichne I_* die Indexmenge eines profitabelsten Rucksackes R_* mit Profit $P_* = \sum_{i \in I_*} p_i$. Falls $|I_*| \leq k$, dann wäre R_* einer der Kandidatenrucksäcke von A_k. In diesem Fall würde A_k den maximalen Profit erzielen. Für die weitere Diskussion können wir uns also auf den Fall $|I_*| \geq k+1$ konzentrieren. Wir indizieren die Objekte so um, dass $I_* = \{1, \ldots, k, k+1, \ldots, k+l\}$, wobei die Objekte $1, \ldots, k$ die k profitabelsten Objekte in R_* seien. Weiterhin seien die Objekte $k+1, \ldots, k+l$ absteigend nach ihrer Profitrate geordnet. Da $1, \ldots, k$ die k profitabelsten Objekte in R_* sind und da auch $k+1, \ldots, k+l$ zu R_* gehören, folgt

$$\forall \lambda = 1, \ldots, l: \; p_{k+\lambda} \leq \frac{1}{k+1} \left(p_{k+\lambda} + \sum_{i=1}^{k} p_i \right) \leq \frac{P_*}{k+1} \; . \tag{15}$$

Wir betrachten nun den Kandidatenrucksack $R(I)$ mit $I = \{1, \ldots, k\}$. Dieser Rucksack enthält von vorne herein die Objekte $1, \ldots, k$ mit Gesamtgewicht $W_0 := \sum_{i=1}^{k} w_i$ und Gesamtprofit $P_0 = \sum_{i=1}^{k} p_i$. Wir wollen den von $R(I)$ auf $R := \{k+1, \ldots, n\}$ erzielten Profit mit $P_* - P_0$ vergleichen. Beachte, dass A_k auf R bezüglich Gewichtsschranke $W_R = W - W_0$ Profitraten-basiert vorgeht (so wie Algorithmus A_0). Es bezeichne $k+\lambda' \in \{k+1, \ldots, k+l\}$ den kleinsten Index eines nicht in $R(I)$ aufgenommenen Objektes (den es geben muss, wenn $R(I)$ und R_* nicht übereinstimmen, was wir oBdA annehmen). Weiterhin bezeichne I' die Menge der Objekte aus R, die sich zum Zeitpunkt der Inspektion von Objekt $k+\lambda'$ in $R(I)$ befinden, so dass A_k auf R mindestens den Profit $P' := \sum_{i \in I'} p_i$ erzielt (und damit insgesamt mindestens den Profit $P_0 + P'$). Da Objekt $k+\lambda'$ nicht in $R(I)$ aufgenommen wurde, muss

$$W' := w_{k+\lambda'} + \sum_{i \in I'} w_i > W_R$$

gelten. Bezüglich Gewichtsschranke W' an Stelle von W_R wäre folgendes geschehen:

- A_k hätte Objekt $k+\lambda'$ ebenfalls aufgenommen (und damit Gewichtsschranke W' genau erreicht).
- Gemäß Lemma 8.8 wäre $I' \cup \{k+\lambda'\}$ eine optimale Lösung auf der Objektmenge R bezüglich der Gewichtsschranke W' gewesen.

Hieraus folgt nun, dass

$$P_* - P_0 \leq P' + p_{k+\lambda'} \overset{(15)}{\leq} P' + \frac{P_*}{k+1} \; ,$$

woraus sich mit einer leichten Rechnung $P_*/(P_0 + P') \leq 1 + 1/k$ ergibt. Da der von $R(I)$ erzielte Profit mindestens $P_0 + P'$ beträgt, ist der Beweis jetzt abgeschlossen. •

Anhand von „teuflisch“ ausgewählten Eingabeinstanzen für KNAPSACK **lässt sich zeigen**, dass A_k mit $k \geq 1$ keine konstante Güte c mit $c < 1 + 1/k$ besitzt. **Übg.**

Der Algorithmus A_k ist für jede Konstante k polynomiell zeitbeschränkt. Wie aber ist die Abhängigkeit der Zeitschranke von dem Güteparameter k?[33] Die Antwort ist etwas frustrierend: allein schon die Anzahl der Kandidatenrucksäcke (sprich: die Anzahl aller Teilmengen $I \subseteq \{1, \ldots, n\}$ mit $|I| \leq k$) ist proportional zu n^k. Die Laufzeit wächst also exponentiell mit k. Das Approximationsschema von Sahni ist daher nur für kleine Werte von k praktikabel.

Definition 8.10 *Ein Algorithmus A für ein Optimierungsproblem Π, der neben der eigentlichen Eingabe einen zusätzlichen Eingabeparameter k erhält und für festes k einen PTAA A_k der Güte $1 + 1/k$ für Π repräsentiert, heißt* volles Approximationsschema*, wenn sich seine Laufzeit nach oben durch ein bivariates Polynom in der Eingabelänge N und in k beschränken lässt.*

Ibarra und Kim haben 1975 mit einer Technik namens „Rounding and Scaling“ ein volles Approximationsschema für KNAPSACK entworfen. Wir skizzieren im Folgenden die Grundidee hiervon. Es bezeichne $P_n = \sum_{i=1}^{n} p_i$ die Summe aller Einzelprofite, P_* den vom optimalen Rucksack erzielten Profit, $p_{max} := \max_{i=1,\ldots,n} p_i$ den maximalen Profit-Zahlparameter und N die Kodierungslänge der Eingabeinstanz E. Trivialerweise gilt $N > n$, da jede Eingabeinstanz $2n + 1$ Zahlparameter enthält. Wir nehmen im Folgenden oBdA an, dass kein Objekt ein W überschreitendes Gewicht hat. Folglich gilt $P_* \geq p_{max}$. Zweifellos gibt es einen pseudopolynomiellen Algorithmus A, der eine zweidimensionale Tabelle $T = (T[i, j])_{1 \leq i \leq n, 1 \leq j \leq P_n}$ ausfüllt, so dass $T[i, j]$ das minimale Gewicht ist, mit welchem sich ein Profit von mindestens j durch eine geeignete Auswahl aus den Objekten $1, \ldots, i$ erzielen lässt (bzw. $T[i, j] = \infty$, falls $p_1 + \cdots p_i < j$). Eine geeignete Implementierung dieses Algorithmus (dynamisches Programmieren) hat eine Laufzeit, welche polynomiell von N und in p_{max} abhängt. Aus der n-ten Zeile von T kann man leicht den maximal möglichen Profit P_* ablesen. Algorithmus A ist leider nur pseudopolynomiell, weil p_{max} exponentiell groß in Abhängigkeit von N sein kann. Nun ist der Moment gekommen, in dem „Rounding and Scaling“ ins Spiel kommt. Wir skalieren die Eingabeinstanz E (mit eventuell riesenhaftem p_{max}) um den Faktor

$$K := \frac{p_{max}}{(k+1)n} \leq \frac{P_*}{(k+1)n} \tag{16}$$

herunter, indem wir die Parameter p_i durch

$$p_i' := \left\lfloor \frac{p_i}{K} \right\rfloor \leq \left\lfloor \frac{p_{max}}{K} \right\rfloor \leq (k+1)n \tag{17}$$

ersetzen.[34] Wir bezeichnen die neue Eingabeinstanz mit E'. Der PTAA von Ibarra und Kim, im folgenden mit A bezeichnet, geht vor wie folgt:

[33] Genau genommen sollten wir hier einen uniformen Algorithmus A betrachten, der k als zusätzlichen Eingabeparameter erhält und dann vorgeht wie A_k.

[34] Grundsätzlich gilt, dass hohe Werte von K die Laufzeit verbessern, aber die Güte verschlechtern (und umgekehrt für niedrige Werte von K). Die Wahl von K ist somit ein Balance-Akt.

1. Berechne aus Eingabeinstanz E mit den Parametern $p_1, \ldots, p_n; w_1, \ldots, w_n; W$ und dem zusätzlichen Eingabeparameter k die Eingabeinstanz E' mit den gemäß (16),(17) berechneten Parametern $p'_1, \ldots, p'_n$ anstelle von $p_1, \ldots, p_n$.

2. Wende den pseudopolynomiellen Algorithmus A' auf E' an und lies aus der resultierenden Tabelle T eine optimale Lösung $I \subseteq \{1, \ldots, n\}$ von E' ab.

3. Gib I als Lösung für die ursprüngliche Eingabeinstanz E aus.

Satz 8.11 *Der Algorithmus A von Ibarra und Kim ist ein volles Approximationsschema für KNAPSACK.*

Beweis Wir beginnen mit der Zeitanalyse. Die Laufzeit wird dominiert durch die Berechnung der optimalen Lösung I für die Eingabeinstanz E' unter Anwendung des pseudopolynomiellen Algorithmus' A'. Dies kostet größenordnungsmäßig $\text{poly}(N, p_{max}/K) \leq \text{poly}(N, (k+1)n)$ viele Schritte, woraus sich wegen $n \leq N$ eine in N und k polynomiell beschränkte Rechenzeit ergibt.
Abschließend berechnen wir den von I erzielten Profit und vergleichen ihn mit P_*. Bezogen auf Eingabeinstanz E bzw. E' erzielt I den Profit

$$P_E = \sum_{i \in I} p_i \text{ bzw. } P_{E'} = \sum_{i \in I} p'_i = \sum_{i \in I} \left\lfloor \frac{p_i}{K} \right\rfloor \leq \frac{P_E}{K} \ .$$

Es ist nicht schwer **zu zeigen**, dass $P_* \leq KP_{E'} + Kn$. Offensichtlich unterhalten dann P_E, P_* und $P_{E'}$ die Beziehung **Übg.**

$$KP_{E'} \leq P_E \leq P_* \leq KP_{E'} + Kn \leq P_E + Kn \ . \tag{18}$$

Nun ergibt sich die Güte $1 + 1/k$ durch eine einfache Rechnung aus

$$P_E \geq P_* - Kn = P_* - \frac{p_{max}}{k+1} \geq P_* - \frac{P_*}{k+1} \ .$$

•

8.4 NP-harte Approximation

Einige Optimierungsprobleme (wie zum Beispiel KNAPSACK) erlauben (volle) Approximationsschemata, andere (wie zum Beispiel TSP) besitzen nicht einmal konstante Gütegarantien (sofern $P \neq NP$). Wo liegen die theoretischen Grenzen für die Güte von PTAAs für ein gegebenes Optimierungsproblem? In diesem Abschnitt gehen wir dieser Frage nach und lernen ein paar Techniken zum Nachweis der Nichtapproximierbarkeit kennen.

Als erste Basistechnik zum Nachweis der Nichtapproximierbarkeit eines Optimierungsproblems besprechen wir die Herstellung einer „mupltiplikativen Lücke (multiplicative gap)“

in Bezug auf den zu optimierenden Parameter. In Verbindung mit TSP hatten wir diese Technik bereits kennen gelernt: falls $P \neq NP$, dann besitzt TSP keine konstante Gütegarantie. Wie hatten wir dieses Resultat erzielt? Wesentlich bei der verwendeten Reduktion von HC auf TSP war die folgende Eigenschaft:
Wenn der gegebene Graph $G = (V, E)$ einen Hamilton'schen Kreis besitzt, dann erlaubt die von G induzierte Kostenmatrix eine Rundreise mit Kosten $n = |V|$. Gibt es jedoch keinen Hamilton'schen Kreis, dann betragen die Kosten der besten Rundreise mindestens $kn + 1$.
Die Lücke zwischen den Kostenparametern n und $kn + 1$ erlaubt den Schluss:
Wenn ein Algorithmus für TSP die Güte k besitzt, dann kann er (ohne wesentlichen Effizienzverlust) dazu benutzt werden, HC exakt zu lösen.
Die meisten Nichtapproximierbarkeitsresultate beruhen auf einer polynomiellen Reduktion, welche eine multiplikative Lücke (im eben beschriebenen Sinn) herstellt. Einfache Beispiele hierfür ergeben sich durch Optimierungsprobleme, die bereits für eine konstante Kosten- bzw. Profitschranke NP-hart sind. Diese erlauben die Herstellung einer multiplikativen Lücke auf triviale Weise. Wir demonstrieren dies anhand von COLORABILITY:

Bemerkung 8.12 *Falls $P \neq NP$, dann kann kein PTAA für COLORABILITY eine konstante Güte unterhalb von $4/3$ besitzen.*

Beweis Das Graphenfärbungsproblem ist bereits für Kostenschranke 3 (3-COLORABILITY) NP-vollständig. Ein PTAA der Güte $c < 4/3$ kann benutzt werden, um die Frage der 3-Färbbarkeit eines gegebenen Graphen exakt zu lösen. Falls ein solcher Algorithmus existierte, wäre $P = NP$. •

Völlig analog ergibt sich der

Satz 8.13 *Unter der Voraussetzung $P \neq NP$ gilt folgendes. Wenn ein Minimierungspoblem (bzw. Maximierungsproblem) Π bereits mit konstanter Kostenschranke k NP-hart ist, dann kann kein PTAA für Π eine konstante Güte unterhalb von $(k+1)/k$ (bzw. $k/(k-1)$) besitzen.*

Als zweite Basistechnik verwenden wir den Nachweis der starken NP-Härte. Es gelten nämlich folgende Resultate:

Satz 8.14 *Es sei Π ein Optimierungsproblem mit einem natürlich-zahligen Optimierungsparameter und der Eigenschaft, dass der Wert $C_*(E)$ einer optimalen Lösung für Eingabeinstanz E polynomiell in der Kodierungslänge $N(E)$ und in dem maximalen in E vorkommenden Zahlparameter $M(E)$ nach oben beschränkt sind. Dann kann ein volles Approximationsschema für Π in einen pseudopolynomiellen Algorithmus für Π transformiert werden.*

Beweis Wir beschränken uns auf die Betrachtung von einem Minimierungsproblem Π. **Der Beweis für Maximierungsprobleme lässt sich analog führen.** **Übg.**
Nach Voraussetzung gibt es ein Polynom q mit der Eigenschaft

$$C_*(E) < q(N(E), M(E))$$

für alle Eingabeinstanzen E von Π und ein volles Approximationsschema A für Π. Der korrespondierende pseudopolynomielle Algorithmus A' (zur exakten Lösung von Π) geht auf Eingabeinstanz E vor wie folgt:

1. $k := q(N(E), M(E))$.
2. Wende A auf Eingabeinstanz E und Genauigkeitsparameter k an und erhalte eine Lösung mit Kosten $C(E) \leq (1 + 1/k)C_*(E)$.

Da die Laufzeit von A polynomiell in $N(E)$ und k beschränkt ist, ist die Laufzeit von A' polynomiell in $N(E)$ und $M(E)$ beschränkt (pseudopolynomielle Laufzeit). Offensichtlich gilt

$$C(E) - C_*(E) \leq \frac{C_*(E)}{k} = \frac{C_*(E)}{q(N(E), M(E))} < 1 \ .$$

Aus $C(E), C_*(E) \in \mathbb{N}_0$ folgt $C(E) = C_*(E)$. •

Folgerung 8.15 *Falls $P \neq NP$, dann kann ein stark NP-hartes Optimierungsproblem (mit polynomiell in $N(E)$ und $M(E)$ beschränktem Wert einer optimalen Lösung) kein volles Approximationsschema besitzen.*

Gemäß Satz 8.14 lässt sich ein volles Approximationsschema in einen pseudopolynomiellen Algorithmus transformieren. Zumindest für den Spezialfall von KNAPSACK haben Ibarra und Kim die Umkehrung dieses Satzes demonstriert, indem sie (mit der Technik des „Rounding and Scaling“) einen pseudopolynomiellen Algorithmus in ein volles Approximationsschema transformiert haben. Obschon es kein „Metatheorem“ gibt, welches die Konstruktion von Ibarra und Kim auf beliebige pseudopolynomiell lösbare Optimierungsprobleme verallgemeinert, ist die Technik des „Rounding and Scaling“ in vielen Fällen (ähnlich wie bei KNAPSACK) erfolgreich anwendbar.

Approximation bis auf eine „additive Konstante“? Wir haben Approximatiosnalgorithmen kennen gelernt, die das Optimum bis auf einen konstanten Faktor treffen. Stärker wären freilich mathematische Gütegarantien, das Optimum bis auf eine additive Konstante zu treffen. Ist so etwas denkbar? Die Antwort lautet für fast alle natürlichen Optimierungsprobleme NEIN. Wir demonstrieren dies an zwei Beispielen:

Bemerkung 8.16 *Falls $P \neq NP$, dann kann es keinen PTAA A für KNAPSACK mit einer Gütegarantie von der Form*

$$\exists k \in \mathbb{N}, \forall E : \ C_*(E) - C_A(E) \leq k$$

geben, wobei $C_*(E)$ den auf Eingabeinstanz E maximal erzielbaren Profit bezeichnet und $C_A(E)$ den vom Algorithmus A auf E erzielten Profit.

Beweis Angenommen es gäbe einen PTAA A mit einer solchen Gütegarantie. Dann würde der folgende Algorithmus A' KNAPSACK in Polynomialzeit optimal lösen:

1. Es sei E' die aus E resultierende Eingabeinstanz, wenn wir alle Profitparameter um den Faktor $k+1$ hochskalieren (also mit $k+1$ multiplizieren).

2. Wende A auf E' an und erhalte eine Auswahl I der in den Rucksack gepackten Objekte, deren Gesamtprofit vom maximal erzielbaren Profit additiv nur um maximal k abweicht.

3. Gib I als Lösung für die ursprüngliche Eingabeinstanz E aus.

Beachte, dass die Probleme E und E' „isomorph“ sind: Korrespondierende Eingabeparameter und Gesamtprofite von korrespondierenden Lösungen unterscheiden sich immer nur um den Faktor $k+1$. Eine optimale Lösung für E' ist demnach auch eine optimale Lösung für E. Da in E' nur Zahlen auftauchen, die Vielfache von $k+1$ sind, ist „vom Profit einer optimalen Lösung additiv um maximal k abweichen" gleichbedeutend mit „einen optimalen Profit erzielen“. Somit ist A' ein Optimierungsalgorithmus für KNAPSACK (der offensichtlich in Polynomialzeit arbeitet) und es folgt $P = NP$. •

Der Beweis war darum so leicht, weil ein Zahlenproblem vorlag und wir alle Zahlparameter einfach mit einer geeigneten Konstante multiplizieren konnten. Wie sieht es aber bei rein kombinatorischen Problemen aus? Die Antwort lautet GENAU SO, was wir an dem NP-vollständigen Problem PLANAR IS (= Independent Set eingeschränkt auf planare Graphen) illustrieren:

Bemerkung 8.17 *Falls $P \neq NP$, dann kann es keinen PTAA A für PLANAR IS mit einer Gütegarantie von der Form*

$$\exists k \in \mathbb{N}, \forall E: \ P_*(G) - P_A(G) \leq k$$

geben, wobei $P_*(G)$ die maximale Anzahl paarweise unabhängiger Knoten im Eingabegraphen G bezeichnet und $P_A(G)$ die Mächtigkeit der von Algorithmus A konstruierten unabhängigen Menge in G.

Beweis Gehe vor wie bei KNAPSACK, außer dass die Transformation von G in G' die Technik der „kombinatorischen Multiplikation“ verwendet: G' besteht aus $k+1$ disjunkten Kopien von G. Die weitere Argumentation kann aufgebaut werden wie bei dem entsprechenden Nachweis für KNAPSACK. •

Da fast alle natürlichen Optimierungsprobleme die „kombinatorische Multiplikation“ der Eingabeinstanz mit einer Konstanten (sprich: die Vervielfältigung der Eingabeinstanz) erlauben, sind „additive Gütegarantien“ im Prinzip nicht erreichbar.

8.5 Komplexitätsklassen für Optimierungsprobleme

Es bezeichne NPO die Klasse aller Optimierungsprobleme, die durch eine polynomiell verifizierbare Relation und eine polynomiell auswertbare Wertefunktion gegeben sind. APX

bezeichne die Probleme aus NPO, die einen PTAA konstanter Güte besitzen (die sogenannten „approximierbaren“ Probleme). PTAS bezeichne die Probleme aus NPO, die ein Approximationsschema besitzen. Schließlich bezeichne FPTAS die Probleme aus NPO, die ein volles Approximationsschema besitzen (wie zum Beispiel KNAPSACK). Es ergibt sich die Hierarchie

$$FPTAS \subseteq PTAS \subseteq APX \subseteq NPO \ ,$$

die unter der Voraussetzung $P \neq NP$ echt ist:

1. „Independent Set“ eingeschränkt auf planare Graphen gehört zu PTAS (ohne Beweis). Da es sich hier nicht um ein Zahlenproblem und daher automatisch um ein stark NP-vollständiges Problem handelt, kann es aber gemäß Folgerung 8.15 kein volles Approximationsschema geben.

2. Probleme wie „Bin Packing“, „Vertex Cover“, und „Metrisches TSP“ besitzen jeweils ein PTAAs konstanter Güte, aber kein Approximationsschema (was mit Hilfe des berühmten PCP-Theorems gezeigt werden kann).

3. Probleme wie „Set Cover“, „Graphenfärbung“ und CLIQUE besitzen keine PTAAs einer konstanten Güte (was ebenfalls mit dem PCP-Theorem gezeigt werden kann).

Das angesprochene PCP-Theorem und seine Bedeutung für die Theorie der Approximationsalgorithmen werden wir evtl. zu einem späteren Zeitpunkt der Vorlesung besprechen.

Für die Klassen NPO und APX lassen sich mit Hilfe eines geeigneten Reduktionsbegriffes vollständige Probleme (also schwerste Vertreter ihrer Klasse vom Standpunkt der Approximierbarkeit) ausfindig machen. Die uns bekannten Reduktionsbegriffe sind hierfür nicht geeignet. Man braucht vielmehr eine Art „approximationserhaltende“ Reduktion. Populäre approximationserhaltende Reduktionen sind die sogenannten AP- bzw. L-Reduktionen. Wir werden darauf evtl. in einem späteren Stadium der Vorlesung nochmals zurückkommen. Ansonsten sei auf die Webseite

`www.nada.kth.se/ viggo/problemlist/compendium.html`

verwiesen, die von Viggo Kann gepflegt wird. Sie enthält zu einer großen Liste von grundlegenden Optimierungsproblemen die neuesten „guten“ und „schlechten“ Nachrichten (sprich: verbesserte PTAAs bzw. verbesserte Nachweise der inhärenten Nichtapproximierbarkeit). In einigen Fällen konnte die magische Schwelle k_0 für die bestmögliche Güte exakt bestimmt werden.

9 Parametrisierte Komplexität

Es sei daran erinnert, dass ein „Vertex Cover“ eines Graphen $G = (V, E)$ eine Teilmenge $C \subseteq V$ der Knotenmenge ist, die von jeder Kante $e \in E$ mindestens einen Randknoten enthält. Mit Hilfe der Knoten aus C kann man gewissermaßen alle Kanten des Graphen kontrollieren. Es sei weiter daran erinnert, dass eine unabhängige Menge (independent set) eines Graphen $G = (V, E)$ eine Teilmenge $U \subseteq V$ der Knotenmenge ist, die aus paarweise

nicht benachbarten Knoten besteht. Im Gegensatz dazu ist eine Clique in G ist eine Teilmenge $U \subseteq V$ der Knotenmenge, die aus paarweise benachbarten Knoten besteht. Offensichtlich gilt folgendes:

- U ist eine Clique in $G = (V, E)$ gdw U eine unabhängige Menge in $\bar{G} = (V, \bar{E}) = (V, (V \times V) \setminus E)$ ist. Daher ergibt sich mit der Reduktionsabbildung $(G, k) \mapsto (\bar{G}, k)$ die Karp-Reduktion CLIQUE $\leq_{pol}$ IS: eine Clique der Größe k in G wäre nämlich eine unabhängige Menge der Größe k in $\bar{G}$ und umgekehrt. Die NP-Härte von CLIQUE verberbt sich somit auf IS.

- U ist eine unabhängige Menge in $G = (V, E)$ gdw $\bar{U} = V \setminus U$ ein Vertex Cover in G ist. Daher ergibt sich mit der Reduktionsabbildung $(G, k) \mapsto (G, |V| - k)$ die Karp-Reduktion IS $\leq_{pol}$ VC: eine unabhängige Menge der Größe k in G liefert und mit ihrem Komplement nämlich ein Vertex Cover der Größe $|V| - k$ in G und umgekehrt. Die NP-Härte von IS vererbt sich daher auf VC.

In Bezug auf Bezug auf Entscheidungs- oder Optimierungsalgorithmen sind CLIQUE, IS und VC „isomorphe“ Probleme. Wir wissen bereits, dass dies in Bezug auf Approximationsalgorithmen falsch ist, da VC zur Klasse APX der (mit Güte 2) approximierbaren Probleme gehört, wohingegen IS (unter der $P \neq NP$ Voraussetzung) keine Approximationsalgorithmen mit konstanter Güte zulässt. In diesem Abschnitt wird sich ein ähnlicher Gegensatz ergeben: VC gehört zur Klasse FPT der sogenannten „Fest-Parameter behandelbaren“ Probleme, wohingegen „IS (unter der $P \neq NP$ Voraussetzung) nicht zu FPT gehört. Die Aussage über IS gilt entsprechend auch für das folgende Problem:

Dominating Set (DS) Knotenüberwachung durch möglichst wenige Knoten

Eingabe Ein Graph $G = (V, E)$ und eine Kostenschranke k

Frage Kann man mit k Knoten alle Knoten überwachen, d.h., existiert eine Teilmenge $D \subseteq V$ (genannt „dominating set“) der Maximalgröße k, so dass jeder Knoten $v \in V \setminus D$ mindestens einen Nachbarn in D besitzt.

Kommentar Dieses Problem findet Anwendungen bei der Überwachung von Rechnern in einem Rechnernetz.

Oberflächlich betrachtet sind die Probleme VC und DS enge Verwandte, wie durch die folgende polynomielle Reduktion unterstrichen wird:

Lemma 9.1 *VC* $\leq_{pol}$ *DS.*

Beweis Es sei (G, k) mit $G = (V, E)$ eine gegebene Eingabeinstanz von VC. Wir verwenden die (effizient berechenbare) Eingabetransformation $(G, k) \mapsto (G', k + n)$, wobei G' aus G durch die in Abbildung 9 dargestellte lokale Ersetzung hervorgeht.

Wie aus der Abbildung ersichtlich hat G' die Knotenmenge $V_0 \cup V_1 \cup V_2 \cup E_0$ (mit der offensichtlichen Bedeutung von V_0, V_1, V_2, E_0). Die Kantenmenge von G' ist ebenfalls leicht

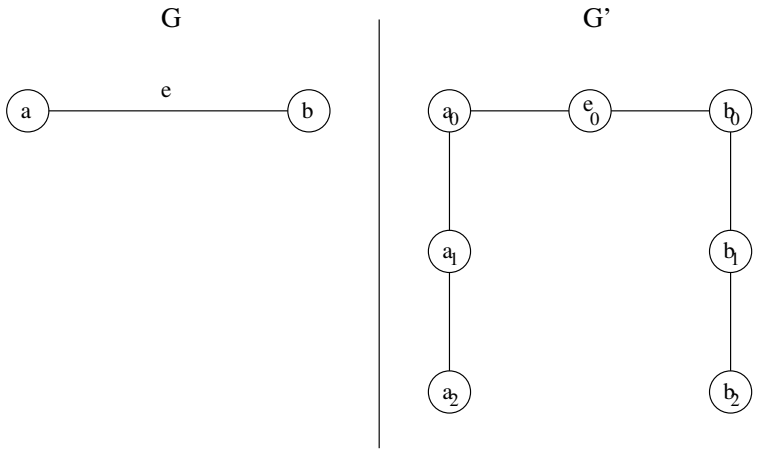


Abbildung 9: Lokale Ersetzung bei der polynomiellen Reduktion von VC auf DS.

aus der Abbildung abzulesen. Wir werden im Folgenden die Knoten aus V mit den Knoten aus V_0 identifizieren. Der Beweis wird abgeschlossen durch die
Behauptung Es gibt genau dann ein Vertex Cover der Maximalgröße k in G, wenn es eine Dominating Set der Maximalgröße $k+n$ in G' gibt.

Nehmen wir an, dass wir in G jede Kante mit Knoten aus $U \subseteq V$ überwachen können und $|U| \leq k$. Dann können wir in G' jeden Knoten mit den Knoten aus $U \cup V_1$ überwachen. Es gilt $|U \cup V_1| = |U| + n \leq k + n$.

Nehmen wir an, dass wir in G' jeden Knoten mit Knoten aus $U' \subseteq V_0 \cup V_1 \cup V_2 \cup E_0$ überwachen können und $|U'| \leq k+n$. Wir können oBdA annehmen, dass in U' keine Knoten aufgenommen werden, die in ihrer „Überwachungspotenz“ von anderen Knoten dominiert werden. Zum Beispiel ist v_1 „potenter“ als v_2, da v_1 die Knoten v_0, v_1, v_2 überwacht und v_2 lediglich die Knoten v_1, v_2. Somit gilt oBdA $U' \cap V_2 = \emptyset$. Da andererseits alle Knoten aus V_2 überwacht werden müssen und v_2 nur von sich selbst oder von v_1 aus überwacht werden kann, gilt $V_1 \subseteq U'$. Durch V_1 werden bereits alle Knoten aus V_0 überwacht, aber noch keine Knoten aus E_0. Das Restproblem ist also auf die Überwachung der Knoten aus E_0 zusammengeschrumpft. Unter diesen Bedingungen gilt für jede Kante $e = \{a, b\}$, dass a_0 (bzw. b_0) „mindestens so potent“ ist wie e_0. Knoten a_0 (bzw. b_0) könnte nämlich potenziell neben e_0 noch weitere Knoten aus E_0 überwachen. Wir können also oBdA annehmen, dass U' die Form $U' = V_1 \cup U_0$ mit $U_0 \subseteq V_0$ hat, wobei die maximal k Knoten aus U_0 die Überwachung von E_0 gewährleisten. Es folgt, dass $U = \{v |\ v_0 \in U_0\}$ ein Vertex Cover der Maximalgröße k in G ist. •

Folgerung 9.2 *DS ist NP-vollständig.*

In Abschnitt 9.1 werden wir die Fest-Parameter Behandelbarkeit von VC herausarbeiten. In Abschnitt 9.2 präsentieren wir eine formale Definition der Klasse FPT und gehen auf den dazu passenden Reduktionsbegriff ein, mit Hilfe dessen die Nichtmitgliedschaft von IS und DS in FPT (unter der $P \neq NP$ Voraussetzung) bewiesen werden könnte.

9.1 Ein Fest-Parameter behandelbares Beispielproblem

Der Star dieses Unterabschnittes ist das Problem VC. Wir wollen im folgenden herausarbeiten, dass VC gegenüber CLIQUE, IS oder DS eine Sonderrolle einnimmt.

Beginnen wir mit einer Gemeinsamkeit der genannten Probleme. Wenn wir die Kostenschranke k zu einer Konstanten einfrieren, dann werden Probleme von der Art CLIQUE, IS, VC oder DS trivialisiert. Wir können nämlich mit *Exhaustive Search* alle $\binom{n}{k}$ Teilmengen $U \subseteq V$ der Größe k der Reihe nach durchmustern und jeweils auf die entsprechende Eigenschaft testen. Falls k eine von $n = |V|$ unabhängige Konstante ist, dann gilt $\binom{n}{k} = O(n^k)$ und wir erhalten eine polynomielle Zeitschranke.

Nun ist n^k zwar ein Polynom, aber $\Omega(n^k)$ Rechenschritte sind für große Konstanten k (selbst bei moderater Eingabelänge n) nicht tolerierbar. Der Ansatz der parametrisierten Komplexität ist zu überprüfen, ob nicht auch eine Zeitschranke der Form $f(k)n^c$ eingehalten werden kann. Hierbei ist f eine beliebige, nur von k (aber nicht von n) abhängige Funktion und $c \geq 1$ ist eine von n und k unabhängige Konstante. Für konstantes k und $c = 1$ hätten wir zum Beispiel eine lineare Laufzeitschranke vorliegen, im Vergleich zu n^k eine radikale Verbesserung. Für VC geht dieser Plan voll auf:

Satz 9.3 (Downey und Fellows, 1992) *VC kann in $O(2^k n)$ Schritten (auf einer Random Access Maschine mit uniformem Kostenmaß) gelöst werden.*

Beweisskizze Wir skizzieren nur die Idee, die den Faktor 2^k ins Spiel bringt. Von jeder Kante $e = \{v, w\}$ enthält ein Vertex Cover den Knoten v oder den Knoten w. Wir können daher einen binären Entscheidungsbaum T aufbauen. Jeder Knoten von T repräsentiert ein partielles Vertex Cover, das anfangs (im Wurzelknoten von T) noch leer ist. Jede neue binäre Entscheidung führt dazu, dass T sich an einem bisherigen Blattknoten z wieder binär verzweigt und das von z repräsentierte partielle Vertex Cover um einen Knoten erweitert wird. Um das Wachstum von T levelweise voranzutreiben, protokollieren wir an jedem Knoten z die korrespondierende partielle Lösung U_z und die Menge der noch unüberwachten Kanten E_z. Wir brechen den Wachstumsprozess von T an einem Blatt z ab, wenn (der Misserfolgsfall) $|U_z| = k, E_z \neq \emptyset$ oder wenn (der Erfolgsfall) $E_z = \emptyset$. T braucht (wegen der Kostenschranke k) nicht über Tiefe k hinaus wachsen und enthält daher $O(2^k)$ Knoten. Falls T zu wachsen aufhört, ohne dass der Erfolgsfall eingetreten ist, dann existiert kein Vertex Cover der Größe k. Im Erfolgsfall hingegen haben wir ein Vertex Cover der Maximalgröße k aufgespürt. Der Overhead, der an jedem Knoten von T zu leisten ist, ist polynomiell in n beschränkt. Eine verfeinerte Implementierung und eine genauere Analyse zeigen, dass eine Laufzeitschranke der Form $O(2^k n)$ eingehalten werden kann. □

Die in diesem Beweis benutzte Datenstruktur trägt den Namen „beschränkter Suchbaum (bounded search tree)“. Eine weitere Schlüsseltechnik ist die sogenannte „Reduktion auf einen Problemkern“, deren Anwendung auf Vertex Cover zu folgendem Resultat führt:

Satz 9.4 (Buss, 1989) *VC kann in $O(k^k + n)$ Schritten (auf einer Random Access Maschine mit uniformem Kostenmaß) gelöst werden.*

Beweisskizze Wir skizzieren nur die Idee, die zu der „additiven“ Form $f(k) + g(n)$ der Zeitschranke führt. Die Reduktion auf den Problemkern von Vertex Cover mit fester Kostenschranke k nutzt die folgenden offensichtlichen Tatsachen aus:

- Jeder Knoten von einem k überschreitenden Grad muss in das „Vertex Cover“ C aufgenommen werden.

- Bei einem Graphen ohne isolierte Knoten ist jedes „vertex cover“ auch eine „dominating set“. Wenn jeder Knoten maximal k Nachbarn hat und es gibt ein „Vertex Cover“ (und somit eine „dominating set“) der Größe k', dann kann es maximal $k'(k + 1)$ Knoten geben.

Dies legt folgenden Algorithmus nahe:

1. Teste, ob es mehr als k Knoten mit jeweils mehr als k Nachbarn gibt. Falls JA, dann kann es kein „Vertex Cover“ der Größe k geben. Falls NEIN, dann nimm die $p \leq k$ Knoten mit jeweils mehr als k Nachbarn in das (anfangs leere) „Vertex Cover“ C auf, reduziere die Kostenschranke auf $k' := k - p$, entferne aus G alle Knoten aus C und die zu ihnen inzidenten Kanten sowie alle resultierenden isolierten Knoten (die zu einer Überwachung der Kanten des Restgraphen nichts beitragen können) und erhalte so den „Restgraphen“ G'.

2. Teste, ob G' mehr als $k'(k + 1)$ Knoten enthält. Falls JA, dann kann es in G' kein „Vertex Cover“ der Größe k' (und somit in G kein „Vertex Cover“ der Größe k) geben. Falls NEIN, dann mache weiter wie folgt.

3. Teste, ob G' (ein Graph mit maximal $k'(k + 1)$ Knoten) ein „Vertex Cover“ C' der Größe k' besitzt. Falls JA, dann gib $C \cup C'$ als „Vertex Cover“ der Größe k von G aus. Falls NEIN, dann existiert kein „Vertex Cover“ der geforderten Größe.

Bei geschickter Implementierung kann die Reduktion auf den Problemkern (hier: die Berechnung des Restgraphen G') in Linearzeit geschehen. Die Berechnung eines „Vertex Cover“ einer Größe von maximal $k' \leq k$ (sofern vorhanden) kann bei geschickter Implementierung in $O(k^k)$ Schritten geschehen. Dies führt zu der Zeitschranke $O(k^k + n)$. □

Folgerung 9.5 (Balasubramanian, Downey, and Fellows, 1992) *VC kann in* $O(2^k k^2 + n)$ *Schritten (auf einer Random Access Maschine mit uniformem Kostenmaß) gelöst werden.*

Beweis Reduziere zuerst auf den Problemkern und wende auf diesen die Methode mit den beschränkten Suchbäumen an. •

Der folgende Satz stammt aus dem Jahre 2006 und liefert eine sehr komfortable Zeitschranke für Vertex Cover:

Satz 9.6 *— ohne Beweis —*
VC kann in $O\left(kn + 1.2738^k\right)$ Schritten (auf einer Random Access Maschine mit uniformem Kostenmaß) gelöst werden.

Diese Schranke ist um eine „Galaxie“ besser als die Zeitschranke n^k eines naiven Verfahrens!

Für welche Probleme außerhalb von P lässt sich ein Parameter k dingfest machen, so dass Lösungsalgorithmen mit einer Zeitschranke der Form $f(k)n^c$ existieren? Für welche Probleme existieren solche Parameter k nicht? Um diese Frage korrekt zu beantworten bedarf es einiger abstrakter Definitionen, die den Kern der Fest-Parameter Behandelbarkeit herausarbeiten.

9.2 Probleme innerhalb und außerhalb FPT

Definition 9.7 *Eine* Parametrisierung *von Σ^* ist eine in Polynomialzeit berechenbare Abbildung $\kappa : \Sigma^* \to \mathbb{N}$.*

Definition 9.8 *Ein* parametrisiertes Entscheidungsproblem *(über dem Grundalphabet Σ) ist gegeben durch eine formale Sprache $L \subseteq \Sigma^*$ und eine Parametrisierung κ von Σ^*.*

Beispiel 9.9 *Es bezeichne $\langle G, k\rangle \in \Sigma^*$ das Codewort (gemäß einer vorgegebenen natürlichen Kodierung) für ein Paar (G, k) mit einem Graphen $G = (V, E)$ und einer natürlichen Zahl k. Wir setzen dabei der Einfachheit halber voraus, dass jeder String aus Σ^* als Codewort für ein Paar (G, k) interpretiert wird. Es sei L die Menge aller Codewörter für Paare (G, k) mit: es gibt eine Auswahl $C \subseteq V$ von k Knoten aus V, so dass jede Kante von G mindestens einen Randknoten in C hat. Weiter sei $\kappa(\langle G, k\rangle) = k$. Dann ist (L, κ) ein parametrisiertes Entscheidungsproblem (und offensichtlich eine parametrisierte Variante von „Vertex Cover“).*

Definition 9.10 *Ein* fpt-Algorithmus *für ein parametrisiertes Entscheidungsproblem (L, κ) ist ein Algorithmus, der das Wortproblem „$x \in L$?“ korrekt entscheidet und dessen Laufzeit durch eine Funktion der Form $p(|x|) \cdot f(\kappa(x))$, mit einem Polynom p und einer berechenbaren Funktion f, beschränkt werden kann.*

Definition 9.11 *FPT ist definiert als die Klasse aller parametrisierter Entscheidungsprobleme, welche einen fpt-Algorithmus besitzen.*

Bemerkung 9.12 *Die Definition von FTP ist robust gegen einige Modifikationen. Zum Beispiel ändert sich die Problemklasse FPT nicht, wenn wir Laufzeitschranken der Form $f(k) + n^c$ (anstelle $f(k) \cdot n^c$) fordern (wie 1997 von Cai, Chen, Downey und Fellows gezeigt wurde).*

In der Vorlesung wurde gezeigt:

Bemerkung 9.13 *Vertex Cover, mit der in Beispiel 9.9 beschriebenen Parametrisierung, liegt in FPT.*

Obwohl wir im Rahmen dieser Vorlesung nicht näher darauf eingehen können, sei bemerkt, dass FPT eine sehr reichhaltige Klasse ist, und dass die Nachweise zur Mitgliedschaft in FPT einige grundlegende Methoden und Tricks zum Algorithmenentwurf hervorgebracht haben. Andererseits gibt es auch eine breite Palette von Problemen, die (vermutlich) nicht zu FPT gehören. In diese Kategorie der *Fest-Parameter harten* Probleme gehören zum Beispiel CLIQUE, IS und DS. Die Vermutung der Fest-Parameter Härte eines Problems wird wieder mit einem geeigneten Reduktionsbegriff gestützt. Anders als bei der Klasse *NPC* haben sich aber mehrere harte Klassen gebildet, die oberhalb von FPT eine Hierarchie bilden. Auch hierauf können wir im Rahmen dieser Vorlesung nicht näher eingehen. Wir definieren lediglich den Reduktionstypus, den man in der Fest-Parameter Komplexitätstheorie zugrunde legen sollte:

Definition 9.14 *Es seien (L, κ) und (L', κ') zwei parametrisierte Entscheidungsprobleme. Wir sagen (L, κ) ist* FPT-reduzierbar *auf (L', κ'), notiert als $(L, \kappa) \leq_{fpt} (L', \kappa')$, falls eine Abbildung $R : \Sigma^* \to \Sigma^*$, berechenbare Funktionen f, g und ein Polynom p existieren mit:*

1. *Für alle $x \in \Sigma^*$ gilt: $x \in L \Leftrightarrow R(x) \in L'$.*
2. *$R(x)$ ist in Zeit $p(|x|) \cdot f(\kappa(x))$ berechenbar.*
3. *$\kappa'(f(x)) \leq g(\kappa(x))$.*

Zum Beispiel ist die polynomielle Reduktion von CLIQUE nach IS (Übergang zum Graphen mit der gleichen Knoten- aber komplementären Kantenmenge) eine Fest-Parameter Transformation (mit $g(k) = k' = k$). Die polynomiellen Reduktionen von IS nach VC (mit $k' = n - k$) und von VC nach DS (mit $k' = n + k$) hingegen sind keine Fest-Parameter Transformationen, da Fest-Parameter k' jeweils eine (verbotene) Abhängigkeit von n aufweist. Offensichtlich gilt:

Bemerkung 9.15 *Aus $(L, \kappa) \leq_{fpt} (L', \kappa')$ und $(L', \kappa') \in FPT$ folgt $(L, \kappa) \in FPT$.*

Beweis Es sei A' ein fpt-Algorithmus mit Zeitschranke $p'(|x|) \cdot f'(\kappa'(x))$ für ein Polynom p' und eine berechenbare Funktion f'. Wir dürfen annehmen, dass p' von der Form $p'(n) = n^c$ für eine Konstante c und dass f' eine monoton wachsende Funktion ist. Wir erhalten einen fpt-Algorithmus A' für (L, κ) wie folgt:

1. Transformiere Eingabe x in $x' = R(x)$. Aufwand: $p(|x|) \cdot f(\kappa(x))$ Rechenschritte.
2. Wende A' auf Eingabe x' an. Aufwand: $p'(|x'|) \cdot f'(\kappa'(x'))$ Rechenschritte.
3. Akzeptiere x genau dann, wenn A' den String x' akzeptiert.

Offensichtlich gilt:

$$|x'| \leq p(|x|) \cdot f(\kappa(x)) \ , \ p'(|x'|) \leq p(|x|)^c \cdot (f(\kappa(x))^c \ , \ f'(\kappa'(x')) \leq f'(g(\kappa(x))) \ .$$

Die aus dieser Diskussion resultierende Zeitschranke weist A' als fpt-Algorithmus aus. •

Die Kehrseite von Bemerkung 9.15 ist natürlich, dass sich eine etwaige Fest-Parameter Härte von L auf L' vererben würde.

Wir bemerken abschließend, dass die Hierarchie, die von FPT und den diversen Klassen von Fest-Parameter harten Problemen gebildet wird, „windschief“ zur polynomiellen Platz-Zeit Hierarchie liegt. Probleme aus *NP* können bereits Fest-Parameter hart sein. Andererseits gehören sogar einige PSPACE-vollständige Probleme zur Klasse FPT.

10 Zwischenbilanz zum Thema „P versus NP“

Bei großen Softwareprojekten entstehen nach einer Strukturierungs- und Modellierungsphase meist klar umrissene Teilprobleme. Ein Lernziel beim Studium der *Effizienten Algorithmen* und der *Komplexitätstheorie* ist

- zu erkennen, dass ein Teilproblem mit einem effizienten Algorithmus lösbar ist,
- zu erkennen, dass dies nicht der Fall ist (zum Beispiel aufgrund der *NP*-Härte des Problems).

NP-harte Probleme verschwinden nicht, wenn wir sie als *NP*-hart erkannt haben. Es stellt sich also weiterhin die Frage, wie mit harten Problemen umzugehen ist. In den voran gegangenen Abschnitten haben wir ein paar Methoden zum Umgang mit NP-harten Problemen (sowie die theoretischen Grenzen dieser Methoden) kennen gelernt:

Einschränkung des Problems Suche nach (praktisch relevanten) und effizient lösbaren Teilproblemen eines NP-vollständigen Problems.
Zum Beispiel sind 2-SAT (im Gegensatz zu 3-SAT) und 2-dimensionales Matching (im Gegensatz zum 3-dimensionalen Matching) effizient lösbar. PRECEDENCE CONSTRAINED SCHEDULING wird effizient lösbar, wenn wir nur Bäume (Sortier- oder Montagebäume) als partielle Relation auf der Menge der Aufgaben zulassen. Manche Zahlenprobleme (wie PARTITION oder KNAPSACK) sind pseudopolynomiell lösbar und somit polynomiell lösbar im Falle „kleiner“ Zahlparameter in der Eingabe. Die Grenzen dieses Ansatzes ergeben sich durch extrem eingeschränkte aber immer noch NP-harte Teilprobleme bzw., im Falle der Zahlprobleme, durch das Phänomen der starken NP-Vollständigkeit.

Aufgabe des Anspruches der Optimalität Versuche ein NP-hartes Optimierungsproblem mit einem Approximationsalgorithmus zu lösen, welcher in Polynomialzeit eine „gute“ (aber i.A. nicht optimale) Lösung liefert.
Die Grenzen dieses Ansatzes (den wir am Beispiel der Probleme TSP und KNAPSACK diskutiert haben) ergeben sich durch das Phänomen der NP-harten Approximation.

Fest-Parameter Behandelbarkeit Versuche aus der Eingabe einen sogenannten „Fest-Parameter“ k zu isolieren, der für die NP-Härte des Problems verantwortlich ist und suche einen Algorithmus mit einer Zeitschranke der Form $f(k)n^c$.
Diesen Ansatz haben wir am Beispiel von Vertex Cover diskutiert. Seine Grenzen findet er im Phänomen der Fest-Parameter Härte.

Ein anderer (bisher von uns nicht diskutierter) Ansatz besteht darin ein mächtigeres Maschinenmodell als die Deterministische Turing Maschine (DTM) zu verwenden. Wir diskutieren ein paar konkrete Vorschläge:

Probabilistisches Maschinenmodell Statte M mit einer *perfekten Münze* aus. M hat pro Schritt zwei Handlungsalternativen und wählt durch Münzwurf eine davon zufällig aus. Dieser Ansatz führt zum Modell der *Probabilistischen Turing Maschine (PTM)*. Um das Mitgliedschaftsproblem für Sprache L zu lösen, muss eine PTM M Eingabewörter $x \in L$ mit “hinreichend hoher” Wahrscheinlichkeit akzeptieren und Eingabewörter $x \notin L$ mit “hinreichend hoher” Wahrscheinlichkeit verwerfen. Wir gehen in einem späteren Stadium der Vorlesung noch genauer auf PTMs und die entsprechenden Komplexitätsklassen ein.

Parallelrechnermodell Verwende p parallel arbeitende Prozessoren, die miteinander kommunizieren können, um kooperativ ein gemeinsames Problem zu lösen. In diese Rubrik fallen neben der *Parallel Random Access Machine (PRAM)* und den diversen *Rechnernetzwerken* auch die sogenannten (in den letzten zwei Dekaden in Mode gekommenen) *DNA-Rechner*. Ein *NP*-hartes Problem lässt sich allerdings auch von einem Parallelrechner nicht zufriedenstellend lösen, da jeder Faktor an Zeitgewinn mit einem mindestens ebenso großen Faktor an zusätzlichem Hardwareaufwand bezahlt wird. Wir können im Rahmen dieser Vorlesung auf Parallelrechner nicht näher eingehen.

Qauntenrechner Die DTM und dazu verwandte Modelle sind mathematische Abstraktionen von physikalischen Rechnern, die den Gesetzen der klassischen Mechanik gehorchen. In den letzten zwei Dekaden wurde eine neue Generation von Rechnern vorgeschlagen, die den Gesetzen der Quantenmechanik gehorchen. Ein Quantenrechner ist zu einem bestimmten Zeitpunkt nicht in **einer** Konfiguration, sondern in einer **Überlagerung von vielen** Konfigurationen. Erst durch externe Beobachtung kollabiert die Überlagerung zu einer einzigen Konfiguration (gemäß einer zur Überlagerung assoziierten Wahrscheinlichkeitsverteilung). Quantenrechner sind vermutlich wesentlich mächtiger als DTMs. Zum Beispiel konnte Peter Shor nachweisen, dass das Faktorisierungsproblem (Berechnung der Primfaktorzerlegung einer gegebenen ganzen Zahl) und das Diskrete-Logarithmus-Problem auf Quantenrechnern effizient lösbar sind. Auf der vermeintlichen Härte dieser Probleme beruhen die meisten aktuell verwendeten asymmetrischen Kryptosysteme. Es ist zur Zeit noch völlig unklar, ob Quantenrechner physikalisch realisiert werden können. Im Rahmen dieser Vorlesung werden wir auf Quantenrechner nicht weiter eingehen können.

Ein weiterer Ansatz, den wir im Rahmen dieser Vorlesung nicht behandeln können ist die

Average-Case Analyse Average-case Analyse steht im Gegensatz zur worst-case Analyse. Hier wird die Forderung aufgegeben, auf **allen** Eingabeinstanzen erfolgreich sein zu müssen. Man verwendet ein Wahrscheinlichkeitsmaß auf den Eingabeinstanzen und ist zufrieden, wenn die mittlere Laufzeit polynomiell ist. Ein Argument **für average-case Analyse** (oder zumindest gegen worst-case Analyse) ist, dass die von polynomiellen Reduktionen produzierten Eingabeinstanzen i.A. kunstvoll und bizarr anmuten. Es könnte also sein, dass genau der Eingabetypus, welcher die *NP*-Härte bezeugt, in Anwendungen nicht (oder selten) anzutreffen ist. Ein Argument **gegen average-case Analyse** ist, dass die Auswahl eines analysierbaren Wahrscheinlichkeitsmaßes meist willkürlich und durch die Anwendung nicht zu rechtfertigen ist.

Smoothed Analysis Es handelt sich um eine Mischform von average-case und worst-case Analyse, bei welcher gemittelt wird über Eingabeinstanzen, die aus leichten zufälligen Abänderungen von worst-case Eingabeinstanzen entstehen. Zum Beispiel konnte gezeigt werden, dass der Simplex-Algorithmus zum Optimieren unter linearen Randbedingungen zwar im worst-case eine exponentielle Laufzeit hat. Die „Smoothed Analysis“ jedoch führt im statistischen Mittel zu einer polynomiellen Zeitschranke.

Schließlich gibt es den sogenannten „heuristischen Ansatz“ zum Lösen von NP-harten Optimierungsproblemen:

Heuristiken und Metaheuristiken Eine *Heuristik* ist ein Algorithmus, der in der Praxis ganz gut zu funktionieren scheint, ohne dass dies durch eine mathematisch beweisbare Qualitätsgarantie abgesichert wäre. *Metaheuristiken* sind vielseitig verwendbare Heuristiken, die durch geeignete Adjustierung einiger programmierbarer Parameter auf die Lösung von konkreten Problemen spezialisiert werden können. Daneben gibt es ad-hoc Heuristiken, die für ein konkretes Problem zusammengeschustert wurden. Beispiele für Metaheuristiken sind:

- Branch and Bound
- Lokale Optimierung
- Simulated Annealing
- Tabu Search
- Neuronale Netze oder Hopfield Netze
- Genetische Algorithmen
- Great Deluge (große Überschwemmung)
- Roaming Ants (umherstreifende Ameisen)

Auch diesen Ansatz werden wir aus Zeitgründen nicht weiter verfolgen können.

11 Die Struktur von NP

Falls $P = NP$, dann gibt es zur Struktur von NP nicht viel zu berichten. Alle Betrachtungen dieses Abschnittes basieren auf der

Vermutung 1 $P \neq NP$.

Wir verwenden im Folgenden Vermutung 1 als Voraussetzung, ohne dies jedesmal explizt kennntlich zu machen. Wir machen also öfter eine Aussage A, die streng genommen die Form: $P \neq NP \Rightarrow A$ haben müsste.

Es sei daran erinnert, dass NPC die Klasse der NP-vollständigen Probleme bezeichnet. Die Klasse

$$NPI := NP \setminus (P \cup NPC)$$

bezeichne die Klasse der Sprachen aus NP, die einerseits nicht zu P gehören, andererseits aber auch nicht NP-vollständig sind.

Aus der NP-Vollständigkeitstheorie folgt direkt, dass die Existenz einer NP-vollständigen Sprache in P die Gleichheit von P und NP zur Folge hätte. Vermutung 1 zugrunde gelegt, müssen also NPC und P disjunkte Teilmengen von NP sein. Mit der Definition von NPI ergibt sich eine erste Strukturaussage:

- NP zerfällt in die paarweise disjunkten Klassen P, NPC und NPI.

Diese Einsicht ist in Abbildung 10 visualisiert.

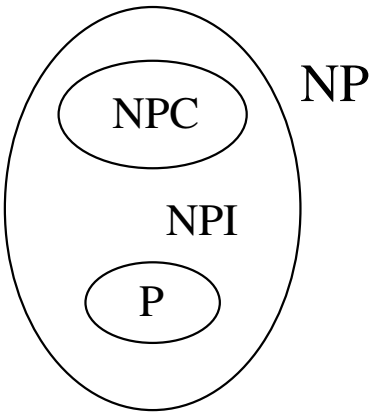


Abbildung 10: Die Struktur von NP.

Wir kennen bereits zahlreiche Sprachen (Probleme) in P bzw. NPC. Wie schaut es aber mit NPI aus: könnte es sein, dass NPI leer ist? Dies würde bedeuten, dass jede Sprache aus $NP \setminus P$ bereits NP-vollständig sein müsste. Der folgende Satz schließt diese Möglichkeit aus:

Satz 11.1 (Ladner, 1975) $NPI \neq \emptyset$.

Beweis Der Beweis benutzt die Technik der Diagonalisierung und basiert auf folgenden Tatsachen:

- Die Binärstrings bezitzen die „natürliche“ Aufzählung $\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots$. Der i-te Binärstring dieser Aufzählung wird im Folgenden als $\alpha(i)$ notiert.

- Jede Boolesche CNF-Formel kann auf natürliche Weise über dem Binäralphabet $\{0, 1\}$ kodiert werden. Binärstrings, die keine ordentlichen CNF-Codewörter sind, können mit einer Formel für die Nullkonstante identifiziert werden. Auf diese Weise repräsentiert jeder Binärstring α eine CNF-Formel F_α.

- Wie wir aus der Vorlesung „Theoretische Informatik“ über die Universelle Turing-Maschine (UTM) wissen, kann jede DTM auf natürliche Weise durch einen Binärstring kodiert werden. Strings, die keine ordentlichen DTM-Codewörter sind, können mit einem Akzeptor für die leere Menge identifiziert werden. Auf diese Weise repräsentiert jeder Binärstring α eine DTM M_α. T Rechenschritte der DTM M_α sind durch $O(|\alpha| T \log T)$ Rechenschritte einer k-Band UTM simulierbar.

- Der natürliche Code α einer DTM kann durch Anhängen von 01^n mit einer beliebig großen Anzahl n von Einsen „gepolstert“ werden. Dann hat jede DTM unendlich viele Codewörter. Die Rechenzeit der UTM hängt aber weiterhin nur von der Länge der nicht-gepolsterten Kodierung ab.

SAT bezeichne die Menge der Binärstrings, die eine erfüllbare CNF-Formel repräsentieren. Für jede Funktion $H : \mathbb{N} \to \mathbb{N}$ definieren wir die folgende Variante von SAT:

$$\mathrm{SAT}_H := \{\alpha 01^{n^{H(n)}} \mid \alpha \in \mathrm{SAT} \wedge n = |\alpha|\}$$

Für die folgende induktiv definierte (und trickreiche) Wahl der Funktion H gilt (wie wir unter der Voraussetzung P $\neq$ NP noch zeigen werden): $\mathrm{SAT}_H \in \mathit{NPI}$:

- Für $n \leq 4$ setze $H(n) = 1$.

- Für $n \geq 5$ setze $H(n)$ auf die kleinste Zahl $i < \log\log n$ mit folgender Eigenschaft: Für alle Binärstrings α mit $|\alpha| < \log n$ gibt $M_{\alpha(i)}$ nach maximal $i|\alpha|^i$ Schritten das Indikatorbit aus, welches anzeigt, ob $\alpha \in \mathrm{SAT}_H$. Falls kein in diesem Sinne „erfolgreicher“ Index i existiert, dann setze behelfsweise $H(n) = \lceil \log\log n \rceil$.

Es ist offensichtlich, dass $H(n)$ eine in n monoton wachsende Funktion ist. Somit existiert der Grenzwert $\lim_{n\to\infty} H(n)$ (evtl. ∞). Die zentralen Bausteine für den Rest des Beweises sind die folgenden weiteren Eigenschaften der Funktion H:

Behauptung 1: $H(n)$ ist in poly(n) Schritten berechenbar.

Behauptung 2: $\mathrm{SAT}_H \in P \Rightarrow \lim_{n\to\infty} H(n) < \infty$.

Behauptung 3: $\mathrm{SAT}_H \notin P \Rightarrow \lim_{n\to\infty} H(n) = \infty$.

Wir demonstrieren zunächst, wie sich mit Hilfe der drei Behauptungen $\mathrm{SAT}_H \in NPI$ ableiten lässt. Aus Behauptung 1 ergibt sich $\mathrm{SAT}_H \in NP$. Wir haben auszuschließen, dass SAT_H zu P oder zu NPC gehört. Wir zeigen, dass die Annahme einer Mitgliedschaft in P oder NPC jeweils zu einem Widerspruch zur $P \neq NP$ Voraussetzung führt. Angenommen, $\mathrm{SAT}_H \in P$. Dann hat $H(n)$ gemäß Behauptung 2 einen endlichen Grenzwert, sagen wir C. Das bedeutet aber, dass eine CNF-Formel F_α der Kodierungslänge $n = |\alpha|$ nur mit maximal n^C Einsen (polynomiell in n viele) ausgepolstert wird. Somit kann ein Polynomialzeitalgorithmus für SAT_H benutzt werden, um SAT in Polynomialzeit zu lösen. Daraus ergäbe sich $P = NP$ im Widerspruch zur $P \neq NP$ Voraussetzung. Nehmen wir nun an, dass $\mathrm{SAT}_H \in NPC$. Dies impliziert $\mathrm{SAT}_H \notin P$, was gemäß Behauptung 3 wiederum $\lim_{n\to\infty} H(n) = \infty$ impliziert. Eine CNF-Formel F_β der Kodierungslänge $|\beta|$ wird demnach im Rahmen der Sprache SAT_H mit superpolynomiell vielen Einsen ausgepolstert. Wegen der (angenommenen) NP-Härte von SAT_H (und wegen der Polynomialzeitberechenbarkeit von H) muss eine in Polynomialzeit berechenbare Reduktionsabbildung der Form $\alpha \mapsto \beta 01^{m^{H(m)}}$ mit $m = |\beta|$ existieren, die SAT auf SAT_H reduziert. Das Anhängen von superpolynomiell in m vielen Einsen kann nur dann in $\mathrm{poly}(|\alpha|)$ Schritten geschehen, wenn β signifikant kürzer als α ist. Zum Beispiel kann $|\beta| \leq \sqrt{|\alpha|}$ gefolgert werden. Iteratives Anwenden der Reduktionsabbildung führt dann aber zu einem Polynomialzeitalgorithmus für SAT im Widerspruch zur $P \neq NP$ Voraussetzung. Die Schlussfolgerung aus dieser Diskussion ist, dass $\mathrm{SAT}_H \in NPI$. Jedoch haben wir abschließend noch die drei Behauptungen zu beweisen.

Beweis von Behauptung 1: Es gibt $\log\log n$ Kandidatenzahlen i und weniger als $2n$ Kandidatenstrings α mit $|\alpha| < \log n$ (da die Anzahl der Binärstrings einer Maximallänge k gleich $2^{k+1} - 1 < 2^{k+1}$ ist). Für jedes fest gewählte Paar (i, α) ist im Wesentlichen folgendes zu tun:

- Verwende die UTM, um die Rechnung von $M_{\alpha(i)}$ auf Eingabewort α für $i|\alpha|^i < \log\log(n)\log(n)^{\log\log(n)}$ Schritte zu simulieren.

Für jeden fest gewählten Binärstring α ist zu testen, ob $\alpha \in \mathrm{SAT}_H$. Dies kann wie folgt geschehen:

- Teste (Test 1), ob α das Symbol 0 enthält und, gegebenenfalls, zerlege α gemäß $\alpha = \beta 01^m$.
- Interpretiere β als CNF-Formel F_β und teste (Test 2) diese auf Erfüllbarkeit (wegen $|\beta| < |\alpha| \leq \log n$ in $\mathrm{poly}(n)$ Schritten machbar).
- Berechne rekursiv[35] $H(|\beta|)$ und teste (Test 3), ob $m = |\beta|^{H(|\beta|)}$.
- Setze das Indikatorbit für „$\alpha \in \mathrm{SAT}_H$?“ auf 1, falls α alle drei Tests besteht; andernfalls setze das Indikatorbit auf 0.

[35] Bei der genauen Zeitanalyse muss ausgenutzt werden, dass der Parameter β, auf welchem der rekursive Aufruf erfolgt, wesentlich kleiner ist als der Parameter n, mit dem wir gestartet waren.

Um das kleinste „erfolgreiche“ i zu finden, müssen die Indikatorbits zu den α-Strings mit den Indikatorbits verglichen werden, welche die DTMs $M_{\alpha(i)}$ angesetzt auf α produzieren. Bei vernünftiger Implementierung des skizzierten Algorithmus zur Berechnung von $H(n)$ ergibt sich eine polynomiell in n beschränkte Laufzeit. (Wir verzichten an dieser Stelle auf die Angabe weiterer Details.)

Beweis von Behauptung 2: Unter der Annahme $\mathrm{SAT}_H \in P$ existiert eine DTM M, welche die Mitgliedschaft in SAT_H korrekt in Polynomialzeit entscheidet. Für eine hinreichend groß gewählte Konstante j vollzieht M auf Eingabewörtern der Länge n maximal jn^j Rechenschritte. Da M (wie jede andere DTM auch) unendlich viele Kodierungsstrings besitzt, existiert ein Index $k \geq j$ mit $M = M_{\alpha(k)}$. Aus der Definition von H ergibt sich dann aber, dass die Funktion $H(n)$ durch k nach oben beschränkt ist.

Beweis von Behauptung 3: Wir führen einen indirekten Beweis und zeigen, dass ein endlicher Grenzwert k von $H(n)$ impliziert, dass $\mathrm{SAT}_H \in P$. Grenzwert k bedeutet, dass es einen Index n_0 gibt, so dass $H(n) = k$ für alle $n \geq n_0$. Aus der Definition von H ergibt sich dann, dass $M_{\alpha(k)}$ das Mitgliedschaftsproblem für die Sprache SAT_H mit Laufzeitschranke kn^k löst (wobei die endlichen vielen Strings einer Länge unterhalb von n_0 von der endlichen Kontrolle übernommen werden können).

Damit ist der Beweis des Satzes abgeschlossen. •

Wir wollen nun die Stuktur von *NP* differenzierter untersuchen.

Definition 11.2 *Wir sagen zwei Sprachen L_1 und L_2 sind* polynomiell äquivalent*, wenn sie wechselseitig aufeinander polynomiell reduzierbar sind. In Zeichen:*

$$L_1 \overset{pol}{\sim} L_2 :\Leftrightarrow L_1 \leq_{pol} L_2 \text{ und } L_2 \leq_{pol} L_1 \ .$$

Diese Relation ist (offensichtlich) eine Äquivalenzrelation. Eine Äquivalenzklasse bezüglich „$\overset{pol}{\sim}$“ heiße Schwierigkeitsgrad *bezüglich „$\overset{pol}{\sim}$“ oder kurz* p-Grad.

Wenn wir polynomielle Unterschiede zwischen Laufzeiten ignorieren, können wir zwei polynomiell äquivalente Sprachen als (im Wesentlichen) gleich schwer betrachten. Die folgende Definition liefert eine partielle Ordnung auf den p-Graden;

Definition 11.3 *Wir sagen L_1 ist* leichter *als L_2 bezüglich polynomieller Reduktion, wenn zwar L_1 polynomiell reduzierbar auf L_2 ist, aber nicht umgekehrt. In Zeichen (mit „$\neg$“ für „logische Negation“):*

$$L_1 <_{pol} L_2 :\Leftrightarrow L_1 \leq_{pol} L_2 \text{ und } \neg(L_2 \leq_{pol} L_1) \ .$$

Wir sagen der p-Grad von L_1 ist leichter *als der p-Grad von L_2, wenn L_1 leichter als L_2 bezüglich polynomieller Reduktion ist.*

Es ist leicht einzusehen, dass $<_{pol}$ auf Sprachen eine irreflexive, asymmetrische partielle Ordnung bildet, die auch auf p-Graden wohldefiniert (also nicht abhängig von der Auswahl des Repräsentanten eines p-Grades) ist. In diesem Lichte besteht NP aus einer Hierarchie von p-Graden. Eine Teilhierarchie ist in Abbildung 11 zu sehen. Die folgenden Aussagen sind nicht schwer **zu zeigen**. **Übg.**

- Die leere Menge und Σ^* bilden zwei triviale p-Grade, angesiedelt auf der untersten Stufe der Hierarchie.
- $P \setminus \{\emptyset, \Sigma^*\}$, also die restlichen Sprachen aus P, bilden den nächst-einfachsten p-Grad, den wir als P-Grad bezeichnen.
- NPC, also die NP-vollständigen Probleme, bilden den schwersten p-Grad in NP.
- Da NPI nicht leer ist, können wir willkürlich eine Sprache L aus NPI rausgreifen. Ihr p-Grad muss echt zwischen dem P-Grad und NPC liegen.

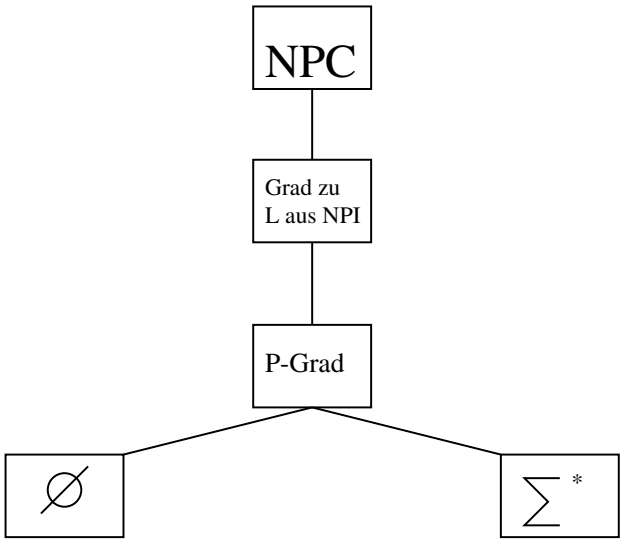


Abbildung 11: Eine Teilhierarchie der p-Grade.

Es stellt sich nun die Frage, ob die Struktur der p-Grade zwischen P und NPC nicht wesentlich komplexer ist, als dies durch Abbildung 11 suggeriert wird. Dies ist in der Tat der Fall, wie folgende Sätze belegen:

Satz 11.4 (Ladner, 1975) *1. Es gibt unendlich lange Ketten von p-Graden zwischen P und NPC.*

2. Es gibt unvergleichbare p-Grade zwischen P und NPC.

Die erste Aussage bedeutet, dass es Sprachen $L_0, L_1, L_2, \ldots, L_* \in NP$ gibt, so dass $L_0 \in P$, $L_* \in NPC$ und

$$L_0 <_{pol} L_1 <_{pol} L_2 <_{pol} \cdots <_{pol} L_* \ .$$

Die Sprachen $L_1, L_2, \ldots$ spannen eine unendliche Kette zwischen dem P-Grad und NPC auf. Die zweite Ausage macht deutlich, dass $<_{pol}$ nur eine partielle, nicht aber totale, Ordnung auf den p-Graden liefert. Der Beweis dieses Satzes, den wir in unserer Vorlesung auslassen, basiert (wie schon der Beweis von Satz 11.1) auf der Diagonalisierungstechnik.

Wir wollen im Folgenden das Bild durch die Betrachtung von co-NP abrunden. Die allgemeine Definition einer *Komplementärklasse* zu einer Sprachklasse $\mathcal{C}$ ist wie folgt:

$$\text{co-}\mathcal{C} := \{\bar{L} : L \in \mathcal{C}\} \ .$$

Wir machen folgende (offensichtlichen) Beobachtungen:

1. $\mathcal{C} = \text{co-}\mathcal{C}$ gilt genau dann, wenn $\mathcal{C}$ abgeschlossen unter Komplementbildung ist.
2. Aus $\mathcal{C} \subseteq \mathcal{C}'$ folgt, dass co-$\mathcal{C} \subseteq$ co-$\mathcal{C}'$.
3. co-co-$\mathcal{C} = \mathcal{C}$.

Wir formulieren jetzt die Vermutung, dass NP nicht unter Komplementbildung abgeschlossen ist:

Vermutung 2 *$NP \neq$ co-NP.*

Die folgende Überlegung zeigt, dass Vermutung 2 stärker ist als Vermutung 1:

Bemerkung 11.5 *$NP \neq$ co-$NP \Rightarrow P \neq NP$.*

Beweis Wir führen einen indirekten Beweis, d.h., wir beweisen (unter Ausnutzung von $P =$ co-P) die Aussage $P = NP \Rightarrow NP =$ co-NP:

$$P = NP \Rightarrow \text{co-}NP = \text{co-}P = P = NP \ .$$

•

co-NP ist eine zu NP duale Klasse. Strukturen in NP haben ihr duales Pendant in co-NP. Eine erste Illustration dieses Sachverhaltes liefert

Lemma 11.6 *co-NPC, also die Komplementärklasse der NP-vollständigen Sprachen, stimmt überein mit der Klasse der co-NP-vollständigen Sprachen (also der Klasse aller Sprachen aus co-NP, auf welche jede Sprache aus co-NP polynomiell reduzierbar ist).*

Beweis Wir nutzen die Implikation $L_1 \leq_{pol} L_2 \Rightarrow \bar{L}_1 \leq_{pol} \bar{L}_2$ aus. Somit sind die folgenden Aussagen äquivalent:

$$\begin{aligned} L_0 \in \text{co-}NPC &\Leftrightarrow \bar{L}_0 \in NPC \\ &\Leftrightarrow (\bar{L}_0 \in NP) \wedge (\forall L \in NP : L \leq_{pol} \bar{L}_0) \\ &\Leftrightarrow (L_0 \in \text{co-}NP) \wedge (\forall L \in NP : \bar{L} \leq_{pol} L_0) \\ &\Leftrightarrow (L_0 \in \text{co-}NP) \wedge (\forall L \in \text{co-}NP : L \leq_{pol} L_0) \\ &\Leftrightarrow L_0 \text{ ist co-}NP\text{-vollständig.} \end{aligned}$$

•

Definition 11.7 *Eine Beziehung $\leq_{pol}$ zwischen formalen Sprachen heißt* abstrakte polynomielle Reduktion*, wenn $\leq_{pol}$ transitiv ist und wenn $L \leq_{pol} L'$ und $L' \in P$ impliziert, dass $L \in P$.*

Wir merken kurz an, dass Lemma 11.6 allgemeiner hätte formuliert werden können: an der Stelle der polynomiellen Reduktion „$\leq_{pol}$“ könnte eine beliebige abstrakte polynomielle Reduktion „$\leq_{pol}$“ stehen, welche zusätzlich die Bedingung

$$L_1 \leq_{pol} L_2 \Rightarrow \bar{L}_1 \leq_{pol} \bar{L}_2$$

erfüllt. Cook- oder Levin-Reduktionen bezitzen diese Eigenschaft ebenfalls, wie sich leicht **zeigen ließe**. Somit ergibt sich die **Übg.**

Folgerung 11.8 *Die Komplementärklasse der unter Cook- bzw. Levin-Reduktionen NP-vollständigen Sprachen stimmt überein mit der Klasse unter Cook- bzw. Levin-Reduktionen co-NPC-vollständigen Sprachen.*

Das folgende Lemma gibt ein paar zur $NP \neq$ co-NP-Voraussetzung äquivalente Bedingungen an.

Lemma 11.9 *Folgende Aussagen sind äquivalent:*

1. *$NP \neq$ co-NP.*
2. *$NPC \cap$ co-$NP = \emptyset$.*
3. *co-$NPC \cap NP = \emptyset$.*
4. *$NP \cap$ co-$NP \subseteq NPI \cup P$.*

Beweis Zur Äquivalenz der ersten beiden Aussagen genügt es freilich

$$NP = \text{co-}NP \Leftrightarrow NPC \cap \text{co-}NP \neq \emptyset$$

zu zeigen. $NP = \text{co-}NP \Rightarrow NPC \cap \text{co-}NP \neq \emptyset$ erhalten wir wie folgt:

$$NP = \text{co-}NP \Rightarrow NPC \cap \text{co-}NP = NPC \cap NP = NPC \neq \emptyset \ .$$

Nehmen wir umgekehrt an, dass $NPC \cap \text{co-}NP \neq \emptyset$. Dann gibt es eine NP-vollständige Sprache L_0, die auch zu co-NP gehört. Da jede Sprache $L \in NP$ auf L_0 polynomiell reduzierbar ist, ergibt sich hieraus $NP \subseteq \text{co-}NP$. Dies wiederum impliziert co-$NP \subseteq$ co-co-$NP = NP$ und wir erhalten $NP = \text{co-}NP$.
Zur Äquivalenz von Aussage 2 und 3 genügt es freilich

$$NPC \cap \text{co-}NP \neq \emptyset \Leftrightarrow \text{co-}NPC \cap NP \neq \emptyset$$

zu zeigen. Hierzu nutze aus, dass für alle Sprachen L die Aussagen $L \in NPC \cap \text{co-}NP$ und $\bar{L} \in \text{co-}NPC \cap NP$ äquivalent sind.
Die Äquivalenz der Aussagen 2 und 4 ist offensichtlich. •

Die durch Lemma 11.9 beschriebene Situation ist in Abbildung 12 noch einmal zusammengefasst.

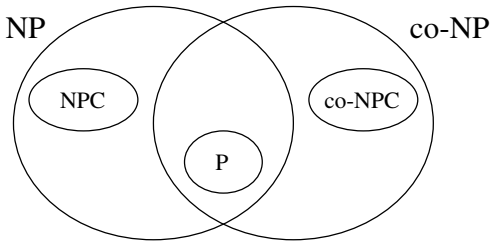


Abbildung 12: Struktur von $NP \cup \text{co-}NP$ unter der $NP \neq \text{co-}NP$-Voraussetzung.

Gemäß Satz 11.1 gibt es ein Problem in NPI. Wie sich dem Beweis dieses Satzes entnehmen lässt, ist dieses Problem ein sehr künstliches — einzig geschaffen, um $NPI \neq \emptyset$ zu demonstrieren. Wir präsentieren zum Abschluss dieses Kapitels einen *natürlichen* Kandidaten für ein Problem aus NPI. Es handelt sich dabei um eine Entscheidungsproblemversion des Faktorisierungsproblems, also des Problems eine natürliche Zahl in ihre Primfaktoren zu zerlegen:

Eingabe: $n \in \mathbb{N}$ und $i \in \{1, \ldots, \lfloor \log n \rfloor\}$

Frage: Ist das i-te Bit des kleinsten Primteilers p von n gleich 1?

Hierbei wird angenommen, dass die Binärdarstellung von p mit führenden Nullen auf exakt $\lfloor \log n \rfloor$ Bits ausgepolstert ist. Das beschriebene Entscheidungsproblem höre auf den Namen

FACTORING. Es ist leicht zu sehen, dass aus FACTORING $\in P$ folgen würde, dass die Zerlegung in Primzahlen in Polynomialzeit bewerkstelligt werden kann. Dies wiederum hätte zur Folge, dass das Public Key Cryptosystem (= PKCS) RSA geknackt wäre (das derzeit einzige im industriellen Maßstab eingesetzte PKCS. Die meisten Expertinnen und Experten vermuten daher, dass FACTORING $\notin P$. Andererseits gilt:

Lemma 11.10 *FACTORING* $\in NP \cap$ *co-NP.*

Beweis Rate und verifiziere die Primfaktorzerlegung von n, schnappe dir den kleinsten Primteiler p von n und verifiziere, dass das i-te Bit von p gleich 0 bzw. gleich 1 ist. In letzterem Fall wäre $\langle n, i\rangle \in FACTORING$, in ersterem Fall wäre $\langle n, i\rangle$ im Komplement von FACTORING. Aus diesen Überlegungen folgt FACTORING $\in NP \cap$ co-NP. •

Aus FACTORING $\in NPC$ würde nun mit Hilfe von Lemma 11.9 $NP =$ co-NP folgen. Wir erhalten also das

Corollary 11.11 *Unter der NP* $\neq$ *co-NP-Voraussetzung gilt*

$$FACTORING \in (NP \cap \text{co-}NP) \setminus (NPC \cup \text{co-}NPC)$$

und somit auch FACTORING $\in NPI \cup P$.

Aus der Vermutung FACTORING $\notin P$ ergibt sich damit konsequenterweise die Vermutung FACTORING $\in NPI$.

12 Isomorphe, dünne und dichte Sprachen

In diesem Abschnitt legen wir die Vermutung nahe, dass alle NP-vollständigen Sprachen zueinander „isomorph“ sind. Dies würde ausschließen, dass es „dünne“ NP-vollständige Sprachen (mit poly(n) Wörtern einer Maximallänge n) gibt.

Die Theorie der isomorphen, dünnen und dichten Sprachen wurde 1977 von Berman und Hartmanis ins Leben gerufen. Da sie die „P versus NP“ Frage aus einer neuen Perspektive beleuchtet, präsentieren wir im Folgenden ihre zentralen Konzepte und Resultate.

Definition 12.1 *Zwei Sprachen* $L_1, L_2 \in \Sigma^*$ *heißen* polynomiell isomorph*, wenn eine bijektive Abbildung* $h : \Sigma^* \to \Sigma^*$ *mit Umkehrabbildung* $h^{-1} : \Sigma^* \to \Sigma^*$ *existiert, die folgende Bedingungen erfüllt:*

1. *Sowohl h als auch h^{-1} sind in Polynomialzeit berechenbar.*
2. *Für alle $x \in \Sigma^*$ gilt $x \in L_1$ gdw $h(x) \in L_2$.*

In Zeichen: $L_1 \overset{pol}{\simeq} L_2$.

Offensichtlich gilt

$$L_1 \overset{pol}{\simeq} L_2 \Rightarrow L_1 \overset{pol}{\sim} L_2 \ ,$$

d.h., zwei polynomiell isomorphe Sprachen sind erst recht polynomiell äquivalent.

Wie wir wissen sind alle NP-vollständigen Sprachen wechselseitig aufeinander polynomiell reduzierbar und somit polynomiell äquivalent. Die polynomiellen Reduktionen, die wir dabei üblicherweise benutzen, sind aber weit davon entfernt, Bijektionen zu sein. Insbesondere die Surjektivität ist selten erfüllt, da wir bei einer Reduktion eines Quellproblems auf ein Zielproblem in der Regel extrem spezielle Eingabeinstanzen des Zielproblems konstruieren. Eine der seltenen Ausnahmen bildet die Reduktion $(G,k) \mapsto (\bar{G},k)$ von CLIQUE auf IS bzw. die Reduktion $(G,k) \mapsto (G, n-k)$ von IS auf VC, wobei $\bar{G}$ den Komplementärgraphen[36] zu G bezeichnet und n die Anzahl der Knoten in G.

Wir skizzieren im Folgenden eine Technik zur Verwandlung einer polynomiellen Reduktion in eine polynomielle Isomorphie. Diese Technik ist immer dann anwendbar, wenn die Zielsprache der polynomiellen Reduktion über eine sogenannte „Polsterfunktion (padding function“) verfügt:

Definition 12.2 *Eine Funktion* $pad : \Sigma^* \times \Sigma^* \to \Sigma^*$ *heißt* Polsterfunktion *für* $L \subseteq \Sigma^*$*, wenn die folgenden Bedingungen erfüllt sind:*

1. *Die Funktion pad ist in Polynomialzeit berechenbar.*
2. *Für alle* $x, y \in \Sigma^*$ *gilt* $x \in L$ *gdw* $pad(x,y) \in L$*.*
3. *Für alle* $x, y \in \Sigma^*$ *gilt* $|pad(x,y)| > |x| + |y|$*.*
4. *Aus* $pad(x,y)$ *kann* y *in Polynomialzeit extrahiert werden.*

Eine Polsterfunktion für L ist also im Wesentlichen eine Längen-expandierende polynomielle Reduktion von L auf sich selbst, die (auf leicht dekodierbare Weise) einen zusätzlichen String y in den Eingabestring x hinein kodiert. Wir verwenden im Folgenden oBdA das Binäralphabet $\Sigma = \{0,1\}$.

Beispiel 12.3 *Wir skizzieren den Entwurf einer Polsterfunktion für SAT. Es bezeichne* $x \in \Sigma^*$ *einen String zur natürlichen Kodierung einer Eingabeinstanz von SAT, d.h., x repräsentiert eine Boolesche Formel in konjunktiver Normalform mit m Klauseln* $C_0, \ldots, C_{m-1}$*, in denen n Variable, sagen wir* $v_1, \ldots, v_n$*, verwendet werden. Für* $p = |y|$ *bezeichne dann* $pad(x,y)$ *den Kodierungsstring für die Klauseln* $C_0, \ldots, C_{m-1}$ *(alte Klauseln) sowie* $C_m, \ldots, C_{m+p+1}$ *(neue Klauseln). Die neuen Klauseln verwenden neue Variable* $v_{n+1}, \ldots, v_{n+1+p}$ *und haben die folgende Form:*

- C_m *bestehe nur aus dem Literal* v_{n+1} *und* C_{m+1} *sei ein Duplikat von* C_m*.*
- *Für* $i = 1, \ldots, p$ *bestehe* C_{m+1+i} *nur aus dem Literal* v_{n+1+i} *sofern* $y_i = 1$ *bzw. nur aus dem Literal* $\bar{v}_{n+1+i}$ *sofern* $y_i = 0$*.*

[36] Zwei Knoten sind in $\bar{G}$ benachbart gdw sie in G nicht benachbart sind.

Die hierdurch beschriebene Funktion pad ist in Polynomialzeit berechenbar. Offensichtlich sind die Klauseln $C_0, \ldots, C_{m-1}$ erfüllbar gdw $C_0, \ldots, C_{m-1}, C_m, \ldots, C_{m+1+p}$ erfüllbar sind (da die neu hinzu gefügten Klauseln erstens erfüllbar sind und zweitens nur neue Variable verwenden). Die neue Instanz $pad(x, y)$ hat offensichtlich eine $|x|+|y|$ überschreitende Länge. Weiterhin lässt sich y aus $pad(x, y)$ leicht ablesen: durchmustere $pad(x, y)$ von rechts nach links bis zum ersten Auffinden von zwei identischen Klauseln (C_m und C_{m+1}); lies aus den davon rechts befindlichen Klauseln den Binärstring y (auf die offensichtliche Weise) ab. Somit sind alle an eine Polsterfunktion gerichteten Bedingungen erfüllt.

Polsterfunktionen lassen sich für NP-vollständige Sprachen in der Regel ähnlich leicht angeben wie für SAT. Den Entwurf von Polsterfunktionen für die Sprachen CLIQUE und PARTITION empfehlen wir als **Übung**. **Übg.**

Das folgende Resultat ergibt sich unmittelbar aus den Eigenschaften von Polsterfunktionen und Reduktionsabbildungen.

Lemma 12.4 *Es sei pad eine Polsterfunktion für L_2 und $L_1 \leq_{pol} L_2$ mit Reduktionsabbildung $R : \Sigma^* \to \Sigma^*$. Dann ist*

$$x \mapsto pad(R(x), x)$$

eine Längen-expandierende, injektive und in Polynomialzeit invertierbare Reduktionsabbildung.

Wechselseitige Längen-expandierende, injektive und in Polynomialzeit invertierbare Reduktionsabbildungen können in Isomorphismen transformiert werden:

Lemma 12.5 (Berman und Hartmanis, 1977) *Es seien L_1 und L_2 polynomiell äquivalente Sprachen, R eine Reduktionsabbildung für $L_1 \leq_{pol} L_2$ und S eine Reduktionsabbildung für $L_2 \leq_{pol} L_1$. Sowohl R als auch S seien Längen-expandierend, injektiv und in Polynomialzeit invertierbar. Dann sind L_1 und L_2 polynomiell isomorph.*

Beweis Da $R, S : \Sigma^* \to \Sigma^*$ Längen-expandierend, injektiv und in Polynomialzeit invertierbare Abbildungen sind, sind die Inversen $R^{-1}, S^{-1} : \Sigma^* \to \Sigma^*$ Längen-reduzierende, partiell definierte, injektive und in Polynomialzeit berechenbare Abbildungen. Der Beweis beruht auf dem Konzept der R- und S-Ketten. Eine *S-Kette* für $x \in \Sigma^*$ hat die Form

$$\cdots R^{-1}(S^{-1}(R^{-1}(S^{-1}(x)))) \ . \tag{19}$$

Dies ist so zu verstehen, dass wir die Abbildungen S^{-1} und R^{-1} solange alternierend anwenden wie das jeweilige Argument im Definitionsbereich liegt. Da R^{-1} und S^{-1} beide Längenreduzierend sind, muss die Kette nach maximal $n = |x|$ Schritten abbrechen. Wir unterscheiden die folgenden beiden Fälle:

Fall 1 Die S-Kette für x stoppt wegen eines undefinierten R^{-1}.
In diesem Fall hat die S-Kette die Form $S^{-1}(R^{-1}(\cdots S^{-1}(x) \cdots))$.

Fall 2 Die S-Kette für x stoppt wegen eines undefinierten S^{-1}.
In diesem Fall hat die S-Kette die Form $R^{-1}(S^{-1}(\cdots S^{-1}(x)\cdots))$.

Beachte, dass der Grenzfall eines undefinierten $S^{-1}(x)$ zur S-Kette x führt und einen Unterfall zu Fall 2 darstellt. Eine *R-Kette* für $x \in \Sigma^*$ ist analog definiert mit vertauschten Rollen von R und S. Wir definieren jetzt eine Abbildung $h : \Sigma^* \to \Sigma^*$, von der wir anschließend nachweisen, dass sie einen Isomorphismus von L_1 nach L_2 darstellt:

1. Falls die S-Kette für x wegen eines undefinierten R^{-1} stoppt, dann setzen wir $h(x) := S^{-1}(x)$.

2. Falls die S-Kette für x wegen eines undefinierten S^{-1} stoppt, dann setzen wir $h(x) := R(x)$.

Die folgende Abbildung g ist analog definiert (wieder mit vertauschten Rollen von R und S) und wird sich später als die Umkehrfunktion von h erweisen:

1. Falls die R-Kette für x wegen eines undefinierten S^{-1} stoppt, dann setzen wir $g(x) := R^{-1}(x)$.

2. Falls die R-Kette für x wegen eines undefinierten R^{-1} stoppt, dann setzen wir $g(x) := S(x)$.

Die zu beweisende Aussage $L_1 \simeq L_2$ ergibt sich unmittelbar aus den folgenden
Behauptungen:

1. h und g sind in Polynomialzeit berechenbar.

2. Für alle $x \in \Sigma^*$ gilt:

$$x \in L_1 \Leftrightarrow h(x) \in L_2 \text{ und } x \in L_2 \Leftrightarrow g(x) \in L_1 \ .$$

3. h und g sind Bijektionen und g ist die Umkehrfunktion von h.

Wir kommen nun zum Nachweis dieser drei Behauptungen. Zur Berechnung von $h(x)$ müssen wir zunächst die vollständige S-Kette für x berechnen, um festzustellen, ob sie wegen eines undefinierten R^{-1} oder wegen eines undefinierten S^{-1} stoppt. Dies erfordert maximal n Auswertungen der Funktionen R^{-1} bzw. S^{-1} (wobei die Eingabestrings niemals länger als $n = |x|$ werden, da R^{-1} und S^{-1} längenverkürzende Abbildungen sind), was in Polynomialzeit geleistet werden kann. Nachdem klar ist, welcher Fall vorliegt, muss dann entweder $h(x) = S^{-1}(x)$ oder $h(x) = R(x)$ berechnet werden. Auch dies ist geschieht in Polynomialzeit. Die Berechnung von $g(x)$ ist aus Symmetriegründen genau so aufwendig wie die Berechnung von h, was den Beweis von Behauptung 1 abschließt.
Freilich gilt $x \in L_1$ genau dann, wenn $h(x) \in L_2$: entweder $h(x) = S^{-1}(x)$ oder $h(x) = R(x)$, und sowohl S^{-1} als auch R (in ihrer Funktion als Reduktionsabbildungen) besitzen die

geforderte Eigenschaft. Der zweite Teil von Behauptung 2 ergibt sich analog.
Zum Nachweis von Behauptung 3 genügt es zu zeigen, dass

$$g(h(x)) = x \text{ und } h(g(x)) = x$$

für alle $x \in \Sigma^*$ gültig ist. Aus Symmetriegründen genügt es, die Gleichung $g(h(x)) = x$ zu verifizieren. Dies geschieht im Rahmen einer Fallunterscheidung:

Fall 1: Die S-Kette für x bricht wegen eines undefinierten R^{-1} ab.
Dann gilt $h(x) = S^{-1}(x)$. Eine Inspektion von (19) ergibt, dass die S-Kette für x übereinstimmt mit der R-Kette für $S^{-1}(x)$. Daher bricht letztere (genau so wie erstere) mit undefiniertem R^{-1} ab. Gemäß der Definition von g ergibt sich daher $g(h(x)) = g(S^{-1}(x)) = S(S^{-1}(x)) = x$.

Fall 2: Die S-Kette für x bricht wegen eines undefinierten S^{-1} ab.
Dann gilt $h(x) = R(x)$. Die R-Kette für $h(x) = R(x)$ hat die Form

$$\ldots R^{-1}(S^{-1}(R^{-1}(S^{-1}(\underbrace{R^{-1}(R(x))}_{=x})))) \ ,$$

d.h., sie entwickelt sich nach links wie die S-Kette für x. Daher bricht sie mit undefiniertem S^{-1} ab. Gemäß der Definition von g ergibt sich daher $g(h(x)) = g(R(x)) = R^{-1}(R(x)) = x$, was den Beweis der dritten Behauptung vervollständigt.

•

Folgerung 12.6 (Berman und Hartmanis, 1977) *Alle NP-vollständigen Sprachen, die mit einer Polsterfunktion versehen werden können, sind zueinander polynomiell isomorph.*

Die Tatsache, dass alle bekannten NP-vollständigen Probleme eine Polsterfunktion besitzen, führte zu der

Isomorphie-Vermutung (Berman und Hartmanis, 1977) Alle NP-vollständigen Sprachen sind zueinander polynomiell isomorph.

Es ist bis heute nicht geklärt, ob diese Vermutung zutrifft.

Definition 12.7 *Die* Dichte *bzw.* Dichtefunktion *einer Sprache $L \subseteq \Sigma^*$ ist definiert als die folgende Funktion in der Eingabelänge n:*

$$dens_L(n) := |\{x \in L : \ |x| \leq n\}| \ .$$

L heißt dünn *(oder auch* spärlich*), wenn ihre Dichte polynomiell in n beschränkt ist; L heißt* dicht*, wenn ihre Dichte superpolynomiell mit n wächst.*

Alle uns bisher bekannten NP-vollständigen Sprachen sind dicht!

Beobachtung 12.8 *Wenn L_1 und L_2 polynomiell isomorph sind, dann gibt es Polynome q_1, q_2, so dass*

$$dens_{L_1}(n) \leq dens_{L_2}(q_1(n)) \text{ und } dens_{L_2}(n) \leq dens_{L_1}(q_2(n)) \ .$$

Hieraus folgt, dass eine dünne und eine dichte Sprache nicht zueinander polynomiell isomorph sein können.

Wenn die Isomorphie-Vermutung zutrifft, dann würde aus dieser Beobachtung folgen, dass eine dünne Sprache nicht NP-vollständig sein kann. Diese Folgerung kann jedoch bereits unter der schwächeren $P \neq NP$-Voraussetzung bewiesen werden:

Satz 12.9 (Mahaney, 1982) *Falls $P \neq NP$, dann gibt es keine dünne und zugleich NP-vollständige Sprache.*

Ein Spezialfall von dünnen Sprachen stellen die *unären Sprachen* $L \subseteq \{0\}^*$ dar. Anstelle des Satzes von Mahaney beweisen wir den folgenden (etwas leichteren)

Satz 12.10 (Berman, 1978) *Falls $P \neq NP$, dann gibt es keine unäre und zugleich NP-vollständige Sprache.*

Beweis Wir führen den Beweis indirekt und zeigen, dass die Existenz einer NP-vollständigen unären Sprache $L_0 \subseteq \{0\}^*$ die Gleichheit von P und NP zur Folge hätte. Wir dürfen dabei $L_0 \neq \{0\}^*$ annehmen, da $L_0 = \{0\}^*$ (wegen $\{0\}^* \in P$ und $L_0 \in NPC$) sofort $P = NP$ zur Folge hätte. Es sei $R : \Sigma^* \to \Sigma^*$ eine Reduktionsabbildung für SAT $\leq_{pol} L_0$. Wir dürfen annehmen, dass R Strings aus Σ^* stets auf unäre Strings $R(x) \in \{0\}^*$ abbildet: Bildstrings $R(x)$, welche ein von 0 verschiedenes Symbol enthalten, können nämlich problemlos ersetzt werden durch einen willlkürlich und fest ausgewählten String aus $\{0\}^* \setminus L_0$. Wir werden im Folgenden die Reduktionsabbildung R benutzen, um einen polynomiellen Algorithmus für SAT zu entwerfen. Sei also $x \in \Sigma^N$ der Kodierungsstring für eine CNF-Formel F über den Booleschen Variablen $v_1, \ldots, v_n$. Wir skizzieren im Folgenden ein effizientes Verfahren, um die Erfüllbarkeit von F zu testen. Dabei identifizieren wir einen String $a \in \{0,1\}^j$ mit der partiellen Belegung

$$v_1 = a_1, \ldots, v_j = a_j \ .$$

Es bezeichne $F[a]$ die vereinfachte CNF-Formel, die aus F durch die von a repräsentierte partielle Belegung hervor geht.[37] Der Kodierungsstring für $F[a]$ sei mit $x[a]$ bezeichnet. Für $a = \epsilon$ (leeres Wort bzw. leere partielle Belegung) identifizieren wir $F[a]$ mit F und $x[a]$ mit x. Offensichtlich gilt für $j = 0, \ldots, n-1$ und alle $a \in \{0,1\}^j$ die Beziehung

$$F[a] \text{ ist erfüllbar} \Leftrightarrow F[a0] \text{ oder } F[a1] \text{ ist erfüllbar.}$$

Für alle $a \in \{0,1\}^n$ ist $F[a]$ eine Boolesche Konstante und somit erfüllbar gdw es sich um die Konstante 1 handelt. Ein exponentiell zeitbeschränkter Algorithmus könnte vorgehen wie folgt:

[37] Elimination aller durch die partielle Belegung bereits erfüllten Klauseln und, in den verbleibenden Klauseln, Elimination der durch die partielle Belegung bereits falsifizierten Literale

Top-down Pass Bilde einen vollständig binären Baum T der Tiefe n. Assoziiere mit einer Kante zum linken Kind das Bit 0 und mit einer Kante zum rechten Kind das Bit 1. Auf diese Weise ist zu jedem Knoten z der Tiefe j des Baumes ein eindeutiger Binärstring $a(z) \in \{0,1\}^j$ assoziiert. Insbesondere entsprechen die 2^n Blätter (Knoten der Tiefe n) 1-zu-1 den Strings aus $\{0,1\}^n$.

Bottom-up Pass Markiere jedes Blatt z mit der Booleschen Konstante $F[a(z)]$. Markiere anschließend jeden inneren Knoten z mit bereits markierten Kindern z_0, z_1 mit der Disjunktion (logisches „Oder“) der Booleschen Markierungen seiner Kinder.

Nach unseren obigen Ausführungen sollte klar sei, dass eine Formel $F[a(z)]$ mit 1 markiert wird gdw $F[a(z)]$ erfüllbar ist. Die Markierung an der Wurzel verrät uns, ob F erfüllbar ist. Der „Schönheitsfehler“ dieses Verfahrens ist die exponentielle Größe der verwendeten Datenstruktur T. Jetzt kommen zwei Ideen ins Spiel, die darauf hinaus laufen, dass nur ein polynomiell kleines Anfangsstück T' von T wirklich gebraucht wird:

Lazy Evaluation Wenn $F[a0]$ erfüllbar ist, dann ist (unabhängig von der Erfüllbarkeit von $F[a1]$) auch die Formel $F[a]$ erfüllbar. Im Baum T bedeutet dies für einen Knoten z mit linkem Kind z_0 und rechtem Kind z_1: wenn z_0 beim „bottom-up pass“ die Markierung 1 erhalten hat, brauchen wir uns für den Unterbaum mit Wurzel z_1 nicht mehr zu interessieren und können z direkt mit 1 markieren. Nach welcher Strategie muss der Baum T gebildet werden, um diesen Effekt voll auszunutzen? Die Antwort lautet: „Durchlauf in Präordnung (preorder traversal)“.[38] Der Durchlauf in Präordnung stellt sicher, dass wir den „bottom-up pass“ des Unterbaumes von z_0 bereits abgeschlossen haben, bevor wir in den „top-down pass“ des Unterbaumes von z_1 einsteigen. Abbildung 13 illustriert diesen Gedankengang.

Hashing Wir verwenden die Reduktionsabbildung R als eine Art „Hashfunktion“. Dabei nutzen wir aus, dass zwei CNF-Formeln F, F' mit Kodierungsstrings x, x' und $R(x) = R(x')$ entweder beide erfüllbar sind (nämlich, wenn $R(x) = R(x') \in L_0$) oder beide nicht erfüllbar sind (nämlich, wenn $R(x) = R(x') \notin L_0$). Im Baum T bedeutet dies für einen Knoten z': falls zum Zeitpunkt des Kreierens von z' bereits ein Knoten z in T existiert, welcher mit einem Bit markiert ist und die Bedingung $R(x[a(z)]) = R(x[a(z')]$ erfüllt, dann können wir z' direkt mit dem gleichen Bit markieren. (Da wir für z' keine Kinder kreieren müssen, wird z' dann zu einem Blatt des Baumes T'.)

Beide Ideen verfolgen im Kern eine „pruning“-Technik: wir können von dem exponentiell großen Baum T Teilbäume „abschneiden“, wenn sie für unsere Markierungsstrategie nur redundante Information liefern. Es sollte klar sein, dass der komplette Algorithmus für SAT (unter Einsatz von „pruning“) in Polynomialzeit implementiert werden kann, wenn das Anfangsstück T' von T (T ohne die abgeschnittenen bzw. gar nicht erst kreierten Teilbäume)

[38] Durchlauf eines Baumes mit Wurzel r, Wurzelkindern r_0, r_1 und den Unterbäumen T_0, T_1 in Präordnung ist rekursiv definiert wie folgt: durchlaufe zuerst r dann (rekursiv) alle Knoten von T_0 und schließlich (rekursiv) alle Knoten von T_1.

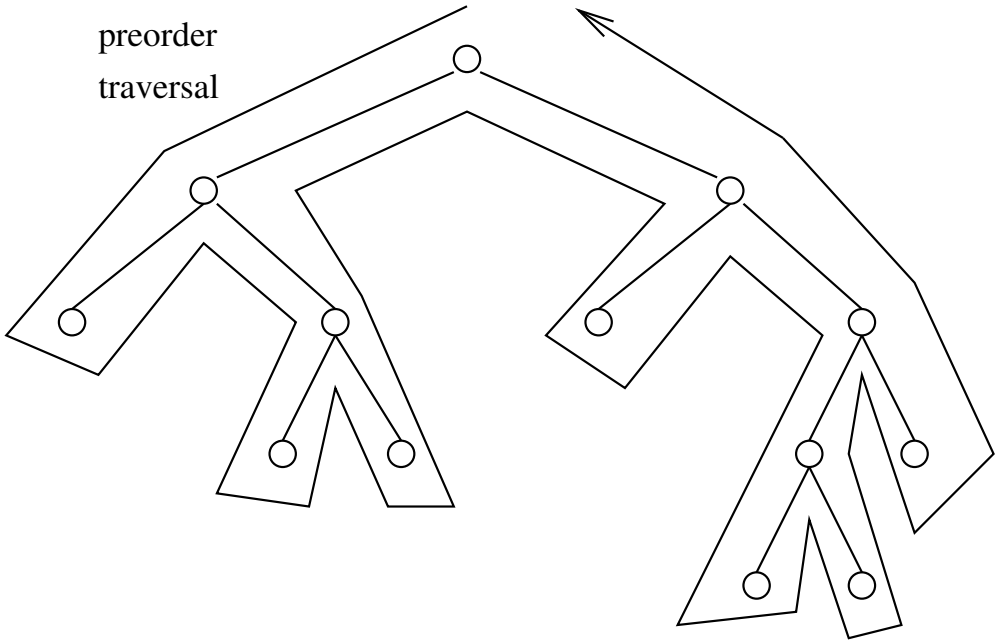


Abbildung 13: Ein Durchlauf der Knoten eines Baumes in Präordnung: bei zwei unabhängigen Knoten z, z' mit z links von z' ist der „bottom-up pass“ von z abgeschlossen, bevor der „top-down pass“ von z' beginnt.

eine polynomiell beschränkte Größe hat. Wir argumentieren nun, dass dies der Fall ist. Da R in Polynomialzeit berechenbar ist, muss es ein Polynom q geben, so dass für alle $x \in \Sigma^*$ gilt:

$$|x| \leq N \Rightarrow |R(x)| < q(N) \ .$$

Da R ausschließlich auf Wörter aus $\{0\}^*$ abbildet, folgt hieraus, dass die Anzahl der bei unserem SAT-Algorithmus auftretenden Hashwerte durch $q(N)$ beschränkt ist. Wie ist nun die Beziehung zwischen der Anzahl der Knoten in T' und der maximalen Anzahl $q(N)$ von Hashwerten? Die entscheidende Beobachtung ist die folgende:
Zwei unabhängige[39] innere Knoten z, z' in T' haben verschiedene Hashwerte. Insbesondere haben alle Knoten der selben Tiefe $j \in \{0, 1, \ldots, n\}$ paarweise verschiedene Hashwerte.
Der Grund hierfür ist einfach: einer der Knoten z und z', sagen wir z, wurde zuerst durchlaufen und mit einem Bit markiert, bevor der andere, also z', besucht wird; wäre $R(z) = R(z')$ dann hätten wir zu z' keine Kinder kreiert und z' müsste ein Blatt sein. Auf Tiefe 0 befindet sich nur die Wurzel von T'. Auf den Tiefenstufen $1, \ldots, n$ können sich maximal $nq(N)$ weitere innere Knoten befinden. Die Gesamtanzahl der inneren Knoten in T' ist daher durch $1 + nq(N)$ beschränkt. Da jeder innere Knoten maximal zwei Kinder hat, kann die Anzahl der Blätter in T' die Anzahl der inneren Knoten maximal um 1 überschreiten. Demzufolge hat T' insgesamt maximal $3 + 2nq(N)$ Knoten. Da $n \leq N$ und $N = |x|$ die Eingabelänge bezeichnet, ist der Beweis abgeschlossen. •

[39] „unabhängig“ bedeutet „kein Knoten ist Vorfahr des anderen“.

Die Sätze 12.9 und 12.10 bedeuten im Umkehrschluss: wenn es gelänge eine NP-vollständige unäre bzw. dünne Sprache ausfindig zu machen, dann wäre $P = NP$.

13 Relativierung der P versus NP Frage

Baker, Gill und Solovay haben in einer Aufsehen erregenden Publikation aus dem Jahre 1975 untersucht, ob die Frage der (Un-)gleichheit von P und NP in einer durch die Anwesenheit von Orakeln „relativierten Welt“ entschieden werden kann. In einer solchen Welt würde die Rechenkraft von DTMs oder NTMs künstlich erhöht, indem bestimmte (einer Relation oder formalen Sprache entsprechende) Anfragen an ein Orakel weiter geleitet werden können und von diesem dann jeweils in einem Schritt beantwortet werden (Konzept der Orakel-Turing-Maschinen oder kurz OTMs, DOTMs, NOTMs). Warum sollte die Anwesenheit eines Orakel die Beantwortung der P versus NP Frage erleichtern? Hier sind zwei Gründe:

- Um den Unterschied zwischen P und NP zu nivellieren, könnte man ein Orakel benutzen, welches deterministische Maschinen so mächtig macht, dass durch Nichtdeterminismus keine zusätzlichen Fähigkeiten erwachsen.

- Um den Unterschied zwischen P und NP zu verstärken (und dadurch mathematisch beweisbar zu machen), könnte man Orakel benutzen, die die Fähigkeiten nichtdeterministischer Maschinen signifikant stärker potenzieren als die von deterministischen.

Es wird sich zeigen, dass in der Tat beide Grundideen durchführbar sind. In diesem Sinne ergibt sich durch Relativierung der P versus NP Frage gewissermaßen ein „unentschieden“: für manche Relativierungen gilt $P = NP$, für andere gilt $P \neq NP$.

Also was (so what)?

Diese Untersuchungen zeigen in erster Linie auf, dass klassische Techniken zum Nachweis der (Un-)gleichheit zweier Komplexitätsklassen in Bezug auf die P versus NP Frage zum Scheitern verdammt sind:

- Eine klassische Technik zum Nachweis von $\mathcal{C} = \mathcal{C}'$ (Gleichheit zweier Komplexitätsklassen) ist *(wechselseitige) Schritt-für-Schritt Simulation*: zeige, dass jeder Rechenschritt einer zu $\mathcal{C}$ passenden Maschine M durch (evtl. mehrere) Rechenschritte einer zu $\mathcal{C}'$ passenden Maschine M' simuliert werden kann (und umgekehrt). Diese Beweistechnik lässt sich relativieren: wenn beide Maschinen zusätzlich mit einem Orakel versehen sind, dann kann die Simulation aufrecht erhalten werden, indem herkömmliche Rechenschritte simuliert werden wie zuvor und eine Orakelanfrage von M „simuliert wird“ durch dieselbe Orakelanfrage von M'. Wenn sich also $\mathcal{C} = \mathcal{C}'$ mit Hilfe von Schritt-für-Schritt Simulation nachweisen lässt, dann sollte die Gleichheit in **jeder** relativierten Welt gelten.

- Eine klassische Technik zum Nachweis von $\mathcal{C} \subset \mathcal{C}'$ (echte Inklusion) ist *(einseitige) Schritt-für-Schritt Simulation* verbunden mit *Diagonalisierung*: wenn $M_1, M_2, \ldots$ eine

Aufzählung aller zu $\mathcal{C}$ passenden Maschinen ist und $z_1, z_2, \ldots$ ist eine Aufzählung aller Eingabestrings, dann versuche eine zu $\mathcal{C}'$ passende Maschine M' anzugeben, die auf Eingabestring z_i die Maschine M_i Schritt-für-Schritt simuliert, aber am Ende der Rechnung die Antwort JA/NEIN zu NEIN/JA rumdreht. M' ist dann Akzeptor einer Sprache aus $\mathcal{C}' \setminus \mathcal{C}$. Auch diese Beweistechnik lässt sich (in der offensichtlichen Weise) relativieren. Wenn sich also $\mathcal{C} \subset \mathcal{C}'$ mit Hilfe von (auf Schritt-für-Schritt Simulation basierender) Diagonalisierung nachweisen lässt, dann sollte die echte Inklusionsbeziehung in **jeder** relativierten Welt gelten.

Diese Überlegungen zeigen im Umkehrschluss: da die Antwort auf die P versus NP Frage in den relativierten Welten zweideutig ausfällt, kann weder die Gleicheit noch die Ungleichheit von P und NP mit einer relativierenden Technik (wie Schritt-für-Schritt Simulation, Diagonalisierung,...) geführt werden.[40]

Nach diesen philosophischen Anmerkungen kommen wir nun zur eigentlichen Arbeit. Es sei daran erinnert, dass M^R bzw. M^L eine Orakel-Turing-Maschine (OTM) mit einem Orakel für die Relation R bzw. für die Sprache L bezeichnet. Wenn M einen Fragestring x auf das spezielle Orakelband schreibt und in den Fragezustand $q_?$ übergeht, dann ersetzt das R-Orakel (in **einem** Schritt) den String x durch einen String y mit $(x, y) \in R$ (sofern möglich) und „beamt" M in den Zustand q_+. Wenn es keinen passenden String y gibt, dann wird x nicht ersetzt und M wird in den Zustand q_- „gebeamt". Im Falle eines L-Orakels wird im Falle $x \in L$ der String x gelöscht und M in Zustand q_+ „gebeamt"; falls $x \notin L$, dann wird x nicht gelöscht und M in Zustand q_- „gebeamt". Wenn M deterministisch ist, sprechen wir von einer DOTM M^R bzw. M^L; ist M nichtdeterministisch sprechen wir von einer NOTM. Wenn $\mathcal{C}$ eine (deterministische oder nicht-deterministische) Zeitkomplexitätsklasse bezeichnet, dann sei $\mathcal{C}^R$ bzw. $\mathcal{C}^L$ die Klasse aller Sprachen, die sich mit einer OTM M^R bzw. M^L erkennen lassen, wobei M eine zu $\mathcal{C}$ passende Maschine sein muss.

Welches Orakel könnte geeignet sein, den Unterschied zwischen P und NP (beweisbar) zu nivellieren? Zur Beantwortung dieser Frage erinnern wir an die Definition von PSpace:

$$PSpace = \bigcup_{k \geq 1} DSpace(n^k) = \bigcup_{k \geq 1} NSpace(n^k) \ ,$$

wobei sich letzte Gleichung aus dem (in der Einleitung bereits genannten) Satz von Savitch ergibt, der im Wesentlichen besagt, dass eine NTM mit Platzschranke $S(n) \geq \log n$ sich durch eine DTM mit Platzschranke $S^2(n)$ simulieren lässt. Das impliziert

$$PSpace = \bigcup_{k \geq 1} NSpace(n^k) \ .$$

Vereinfacht gesagt: PSpace nivelliert den Unterschied zwischen Determinismus und Nichtdeterminismus. Diese Tatsache kommt im Beweis des folgenden Resultates zum Tragen:

[40] Viele „Beweise" für $P = NP$ oder $P \neq NP$, die (mehr oder weniger qualifizierte) Forscher der Fachwelt präsentiert haben, scheitern bereits an dieser Hürde: es wurde eine relativierende Beweistechnik verwendet. Ein solcher Beweis kann von vorne herein nicht korrekt sein.

Satz 13.1 (Baker, Gill, and Solovay, 1975) *Es gibt eine Sprache A mit $P^A = NP^A$.*

Beweis Es sei A eine (unter Karp-Reduktionen)*PSpace*-vollständige Sprache.[41] Dann folgt $P^A = NP^A$ aus

$$PSpace \subseteq P^A \subseteq NP^A \subseteq \bigcup_{k\geq 1} NSpace(n^k) = \bigcup_{k\geq 1} DSpace(n^k) = PSpace \ . \tag{20}$$

Die erste Inklusion gilt wegen der *PSpace*-Vollständigkeit von A. Sei nämlich $L \in PSpace$, R eine Reduktionsabbildung für $L \leq_{pol} A$ und $x \in \Sigma^n$ ein Eingabestring. Um die Frage der Mitgliedschaft von x in L zu entscheiden, können wir zunächst in poly(n) Schritten $R(x)$ berechnen und dann das A-Orakel nach der Mitgliedschaft von $R(x)$ in A befragen. Die zweite Inklusion in (20) ist trivial. Die dritte Inklusion in (20) ergibt sich daraus, dass eine NOTM M^A für $L \in NP^A$ durch eine NTM M' mit polynomieller Platzschranke simuliert werden kann: wann immer M^A das A-Orakel nach der Mitgliedschaft eines Strings z zur Sprache A befragt, kann sich M' (wegen $A \in PSpace$) die Antwort selber herleiten. Die vorletzte Gleichung ergibt sich aus dem Satz von Savitch, und die letzte Gleichung ist die Definition von PSpace. •

Ein Orakel, dass P von NP trennt, ist etwas schwieriger zu konstruieren, existiert aber ebenfalls:

Satz 13.2 (Baker, Gill, and Solovay, 1975) *Es gibt eine Sprache B mit $P^B \neq NP^B$.*

Beweis Wir konstruieren eine Sprache B und eine (von B abhängige) Sprache $L_* \in NP^B \setminus P^B$ und beginnen mit der Definition von L_*:

$$L_* := \{0^n |\ B \cap \Sigma^n \neq \emptyset\} \ .$$

Bereits ohne Kenntnis von B ist sonnenklar, dass L_* zur Klasse NP^B gehört, da die Wörter $0^n \in L_*$ mit Hilfe des B-Orakels nichtdeterministisch in Polynomialzeit erkannt werden können: rate $x \in \Sigma^n$ und verifiziere $x \in B$ durch Anfrage an das B-Orakel. Die Kampfaufgabe im restlichen Beweis wird darin bestehen, B so zu definieren, dass $L_* \notin P^B$. Zu diesem Zweck greifen wir auf die Technik der Diagonalisierung zurück. Es bezeichne

$$M_1^B, M_2^B, \ldots$$

eine Aufzählung[42] aller polynomiell zeitbeschränkter DOTMs mit Orakel B, so dass

$$P^B = \{L_{M_i^B} |\ i \geq 1\} \ .$$

[41] Beispiele solcher Sprachen werden wir in einem späteren Abschnitt kennen lernen.

[42] Diese Aufzählung hängt **nicht** von B ab, da im Programm einer DOTM M_i lediglich festgelegt ist, wie nach Erreichen der Zustände q_- bzw. q_+ weiter gearbeitet wird. Die Abhängigkeit von B ergibt sich so zu sagen erst zur Laufzeit.

Wir setzen zusätzlich voraus, dass jede Sprache in P^B, die in Zeit poly(n) mit Hilfe eines B-Orakels erkannt werden kann in der Aufzählung mit unendlich vielen Akzeptoren vertreten ist (was durch „redundante Kodierung“ von Turing-Maschinen-Programmen leicht zu erreichen ist). Wir definieren nun B stufenweise, wobei

$$B_i = B \cap \Sigma^i$$

die *i-the Stufe* von B darstellt. Nebenbei protokollieren wir eine Menge X von „Ausnahme-strings“, die wir auf keinen Fall in B aufzunehmen wünschen. Anfangs gilt

$$B_0 = X = \emptyset \ .$$

Die Menge X ist aber eine dynamische Größe, die sich im Laufe der Konstruktion von B verändert. In Stufe i der Konstruktion wollen wir im Prinzip erreichen, dass der String 0^i die Sprachen L_* und $L_{M_i^B}$ trennt:

Stufe i „Simuliere“ M_i^B auf Eingabestring 0^i für

$$q(i) := i^{\lfloor \log i \rfloor}$$

viele Schritte.[43] Bei der Simulation kann M_i^B das B-Orakel nach der Mitgliedschaft von Strings z zur Sprache B befragen. Falls $|z| \leq i-1$, antworten wir in der Simulation mit JA, falls $z \in B_{|z|}$, und mit NEIN, falls $z \notin B_{|z|}$. Beachte dabei, dass die $|z|$-te Stufe von B bereits vorher konstruiert wurde. Falls aber $|z| \geq i$, dann antworten wir mit NEIN und nehmen z in X auf (was die Antwort NEIN im Nachhinein rechtfertigen wird, da wir Strings aus X niemals in B aufnehmen werden). Auf diese Weise kann die Simulation am Laufen gehalten werden. Beachte, dass in Stufe i maximal $q(i)$ Strings in X aufgenommen werden. Eine leichte Rechnung ergibt, dass

$$\sum_{j=1}^{i} q(j) < 2^i \ ,$$

d.h., nach Stufe i muss $X \cap \Sigma^i$ eine echte Teilmenge von Σ^i sein. Nun sind drei Fälle denkbar:

Fall 1 M_i^B verwirft 0^i in maximal $q(i)$ Schritten.
Dann setzen wir

$$B_i := \Sigma^i \setminus X \neq \emptyset \ .$$

Nun gilt $0^i \notin L_{M_i^B}$ und (gemäß der Definition von L_*) $0^i \in L_*$.

Fall 2 M_i^B akzeptiert 0^i in maximal $q(i)$ Schritten.
Dann setzen wir

$$B_i := \emptyset$$

mit der Konsequenz, dass $0^i \in L_{M_i^B}$ und (wieder gemäß der Definition von L_*) $0^i \notin L_*$.

[43] Die Funktion $q(i)$ ist so gewählt, dass sie asymptotisch schneller wächst als jedes Polynom, aber langsamer als 2^i.

Fall 3 M_i^B kommt im Laufe von $q(i)$ Schritten zu keiner Entscheidung.
In diesem Fall setzen wir auch $B_i := \emptyset$, gelangen aber (im Unterschied zu den ersten beiden Fällen) zu keiner Separation von L_* und $L_{M_i^B}$.

Ohne den Fall 3 wäre der Diagonalisierungsbeweis nun abgeschlossen. Erinnern wir uns daran, dass wir listigerweise verlangt hatten, dass jede Sprache $L \in P^B$, die mit Hilfe eines B-Orakels in Zeit poly(n) erkennbar ist, in der Aufzählung der M_i unendlich oft vertreten ist. Da $q(i)$ schneller wächst als jedes Polynom, wird für Akzeptoren von L mit hinreichend großem Index i der Fall 3 nicht eintreten. Somit kann jede Sprache aus P^B von L_* getrennt werden. •

14 Beziehungen zwischen den Komplexitätsklassen

In Abschnitt 14.1 behandeln wir die Frage, ob eine TM selber erzwingen kann, dass sie eine vorgegebene Platz- oder Zeitschranke nicht überschreitet. Dies führt uns zum Thema der „Kontrolle der Ressourcenschranken“ und zu dem Begriff der „platz- und zeitkonstruierbaren Funktionen“. In Abschnitt 14.2 betrachten wir das Verhältnis von Determinismus und Nichtdeterminismus. Es wird sich zeigen, dass NTMs platzeffizient deterministisch simulierbar sind (Satz von Savich). Die Frage, ob eine solche Simulation auch zeiteffizient gestaltet werden kann, hängt mit der „P versus NP“ Frage zusammen. Die besten bekannten deterministichen Simulationen von $T(n)$-zeitbeschräkten NTMs benötigen exponentiell in $T(n)$ viele Rechenschritte. In Abschnitt 14.3 diskutieren wir, welche Komplexitätsklassen abgeschlossen unter Komplement sind. Für deterministische Komplexitätsklassen ist das (trivialerweise) der Fall, ebenso bei nichtdeterministischen Platzkomplexitätsklassen (Satz von Immerman und Szelepcsényi). Den Abschluss des Kapitels bildet die Platz-Zeit-Hierarchie: eine aufsteigende Kette von Komplexitätsklassen zwischen $\mathcal{L}$ und *PSpace*.

14.1 Kontrolle der Ressourcenschranken

Eine Schlüsseltechnik zur Analyse der Beziehungen zwischen verschiedenen Komplexitätsklassen ist die Simulation einer TM M durch eine andere TM M'. Dabei wird es von Bedeutung sein, dass M' ihre Ressourcenschranken nicht überschreitet. Unter Umständen muss eine Simulation von M durch M' erfolglos abgebrochen werden, wenn sie zur Überschreitung der Ressourcenschranke führen würde. Wie aber kann M' die Kontrolle darüber bewahren? Die Beantwortung dieser Frage beruht auf den Konzepten der Platz- und Zeitkonstruierbarkeit.

Definition 14.1 (Platzkonstruierbarkeit) *Eine Funktion $S : \mathbb{N} \to \mathbb{N}$ heißt* platzkonstruierbar*, wenn die Funktion $1^n \mapsto 1^{S(n)}$ von einer $S(n)$-platzbeschränkten DTM berechnet werden kann.*

Die Hauptanwendung der Platzkonstruierbarkeit besteht darin, ein Bandsegment der Größe $S(n)$ abzustecken (zum Beispiel mit Endmarkierungen auf der Zelle am weitesten links

bzw. rechts). Dies geschieht oft in Verbindung mit einer Simulation, welche abgebrochen wird, wenn sie zum Verlasssen des abgesteckten Bandsegmentes führen würde.[44]

Definition 14.2 (Zeitkonstruierbarkeit) *Eine Funktion $T : \mathbb{N} \to \mathbb{N}$ heißt* zeitkonstruierbar*, wenn die Funktion $1^n \mapsto bin(T(n))$ von einer $T(n)$-zeitbeschränkten 2-Band DTM M berechnet werden kann.*

Die Hauptanwendung der Zeitkonstruierbarkeit besteht darin, einen Binärzähler mit $\text{bin}(T(n))$ zu initialisieren und dann in Verbindung mit einer Simulation als „Uhr“ zu verwenden: nach jedem Schritt in der Simulation wird der Zähler um 1 dekrementiert; erreicht er den Wert 0 (Zeit abgelaufen), dann wird die Simulation abgebrochen.[45]

Wir merken kurz an: offensichtlich sind zeitkonstruierbare Funktionen erst recht platzkonstruierbar. Die folgenden Beobachtungen zeigen, dass die Klasse der konstruierbaren Funktionen sehr reichhaltig ist: **Übg.**

1. Jede konstante Funktion $n \mapsto c$ mit $c \in \mathbb{N}$ ist platz- und zeitkonstruierbar.
2. Die Funktionen $n \mapsto |\text{bin}(n)| = 1 + \lfloor \log n \rfloor$, $n \mapsto \lfloor \log n \rfloor$ und $n \mapsto \lceil \log n \rceil$ sind platzkonstruierbar.
3. Die Funktionen $n \mapsto n$ und $n \mapsto 2^n$ sind platz- und zeitkonstruierbar.
4. Platz- und zeitkonstruierbare Funktionen sind abgeschlossen unter Addition und Multiplikation.
5. Alle Polynome mit nicht-negativen ganzzahligen Koeffizienten (kurz: positive Polynome) sind platz- und zeitkonstruierbar.

14.2 Verhältnis von Determinismus und Nicht-Determinismus

Wir beginnen mit ein paar trivialen bzw. uns schon bekannten Beobachtungen. R, S, T seien Ressourcenschranken. Offensichtlich gilt

$$DTime(T) \subseteq NTime(T) \text{ und } DSpace(S) \subseteq NSpace(S) \ ,$$

da sich eine DTM als Spezialfall einer NTM auffassen lässt. Die ebenfalls offensichtlichen Inklusionen

$$DTime(R) \subseteq DSpace(R) \text{ und } NTime(R) \subseteq NSpace(R) \ .$$

ergeben sich einfach daraus, dass jeder Besuch einer noch unbesuchten Zelle mindestens einen Rechenschritt erfordert. Die uns bereits bekannte deterministische Simulation von NTMs impliziert folgenden

[44] Selbst die Berechnung von $1^{S(n)}$ erfordert (gemäß dem Kompressionssatz) lediglich einen Bandbereich von $S(n)$ statt $O(S(n))$ Zellen.

[45] Mit einer amortisierten Analyse lässt sich zeigen, dass der „Overhead“ zum Zurückzählen des Binärzählers von $\text{bin}(T(n))$ auf 0 durch $O(T(n))$ Rechenschritte beschränkt ist. Weiterhin ergibt sich aus dem Beschleunigungssatz, dass eine Rechnung, die $O(T(n))$ Rechenschritte erfordert, auch in $\leq T(n)$ Rechenschritten zu bewerkstelligen ist, sofern eine Konstante $\alpha > 0$ existiert mit $T(n) \geq (1 + \alpha)n$.

Satz 14.3 *Für jede Funktion T gilt:*

$$NTime(T) \subseteq DTime\left(2^{O(T)}\right) \quad und \quad NTime(T) \subseteq DSpace(T) \ .$$

Bevor wir zur nächsten Simulation übergehen, schieben wir eine kleine Zwischenüberlegung ein. Bei einer platzbeschränkten Turing-Maschine ist auf Anhieb nicht klar, ob sie überhaupt auf jeder Eingabe nach endlich vielen Schritten anhält. Allerdings gibt es nur endlich viele Konfigurationen, in die sie auf Eingaben der Länge n geraten kann. Für eine $S(n)$-platzbeschränkte k-Band Turing-Maschine mit einem zusätzlichen Read-only Eingabeband[46] zum Beispiel erhalten wir für die Menge $\mathcal{K}_n$ der möglichen Konfigurationen für Eingaben der Länge n die folgende Ungleichung:

$$|\mathcal{K}_n| \leq |Z| \cdot n \cdot S^k(n) \cdot |\Gamma|^{kS(n)} \ .$$

Hierbei ist $|Z|$ ist die Anzahl der Zustände, n die Anzahl der Positionen des Lesekopfes für die Eingabe, $S(n)$ die Anzahl der Positionen des Kopfes für ein Arbeitsband und $|\Gamma|^{S(n)}$ die Anzahl der Beschriftungen eines Arbeitsbandes. Da es insgesamt k Arbeitsbänder gibt gehen die Faktoren $S(n)$ und $|\Gamma|^{S(n)}$ in die angegebene Schranke in der k-ten Potenz ein. Im Falle $S(n) \geq \log n$ gilt offensichtlich:

$$|\mathcal{K}_n| = 2^{O(S(n))} \ .$$

Der folgende Satz besagt, dass es relativ platzeffiziente deterministische Simulationen nichtdeterministischer Maschinen gibt (lediglich quadratischer „blow-up“ der Platzschranke):

Satz 14.4 (von Savitch) *Es sei $S(n) \geq \log n$ eine platzkonstruierbare Funktion. Dann gilt*

$$NSpace(S) \subseteq DSpace^2(S) \ .$$

Beweis Es sei $L \in NSpace(S)$ und M eine NTM mit Platzschranke S und $L = L_M$. Wegen des 1. Bandreduktionstheorems dürfen wir voraussetzen, dass M lediglich ein Arbeitsband (und evtl. ein weiteres Read-only Eingabeband) besitzt. Wegen $S(n) \geq \log n$ existiert eine Konstante c, so dass M auf Eingaben der Länge n maximal $2^{cS(n)}$ Konfigurationen annehmen kann. Es bezeichne $K_0(w)$ die Startkonfiguration von M (bei Eingabe $w \in \Sigma^n$), K_+ eine oBdA eindeutige akzeptierende Endkonfiguration und $\mathcal{K}_n$ die Menge aller auf Eingaben der Länge n denkbaren Konfigurationen (mit $|\mathcal{K}_n| \leq 2^{cS(n)}$). Die Platzkonstruierbarkeit von $S(n)$ wird im Folgenden benötigt, um alle Konfigurationen aus $\mathcal{K}_n$ systematisch (zum Beispiel in lexicographischer Ordnung) durchlaufen zu können.
Betrachte folgende Relation auf $\mathcal{K}_n$:

$$K \vdash_M^t K' :\Leftrightarrow M \text{ kann } K \text{ in maximal } t \text{ Rechenschritten nach } K' \text{ überführen.}$$

Es sei $T := 2^{cS(n)}$. Beachte, dass jede Rechnung von M mit mindestens T Schritten eine Konfiguration mehrfach durchläuft. „Zykelfreie“ Rechnungen von M auf Eingabe w dauern daher weniger als T Schritte. Somit gilt:

$$w \in L_M \Leftrightarrow K_0(w) \vdash_M^T K_+ \ .$$

[46] benötigt für sublineare Platzschranken: nur die auf den k Arbeitsbändern verwendeten Zellen werden beim Platzverbrauch gezählt.

Wir wollen aufzeigen, wie die Beziehung $K_0(w) \vdash_M^T K_+$ *deterministisch* mit Platzschranke S^2 getestet werden kann. Der Schlüssel hierzu liegt in folgendem rekursiven Ansatz:

$$K \vdash_M^{2^s} K' \Leftrightarrow \exists K'' \in \mathcal{K}_n : K \vdash_M^{2^{s-1}} K'' \wedge K'' \vdash_M^{2^{s-1}} K' \ .$$

Es bezeichne TEST eine rekursive Boolesche Prozedur, die auf Parametern K, K', s den Wert TRUE ausgibt gdw $K \vdash_M^{2^s} K'$. Für $s = 0$ kann diese Prozedur leicht ausgewertet werden, da $K \vdash_M^{2^0} K'$ bedeutet, dass K' eine direkte Folgekonfiguration von K bezüglich der NTM M ist. Für $s \geq 1$ kann TEST rekursiv ausgewertet werden:

$$\text{TEST}(K, K', s) := \bigvee_{K'' \in \mathcal{K}_n} \text{TEST}(K, K'', s-1) \wedge \text{TEST}(K'', K', s-1) \ .$$

Um $w \in L$ zu testen, starten wir den Aufruf $\text{TEST}(K_0(w), K_+, cS(n))$. Es resultiert ein Rekursionsbaum der Tiefe $cS(n)$.
Wir haben abschließend die Frage zu klären, wie eine S^2-platzbeschränkte DTM M' den Prozeduraufruf $\text{TEST}(K_0(w), K_+, cS(n))$ (und die dadurch ausgelösten rekursiven Aufrufe) implementieren kann. Die wesentliche Idee ist, den Rekursionsbaum niemals vollständig aufs Band zu schreiben, sondern immer nur den aktuellen „Ariadne-Faden“.[47] Dies sind die Informationen, die einem Pfad im Rekursionsbaum von der Wurzel (erster Aufruf von TEST) bis zum aktuellen Knoten (aktueller Aufruf von TEST) entsprechen. Zur Speicherung des Ariadne-Fadens verwendet M' eines ihrer Bänder als Kellerspeicher. Zu jedem neuen Knoten auf dem Ariadne-Faden (neuer rekursiver Aufruf $\text{TEST}(K, K', s)$) wird dabei ein neuer Informationsblock (genannt $\text{FRAME}(K, K', s)$) in den Keller gepusht. Wir werden weiter unten sehen, dass ein FRAME die Länge $O(S(n))$ besitzt. Da der Ariadne-Faden maximal aus $cS(n)$ Knoten besteht (dies ist die Tiefe des Rekursionsbaumes) ergibt sich somit die Platzschranke S^2.
Bleibt zu zeigen, dass eine FRAME-Größe von $O(S(n))$ Zellen ausreicht, um die Rekursion am Laufen zu halten. Beim Aufruf $\text{TEST}(K, K', s)$ werden die folgenden Informationen in den Keller gepusht:

- die aktuellen Eingabeparameter $K, K' \in \mathcal{K}_n$ und $s \in \{1, \ldots, cS(n)\}$
- den aktuellen Wert der lokalen „Laufvariable“ $K'' \in \mathcal{K}_n$ (von der wir annehmen, dass sie die Konfigurationen aus $\mathcal{K}_n$ in lexicographischer Ordnung durchläuft)
- die Boolesche Variable SUCCESS (initialisiert auf FALSE) zur Speicherung des Ergebniswertes des ersten rekursiven Aufrufs $\text{TEST}(K, K'', s-1)$

Wir bezeichnen das Bandsegment mit diesen Informationen mit $\text{FRAME}(K, K', s)$. Offensichtlich reichen $O(S(n))$ Zellen für einen FRAME aus.
Zu jeder festen Zwischenkonfiguration $K'' \in \mathcal{K}_n$ verfährt M' wie folgt:

[47] der Sage nach erstmals von Ariadne angewendet, um Theseus davor zu schützen, sich im Labyrinth des Minotaurus zu verirren

1. Der erste rekursive Aufruf TEST$(K, K'', s-1)$ wird getätigt (d.h., FRAME$(K, K'', s-1)$ wird angelegt).

2. Bei erfolgloser Rückkehr aus Aufruf 1 wird K'' lexicographisch inkrementiert (falls möglich) und zu Schritt 1 zurück gegangen. Falls jedoch K'' bereits die lexicographisch letzte Konfiguration in $\mathcal{K}_n$ war (keine weitere Inkrementierung möglich), wird der Aufruf TEST(K, K', s) erfolglos terminiert, d.h., M' löscht FRAME(K, K', s) und kehrt erfolglos in den darunter liegenden FRAME (sofern vorhanden) zurück.

3. Bei erfolgreicher Rückkehr aus Aufruf 1 wird SUCCESS auf TRUE gesetzt und der zweite rekursive Aufruf TEST$(K'', K', s-1)$ getätigt.

4. Bei erfolgloser Rückkehr aus Aufruf 2 wird SUCCESS auf FALSE zurück gesetzt und ansonsten so vorgegangen wie bei der erfolglosen Rückkehr aus Aufruf 1.

5. Bei erfolgreicher Rückkehr aus Aufruf 2 wird der Aufruf TEST(K, K', s) erfolgreich terminiert, d.h., M' löscht FRAME(K, K', s) und kehrt erfolgreich in den darunter liegenden FRAME zurück.

Bei dieser Vorgehensweise merkt sich M' in der endlichen Kontrolle, ob sie sich in einer ADD- (neuer rekursiver Aufruf, neuer FRAME) oder DELETE-Phase (Rückkehr aus einem Aufruf, FRAME löschen) befindet. In einer DELETE-Phase merkt sie sich zudem, ob sie aus dem Aufruf des zuletzt gelöschten FRAME's erfolgreich oder erfolglos zurück kehrt.
Wenn M' irgendwann ihren Keller vollständig geleert hat, wurde gerade der FRAME des initialen Aufrufes TEST$(K_0(w), K_+, cS(n))$ gelöscht. Zu diesem Zeitpunkt weiß M' in der endlichen Kontrolle, ob dieser Aufruf erfolgreich war. Sie akzeptiert die Eingabe w gdw dies der Fall ist.
Aus unserer Diskussion geht hervor, dass $w \in L$ von M' korrekt getestet und dass die Platzschranke S^2 respektiert wird. •

Satz 14.5 *Es sei $S(n) \geq \log n$ eine platzkonstruierbare Funktion. Dann gilt*

$$\mathit{NSpace}(S) \subseteq \mathit{DTime}(2^{O(S)}) \ .$$

Beweis Es sei $L \in \mathit{NSpace}(S)$ und M eine $S(n)$-platzbeschränkte NTM mit $L = L_M$. Konstante $c > 0$, $K_0(w)$, K_+ und $\mathcal{K}_n$ mit $|\mathcal{K}_n| \leq 2^{cS(n)}$ seien gewählt wie im Beweis zum Satz von Savitch. Wir betrachten diesmal $\mathcal{K}_n$ als die Knotenmenge eines Digraphen $G_M(w)$, wobei wir eine gerichtete Kante von K nach K' ziehen, wenn K' eine direkte Folgekonfiguration von K bezüglich der NTM M (bei Eingabe w) ist. Offensichtlich gilt

$$w \in L_M \Leftrightarrow \text{Es gibt in } G_M(w) \text{ einen Pfad von } K_0(w) \text{ nach } K_+.$$

Der Digraph $G_M(w)$ hat maximal $2^{cS(n)}$ Knoten und maximal $(2^{cS(n)})^2 = 2^{2cS(n)}$ Kanten. Erreichbarkeitsprobleme, bei denen nach Pfadverbindungen zwischen zwei Knoten gefragt

wird, sind mit einfachen Graphexplorationstechniken zeiteffizient lösbar. Eine DTM benötigt bei Digraphen der Größe N hierfür maximal $O(N^k)$ Schritten (für eine geeignete Konstante k). Wegen $(2^{2cS(n)})^k = 2^{2kcS(n)}$ ergibt sich die Zeitschranke $2^{O(S)}$. •

Der im Beweis verwendete Digraph $G_M(w)$ heißt der *Konfigurationsgraph* von M.

14.3 Abschluss unter Komplement

Zu einer Komplexitätsklasse K assoziieren wir die Klasse

$$\text{co-}K := \{\bar{L} |\ L \in K\}$$

aller Komplemente von Sprachen aus K. $K = \text{co-}K$ bedeutet dann nichts anderes als dass K unter Komplement abgeschlossen ist.

Deterministische Zeitkomplexitätsklassen sind trivialerweise abgeschlossen unter Komplement:

Satz 14.6 *$DTime(T(n)) = co\text{-}DTime(T(n))$.*

Beweis Vertauschen von Endzuständen und Nicht-Endzuständen macht aus einem $T(n)$-zeitbeschränkten Akzeptor von L einen $T(n)$-zeitbeschränkten Akzeptor von $\bar{L} = \Sigma^* \setminus L$. •

Beachte, dass die entsprechende Aussage für *nichtdeterministische* Zeitkomplexitätsklassen vermutlich nicht gilt. Zum Beispiel gilt ja vermutlich $NP \neq \text{co-}NP$.

Wie verhalten sich Platzkomplexitätsklassen bezüglich Komplementbildung. Die folgenden Resultate liefern eine Antwort.

Satz 14.7 *Für jede platzkonstruierbare Funktion $S(n) \geq \log n$ gilt:*
$DSpace(S(n)) = co\text{-}DSpace(S(n))$.

Beweis Wenn ein $S(n)$-platzbeschränkter Akzeptor von L auf jeder Eingabe nach endlich vielen Schritten stoppt, dann können wir erneut mit dem Vertauschen von Endzuständen und Nicht-Endzuständen argumentieren. Durch etwaige Endlosschleifen ensteht aber ein Zusatzproblem. Andererseits wissen wir bereits, dass ein $S(n)$-platzbeschränkter Akzeptor M nur $2^{O(S(n))}$ Konfigurationen hat. Es gibt daher eine Konstante k mit der Eigenschaft: nach $2^{kS(n)}$ Schritten muss M sich in einer Endlosschleife befinden. Wir können die Platzkonstruierbarkeit von $S(n)$ und damit auch $kS(n)$ benutzen, um einen Binärzähler mit $10^{kS(n)}$, also der Binärdarstellung von $2^{kS(n)}$, zu installieren. Dieser Binärzähler kann als eine Art „Uhr“ rückwärts gezählt werden. Wenn die Uhr auf 0 steht, brechen wir die Rechnung nichtakzeptierend ab. •

Satz 14.7 gilt sogar *ohne* die Voraussetzung der Platzkonstruierbarkeit von $S(n)$. Den Nachweis hierfür empfehlen wir als **Übung**. **Übg.**

Für nichtdeterministische Zeitkomplexitätsklassen ist der Abschluss unter Komplement vermutlich nicht gegeben. (Zum Beispiel wird $NP \neq$ co-NP vermutet.) Für nichtdeterministische Platzkomplexitätsklassen hingegen konnten Immerman und Szelepcsényi folgendes Resultat beweisen:

Satz 14.8 *Für jede platzkonstruierbare Funktion $S(n) \geq \log n$ gilt:*

$$\mathit{NSpace}(S(n)) = \mathit{co\text{-}NSpace}(S(n)) \ .$$

Dieser Satz wird in der Vorlesung „Theoretische Informatik“ für den Spezialfall $S(n) = n$ bewiesen:

- $\mathit{NSpace}(n)$ stimmt überein mit der Klasse der kontextsensitiven Sprachen.
- In der Vorlesung „Theoretische Informatik“ wird gezeigt, dass kontextsensitive Sprachen abgeschlossen sind unter Komplement.

Da der Beweis für den allgemeinen Fall nicht wesentlich aufwändiger ist als für den Spezialfall $S(n) = n$, wollen wir ihn im Rahmen der Vorlesung „Komplexitätstheorie“ nicht im Detail führen, sondern in der folgenden Beweisskizze nur die wesentlichen Ideen ins Gedächtnis rufen:

Beweisskizze

- Für eine Sprache $L \in \mathit{NSpace}(S(n))$ ist zu zeigen $\bar{L} \in \mathit{NSpace}(S(n))$. Es sei M eine $S(n)$-platzbeschränkte NTM, die L erkennt. Es sei $G_M(w)$ wieder der Konfigurationsdigraph von M bei Eingabe w, $K_0(w)$ die Startkonfiguration bei Eingabe w und K_+ eine (oBdA) eindeutige akzeptierende Endkonfiguration. Eine NTM $\bar{M}$ für die Sprache $\bar{L}$ muss für jedes Wort $w \in \bar{L}$ nichtdetermistisch verifizieren können, dass in $G_M(w)$ *kein* Pfad von $K_0(w)$ nach K_+ existiert. Ein Pfad ließe sich durch Raten desselben verifizieren. Das Rätsel lautet: wie lässt sich die Nicht-Existenz eines Pfades verifizieren?
- Angenommen wir wüssten, wieviele Knoten in $G_M(w)$ von $K_0(w)$ aus erreichbar sind, sagen wir $N = 2^{O(S(n))}$ viele. Dann können wir die Nicht-Erreichbarkeit von K_+ dadurch verifizieren, dass wir zu N von K_+ verschiedenen erreichbaren Knoten K jeweils einen Pfad von $K_0(w)$ nach K raten.
- Die Anzahl N der von $K_0(w)$ aus erreichbaren Knoten lässt sich mit der Technik des (nichtdeterministischen) induktiven Zählens bestimmen. Dabei wird Folgendes erreicht:
  - Entweder $\bar{M}$ bestimmt N korrekt oder $\bar{M}$ bricht den Zählprozess erfolglos ab.
  - Es existiert mindestens eine Rechnung von $\bar{M}$, die N korrekt bestimmt.

Weitere Details zum induktiven Zählen können im Skriptum zur Vorlesung „Theoretische Informatik“ nachgelesen werden. Beachten Sie, dass ein Binärzähler, der bis $N = 2^{O(S(n))}$ zählen kann, nur $O(S(n))$ Zellen beansprucht. Bei geschickter Implementierung von $\bar{M}$ ergibt sich die Platzschranke $S(n)$, womit dann $\bar{L} \in \mathit{NSpace}(S(n))$ bewiesen wäre. •

14.4 Die Platz-Zeit-Hierarchie

Aus den vorangegangenen Abschnitten ergibt sich die folgende Situation für eine platzkonstruierbare Ressourcenschranke R:

$$\begin{aligned} DSpace(R) \subseteq NSpace(R) \subseteq DTime\left(2^{O(R)}\right) &\subseteq NTime\left(2^{O(R)}\right) \\ &\subseteq NSpace\left(2^{O(R)}\right) = DSpace\left(2^{O(R)}\right) \ . \end{aligned}$$

Bei der letzten Gleichung wurde der Satz von Savitch benutzt.

Wir erinnern an die Komplexitätsklassen $\mathcal{L} = DSpace(\log n)$, $\mathcal{NL} = NSpace(\log n)$, P, NP und $PSpace$. Mit $R(n) = \log n$ ergibt sich (unter Beachtung von $2^{O(\log n)} = n^{O(1)}$) die

Folgerung 14.9 $\mathcal{L} \subseteq \mathcal{NL} \subseteq P \subseteq NP \subseteq PSpace$.

15 Hierarchiesätze

Im Folgenden bezeichne $DTime_k(T(n))$ die Klasse aller Sprachen, die von einer $T(n)$-zeitbeschränkten k-Band DTM erkannt werden können. Da die Definition der Klasse $DTime(T(n))$ keine Vorschriften über die (konstante) Anzahl von Bändern macht, ergibt sich $DTime(T(n)) = \cup_{k\geq 1} DTime_k(T(n))$. Die Notationen $DSpace_k(S(n))$, $NTime_k(T(n))$ und $NSpace_k(T(n))$ sind analog zu verstehen. Die uns von früher bekannten Bandreduktionstheoreme, resultierend aus Simulationen von k-Band DTMs auf 1- oder 2-Band DTMs, lesen sich dann wie folgt:

$$\begin{aligned} DSpace(S(n)) &= DSpace_1(S(n)) \\ DTime(T(n)) &\subseteq DTime_1(T(n)^2) \\ DTime(T(n)) &\subseteq DTime_2(T(n)\log T(n)) \end{aligned}$$

Die entsprechenden Aussagen für NTMs lauten:

$$\begin{aligned} NSpace(S(n)) &= NSpace_1(S(n)) \\ NTime(T(n)) &= NTime_2(T(n)) \end{aligned}$$

Wir werden in diesem Abschnitt zeigen, dass die (asymptotische) Vergrößerung einer Ressourcenschranke zu einer Vergrößerung der Klasse der mit diesen Ressourcen erkennbaren Sprachen führt. Aussagen dieser Art sind unter dem Namen „Hierarchiesätze“ bekannt. Wir beginnen mit folgendem

Lemma 15.1 *Es seien $f_1(n), f_2(n)$ Funktionen mit $f_1(n) = o(f_2(n))$*

1. *Es sei f_2 zeitkonstruierbar. Dann gilt für alle $k \geq 1$:*

$$DTime_k(f_1(n)) \subset DTime_{k+1}(f_2(n)) \ .$$

2. Es sei $f_2(n) \geq \log n$ platzkonstruierbar. Dann gilt für alle $k \geq 1$:

$$DSpace_k(f_1(n)) \subset DSpace_{k+1}(f_2(n)) \ .$$

Beweis Die Inklusionsbeziehung ist klar, aber die Echtheit der Inklusion muss jeweils nachgewiesen werden. Der Beweis benutzt die Technik der Diagonalisierung. Wie aus Abschnitt 2.7 bekannt ist, können wir eine k-Band DTM M_α mit einer k-Band UTM simulieren, die einen Schritt von M_α simulieren kann, und zwar in $O(|\alpha|)$ Schritten im Falle $k \geq 2$ und in $O(|\alpha|^2)$ Schritten im Falle $k = 1$. In diesem Beweis spendieren wir der UTM ein Band $k+1$, das der Kontrolle der Ressourcenschranke dient. Simulationen, welche zur Überschreitung der Ressourcenschranke (oder zu Endlosschleifen) führen würden, werden abgebrochen.[48] Wir definieren nun eine Sprache D wie folgt: ein Eingabewort $\alpha 01^p$ wird in D *nicht* aufgenommen gdw die UTM-Simulation zum Akzeptieren von $\alpha 01^p$ durch M_α führt. Umgekehrt ausgedrückt: $\alpha 01^p$ wird in D aufgenommen gdw entweder die UTM-Simulation zum *Nicht-Akzeptieren* von $\alpha 01^p$ durch M_α führt oder die Simulation vorzeitig abgebrochen werden musste. Die Sprache D hat eine geeignete Variante der UTM als Akzeptor. Da per Konstruktion Ressourcenüberschreitungen vermieden werden, gehört D zu $DTime(f_2(n))$ bzw. zu $DSpace(f_2(n))$. Wegen $f_1(n) = o(f_2(n))$ gibt es für jede k-Band DTM M_α mit Ressourcenschranke f_1 einen hinreichend grosen „Polsterparameter“ p, so dass akzeptierende Rechnungen von M_α ohne Resourcenüberschreitung zu Ende simuliert werden können. Offensichtlich (per Diagonalisierung) unterscheidet sich D in mindestens einem Wort von jeder Sprache aus $DTime(f_1(n))$ bzw. aus $DSpace(f_1(n))$. Hieraus ergibt sich die Aussage des Lemmas. •

Folgerung 15.2 *Es sei $S_2(n)$ platzkonstruierbar und $S_1(n) = o(S_2(n))$. Dann gilt*

$$DSpace(S_1(n)) \subset DSpace(S_2(n)) \ .$$

Beweis Der Beweis ergibt sich unter Ausnutzung von Lemma 15.1 und der Bandreduktionstheoreme aus

$$DSpace(S_1(n)) = DSpace_1(S_1(n)) \subset DSpace_2(S_2(n)) = DSpace(S_2(n)) \ .$$

•

Folgerung 15.3 *Es sei $T_2(n)$ zeitkonstruierbar und $T_1(n) \log T_1(n) = o(T_2(n))$. Dann gilt*

$$DTime(T_1(n)) \subset DTime(T_2(n)) \ .$$

[48] Endlosschleifen könnten bei platzbeschränkten Rechnungen auftreten. Da $S_2(n) \geq \log n$ platzkonstruierbar ist, können Endlosschleifen des Simulators aber durch Mitzählen der durchlaufenen Konfigurationen erkannt werden.

Beweis Der Beweis ergibt sich unter Ausnutzung von Lemma 15.1 und der Bandreduktionstheoreme aus

$$DTime(T_1(n)) \subseteq DTime_2(T_1(n) \log T_1(n)) \subset DTime_3(T_2(n)) \subseteq DTime(T_2) \ .$$

•

Die in Lemma 15.1 angewendete Diagonalisierungstechnik benutzt drei Grundeigenschaften der beteiligten Komplexitätsklassen:

Universalität: Die Simulation verwendet eine f_2-ressourcenbeschränkte UTM zur Simulation von Maschinen mit Ressourcenschranke f_1.

Kontrollierbarkeit: Ressourcenüberschreitungen werden erkannt und führen zum Abbruch der Simulation.

Abschluss unter Komplement: Die UTM muss die Akzeptanzentscheidung von M_α effizient negieren.

Die erten beiden Eigenschaften sind auch bei *nichtdeterministischen* Komplexitätsklassen gegeben. Gemäß dem Satz von Immerman und Szelepcsényi (s. Satz 14.8) liegt die dritte Eigenschaft wenigstens für nichtdeterministische Platzkomplexitätsklassen vor. Daher ergibt sich (mit Beweisen analog zum deterministischen Fall) die

Folgerung 15.4 *Es sei $S_2(n)$ platzkonstruierbar und $S_1(n) = o(S_2(n))$. Dann gilt*

$$NSpace(S_1(n)) \subset NSpace(S_2(n)) \ .$$

Für nichtdeterministische Zeitkomplexitätsklassen ist der Abschluss unter Komplement vermutlich nicht gegeben. (Zum Beispiel wird $NP \neq$ co-NP vermutet.) Erstaunlicherweise lässt sich dennoch der folgende Zeithierarchiesatz beweisen:

Satz 15.5 *Es sei T_2 zeitkonstruierbar und $T_1(n) \leq T_1(n+1) = o(T_2(n))$. Dann gilt für alle $k \geq 1$: $NTime_k(T_1) \subset NTime_{k+1}(T_2)$.*

Beweis Der Beweis benutzt die Technik der „lazy diagonalization" zur Konstruktion einer Sprache $D \in NTime_{k+1}(T_2(n)) \setminus NTime_k(T_1(n))$. Viele technische Details sind ähnlich wie im Beweis von Lemma 15.1, so dass wir uns hier auf die zusätzlichen Manöver konzentrieren, die der fehlende Abschluss unter Komplement uns aufnötigt. Es sei $f : \mathbb{N} \to \mathbb{N}$ die folgende induktiv definierte Funktion:

$$f(1) = 2 \ \text{ und } \ f(i+1) = 2^{T_2(f(i))}$$

Es ist nicht schwer zu zeigen, dass sich zu jedem $n \in \mathbb{N}$ der (wegen der strengen Monotonie von f eindeutige) Index i mit

$$f(i) < n \leq f(i+1)$$

von einer $T_2(n)$-zeitbeschränkten DTM berechnen lässt. Wir erinnern daran, dass wir mit $\alpha(i)$ den i-ten Binärstring in der natürlichen Aufzählung aller Binärstrings bezeichnen. Im Rahmen der „lazy diagonalization“ soll für jede $T_1(n)$-zeitbeschränkte NTM M Folgendes erreicht werden: wenn $\alpha(i) = \alpha 01^p$ ein Codewort für M mit hinreichend großer Polsterung p ist, dann soll D sich von $L(M)$ auf einem Wort der Form 1^n mit $f(i) < n \leq f(i+1)$ unterscheiden. Hierzu gehen wir bei gegebener Eingabe 1^n vor wie folgt:

Fall 1: $f(i) < n < f(i+1)$.
Dann simuliere $M = M_{\alpha(i)}$ nichtdeterministisch auf Eingabe 1^{n+1} und nimm 1^n in D auf gdw die simulierte Rechnung zum Akzeptieren führt (*keine* Abänderung des Akzeptanzverhaltens). Beachte, dass die Simulation wegen der Voraussetzung $T_1(n+1) = o(T_2(n))$ im Rahmen der Zeitschranke $T_2(n)$ geleistet werden kann.

Fall 2: $n = f(i+1)$.
Dann simuliere $M = M_{\alpha(i)}$ *deterministisch* auf Eingabe $1^{f(i)+1}$ und nimm 1^n in D auf gdw die simulierte Rechnung *nicht* alzeptiert. Da M auf Eingaben der Länge $f(i)+1$ maximal $cT_1(f(i)+1)$ Schritte rechnet (für eine geeignet gewählte Konstante c), gibt es im Konfigurationsgraphen von M maximal $2^{cT_1(f(i)+1)} = 2^{o(T_2(f(i)))}$ Berechnungspfade und die deterministische Simulation kann von einer $2^{o(T_2(f(i)))}$-zeitbeschränkten DTM erledigt werden. Wegen $f(i+1) = 2^{T_2(f(i)}$ ist dann erst recht die Zeitschranke $T_2(f(i+1)) = T_2(n)$ ausreichend.[49]

Wir argmentieren nun, dass D sich von M (hinreichend große Polsterung des Codeworts von M unterstellt) in mindestens einem der Strings 1^n, $f(i) < n \leq f(i+1)$, unterscheidet. Beachte, dass gemäß der obigen Fallunterscheidung

$$\forall n \in \{f(i)+1, \ldots, f(i+1)-1\} : \; D(1^n) = M(1^{n+1}) \;\text{ und }\; D(1^{f(i+1)}) \neq M(1^{f(i)+1}) \; . \quad (21)$$

Wir machen die (heuchlerische) Annahme, dass

$$\forall n \in \{f(i)+1, \ldots, f(i+1)\} : \; D(1^n) = M(1^n) \; . \quad (22)$$

Bedingung (21) kombiniert mit (22) führt dann aber zu einem Widerspruch. Daher ist die Annahme (22) falsch und D unterscheidet sich für mindestens einen Index n auf 1^n von M — wie gewünscht. •

Folgerung 15.6 *Es sei $T_1(n)$ und $T_2(n)$ zeitkonstruierbar mit $T_1(n) \leq T_1(n+1) = o(T_2(n))$. Dann gilt: $NTime(T_1) \subset NTime(T_2)$.*

Beweis Wir erinnern an Satz th:k-auf-2-band, welcher besagt, dass

$$NTime(T(n)) = NTime_2(T(n)) \; .$$

[49] Da in beiden diskutierten Fällen Asymptotik im Spiel ist, stimmt die vorgetragene Argumentation, dass Zeitschranke $T_2(n)$ für die Simulation ausreicht, streng genommen erst bei hinreichend großer Polsterung p (was das Diagonalisierungsargument aber nicht beschädigt).

In Kombination mit Satz 15.5 ergibt sich

$$NTime(T_1(n)) = NTime_2(T_1(n)) \subset NTime_3(T_2(n)) = NTime(T_2(n)) \ ,$$

was den Beweis abschließt. •

Offenes Problem: Wie erinnern an die Platz-Zeit-Hierarchie

$$\mathcal{L} \subseteq \mathcal{NL} \subseteq P \subseteq NP \subseteq PSpace \tag{23}$$

Aus dem nichtdeterministischen Platzhierarchiesatz folgt $\mathcal{NL} \subset PSpace$. Demnach muss die Inklusionskette (23) an irgendeiner Stelle zerreißen, d.h., mindestens eine der Inklusionen muss echt sein. Es wird vermutet, dass *alle* Inklusionen in (23) echt sind. Aber von keiner einzelnen konnte dies bis dato bewiesen werden.

16 Die Klasse NL

Ein Verwandter des P–NP Problems ist das $\mathcal{L}$–$\mathcal{NL}$ Problem: Ist die Inklusion $\mathcal{L} \subseteq \mathcal{NL}$ echt? Da dieses Problem trotz großer Anstrengungen bis heute nicht gelöst werden konnte, ist es naheliegend, nach „schwersten Problemen“ in $\mathcal{NL}$ zu suchen. Diese können, in Analogie zur NP-Vollständigkeitstheorie, wieder mit Hilfe von Problemreduktionen dingfest gemacht werden. Allerdings ist es nicht zweckmäßig, Karp-Reduktionen zu verwenden: aus $L_1 \leq_{pol} L_2$ und $L_2 \in \mathcal{L}$ folgt nämlich *nicht*, dass $L_1 \in \mathcal{L}$, da die zugrunde liegende Reduktionsabbildung zwar in Polynomialzeit, aber nicht notwendig mit logarithmischem Platz, berechnet werden kann. Die für die Berechnung der Reduktionsabbildung zur Verfügung gestellten Ressourcen sollten sich stets an der kleineren der beiden beteiligten Komplexitätsklassen orientieren, in unserem Fall also an der Klasse $\mathcal{L}$. Dies führt uns zum Begriff der logspace-Reduktionen:

Definition 16.1 *Eine Sprache L_1 heißt* logspace-reduzierbar *auf L_2, notiert als $L_1 \leq_{log} L_2$, falls eine Abbildung $f : \Sigma^* \to \Sigma^*$ existiert mit:*

1. *f ist* logspace-berechenbar*, d.h., f ist von einer $log(n)$-platzbeschränkten DTM, versehen mit einem Read-only Eingabeband, einem Write-only Ausgabeband sowie einem Arbeitsband, berechenbar. Dabei wird nur der Platzverbrauch auf dem Arbeitsband gezählt.*

2. *Für alle $x \in \Sigma^*$ gilt: $x \in L_1 \Leftrightarrow f(x) \in L_2$.*

Da k-Band DTMs ohne Verlust an Platzeffizienz von 1-Band DTMs simuliert werden können, ändert sich der Begriff der logspace-Berechenbarkeit von f nicht, wenn wir der betreffenden DTM k Arbeitsbänder (statt nur einem) spendieren.
Die Relation „$\leq_{log}$“ besitzt die gewünschten Eigenschaften:

Lemma 16.2 *1. Aus $L_1 \leq_{log} L_2$ und $L_2 \leq_{log} L_3$ folgt $L_1 \leq_{log} L_3$.*

2. *Aus* $L_1 \leq_{log} L_2$ *und* $L_2 \in \mathcal{L}$ *folgt* $L_1 \in \mathcal{L}$.

Beweis

1. Es sei f die Reduktionsabbildung zu $L_1 \leq_{log} L_2$ und g die Reduktionsabbildung zu $L_2 \leq_{log} L_3$. Wir möchten zeigen, dass $g \circ f$ eine Reduktionsabbildung zu $L_1 \leq_{log} L_3$ ist. Freilich gilt für alle $x \in \Sigma^*$

$$x \in L_1 \Leftrightarrow f(x) \in L_2 \Leftrightarrow g(f(x)) \in L_3 \ .$$

   Aber ist $g \circ f$ logspace-berechenbar? Da $y = f(x)$ einerseits die Ausgabe der f-Berechnung, andererseits aber die Eingabe der g-Berechnung ist, scheint es folgendes Dilemma zu geben:

   - Wenn die Ausgabe y zur ersten Reduktion auf das Write-only Ausgabeband geschrieben wird, dann darf die Prozedur, die $g(y) = g(f(x))$ berechnen soll, ihre Eingabe y nicht lesen.
   - Logarithmischer Speicher reicht aber nicht aus, um $f(x)$ in Gänze auf das Arbeitsand zu schreiben. Schlimm, schlimm, ...

   Der Ausweg aus dem Dilemma ist wie folgt. Wir halten eine Simulation der DTM M_g zur Berechnung von $g(f(x))$ aufrecht und merken uns in einem Zähler die aktuelle Position des Read-only Eingabekopfes von M_g (anfangs die Position 1). Wenn M_g das j-te Symbol der (nicht vorhandenen!) Eingabe $f(x)$ lesen möchte, starten wir die DTM M_f zur Berechnung von $f(x)$, unterdrücken die Ausgabe der ersten $j-1$ Bits (unter Verwendung eines zweiten Zählers) und benutzen das j-te Ausgabebit, um den nächsten Berechnungsschritt von M_g zu simulieren. Die Details dieser Simulation sind leicht auszuarbeiten.

2. Es sei f die Reduktionsabbildung zu $L_1 \leq_{log} L_2$. L_1 kann im Rahmen einer logspace-Berechnung erkannt werden, indem wir die DTM zur Berechnung von $f(x)$ und die DTM, welche „$f(x) \in L_2$?“ entscheidet, kooperieren lassen. Das Eingabe/Ausgabe-Dilemma wird wieder mit dem soeben geschilderten Trick gelöst.

•

Dieser Beweis macht etwas deutlich, was wir schon häufiger beobachtet haben: ein Markenzeichen von „logspace“ ist, dass der zur Verfügung stehende Platz (gerade soeben) ausreicht, um einen (oder konstant viele) Zähler zu speichern, die von 0 bis poly(n) zählen können.

Gegeben unsere Erfahrung mit der NP-Vollständigkeitstheorie ist die folgende Definition und das folgende Lemma „Standard“:

Definition 16.3 *Eine Sprache* L_0 *heißt* $\mathcal{NL}$-hart, *falls* $L \leq_{log} L_0$ *für alle* $L \in \mathcal{NL}$. *Falls darüber hinaus* $L_0 \in \mathcal{NL}$, *dann heißt* L_0 $\mathcal{NL}$*-vollständig.*

Lemma 16.4 *Es gelte $L_1 \leq_{log} L_2$ und L_1 sei $\mathcal{N}L$-hart. Dann ist auch L_2 $\mathcal{N}L$-hart.*

Ausgehend von einem $\mathcal{N}L$-harten Problem können wir mit Hilfe von logspace-Reduktionen nach und nach einen Stammbaum $\mathcal{N}L$-harter Probleme „wachsen lassen“. Wie gelangen wir an ein erstes $\mathcal{N}L$-hartes Problem? Voilà, hier ist es:

Digraph Reachability: Gegeben ein Digraph $G = (V, E)$ mit zwei ausgezeichneten Koten $s, t \in V$, gibt es einen Pfad in G von s nach t?

Satz 16.5 *„Digraph Reachability“ ist $\mathcal{N}L$-vollständig.*

Beweis Die Mitgliedschaft von „Digraph Reachability“ in der Klasse $\mathcal{N}L$ ergibt sich wie folgt: rate einen Pfad von s nach t und durchlaufe ihn dabei, speichere aber auf dem Arbeitsband immer nur den Knoten, an dem der Pfaddurchlauf sich aktuell befindet.
Es sei $L \in \mathcal{N}L$ beliebig aber fest. Weiter sei M eine $log(n)$-platzbeschränkte NTM, die L erkennt. Zu einer Eingabe w sei $G_M(w)$ der resultierende Konfigurationsgraph mit ausgezeichneten Knoten (=Konfigurationen) $K_0(w)$ (Startkonfiguration) und K_+ (akzeptierende Endkonfiguration). Die Abbildung

$$w \mapsto G_M(w), K_0(w), K_+$$

hat die Eigenschaft

$$w \in L \Leftrightarrow \text{es gibt in } G_M(w) \text{ einen Pfad von } K_0(w) \text{ nach } K_+ \ ,$$

und es ist nicht schwer zu sehen, dass sie logspace-berechenbar ist.
Unsere Diskussion hat ergeben, dass „Digraph Reachability“ $\mathcal{N}L$-vollständig ist. •

Da logspace-Rechnungen in Polynomialzeit durchgeführt werden können, gilt: $L_1 \leq_{log} L_2 \Rightarrow L_1 \leq_{pol} L_2$. Wir merken kurz an, dass alle von uns bisher vollzogenen Karp-Reduktionen mit etwas Liebe und Sorgfalt in logspace-Reduktionen transformiert werden können. Insbesondere sind SAT, 3-SAT sowie alle weiteren Sprachen des von uns bisher entwickelten Stammbaums NP-vollständig unter logspace-Reduktionen. Aus der Existenz eines $\log(n)$-platzbeschränkten Algorithmus für SAT (oder 3-SAT, ...) würde also $\mathcal{L} = NP$ folgen!

Sprachen aus NP sind dadurch charakterisiert, dass es für jedes Wort $x \in L$ einen kurzen und zeiteffizient verifizierbaren *Beweis* (auch *Zertifikat* genannt) y gibt, der die Mitgliedschaft von x in L belegt. Sprachen aus $\mathcal{N}L$ sind in ähnlicher Weise durch kurze und platzeffizient verifizierbare *Read-once Zertifikate* charakterisiert:

Definition 16.6 *Ein* Read-once Eingabeband *ist ein Read-only Eingabeband, auf welchem der Kopf in jedem Schritt eine Position nach rechts rückt. Für eine DTM M mit einem Read-only Eingabeband und einem zusätzlichen Read-once Eingabeband bezeichnet $M(x, y) \in \{0, 1\}$ die Ausgabe von M zu Eingabe x auf dem Read-only Eingabeband und Eingabe y auf dem Read-once Eingabeband („1“ = Akzeptieren).*

Satz 16.7 *Eine Sprache L ist aus $\mathcal{NL}$ gdw ein Polynom $p(n)$ und eine (auf dem Arbeitsband) $\log(n)$-platzbeschränkte DTM M existieren, so dass folgendes gilt:*

1. *M ist neben ihrem Read-only Eingabeband, ihrem Write-only Ausgabeband und ihrem Arbeitsband mit einem weiteren Read-once Eingabeband ausgestattet.*
2. *Für alle $x \in \Sigma^*$: $x \in L \Leftrightarrow \exists y \in \{0,1\}^{p(|x|)} : M(x,y) = 1$.*

In Verbindung mit Satz 16.7 heißt y auch das „Read-once Zertifikat“ für $x \in L$. Der (offensichtliche) Beweis dieses Satzes benutzt das Rate-Verifikationsprinzip, welches wir im Zusammenhang mit der Klasse NP bereits kennen gelernt haben.

Aus dem Satz von Immerman und Szelepsènyi ergibt sich die

Folgerung 16.8 *$\mathcal{NL} =$ co-$\mathcal{NL}$.*

Da $\mathcal{NL}$ abgeschlossen unter Komplement ist gehört „Digraph Unreachability“ (das Komplement von „Digraph Reachability”) ebenfalls zu $\mathcal{NL}$. In Verbindung mit Satz 16.7 heißt das, dass die *Nicht-Existenz* eines Pfades von s nach t in einem Digraphen G sich durch ein kurzes und logspace-verifizierbares Read-once Zertifikat belegen lässt.

Wir stellen abschließend die Frage, ob die in Satz 16.7 beschriebene DTM mächtiger wird, wenn das Zertifikat y auf ein zusätzliches Read-only Eingabeband geschrieben worden wäre (anstatt auf ein Read-once Eingabeband). Die Antwort, vielleicht etwas überraschend, lautet „viel mächtiger“: mit DTMs von diesem Zuschnitt lassen sich exakt die Sprachen der Klasse NP erkennen. Den Beweis dieser Aussage empfehlen wir als **Übung**. **Übg.**

17 PSpace-vollständige Probleme

In Abschnitt 17.1 lernt die Leserin und der Leser ein erstes PSpace-vollständiges Problem kennen: die Sprache TQBF der „wahren quantifizierten Booleschen Formeln (True Quantified Boolean Formulae)“. Dies führt uns zu einer Verbindung zwischen Komlexitätstheorie und Prädikatenlogik, welche wir in Abschnitt 17.2 näher beleuchten. Abschnitt 17.3 soll einen Eindruck vermittel, welche Typen von Problemen (neben TQBF) schwerste Vertreter in der Klasse PSpace sind. Das Kapitel schließt in Abschnitt 17.4 mit der prädikatenlogischen Version offener komplexitätstheoretischer Probleme.

17.1 Quantifizierte Boolesche Formeln

Definition 17.1 *Die Sprache der* Wahren Quantifizierten Booleschen Formeln*, notiert als TQBF, enthalte alle (Kodierungen von) quantifizierten Booleschen Formeln der Form*

$$(\mathcal{Q}_1 v_1) \dots (\mathcal{Q}_m v_m) F(v_1, \dots, v_m) \ , \tag{24}$$

welche die Korrektheitsbedingung

$$\mathcal{Q}_1 a_1 \in \{0,1\}, \dots, \mathcal{Q}_m a_m \in \{0,1\} : F(a_1, \dots, a_m) \tag{25}$$

erfüllen. Hierbei sei m eine beliebige nicht-negative ganze Zahl, $\mathcal{Q}_1, \ldots, \mathcal{Q}_m \in \{\exists, \forall\}$, und F sei eine Boolesche Formel in den Booleschen Variablen $v_1, \ldots, v_m$.

Beispiel 17.2 *Folgende quantifizierten Booleschen Formeln haben die in (24) vorgeschriebene Form:*

$$\begin{array}{lcll} qbf_1 & := & (\forall v_1)(\exists v_2)(\exists v_3) & \bar{v}_3 \wedge ((\bar{v}_1 \wedge v_2) \vee (v_1 \wedge \bar{v}_2)) \\ qbf_2 & := & (\exists v_1)(\forall v_2)(\exists v_3) & \bar{v}_3 \wedge ((\bar{v}_1 \wedge v_2) \vee (v_1 \wedge \bar{v}_2)) \end{array}$$

Wir werden weiter unten sehen, dass qbf_1 die Bedingung (25) erfüllt und somit zu TQBF gehört, wohingegen qbf_2 diese Bedingung verletzt und somit nicht zu TQBF gehört.

Satz 17.3 *TQBF $\in$ PSpace.*

Beweis Wir verwenden zunächst eine platzineffiziente EXHAUSTIVE SEARCH, um ein kanonisches Auswertungsschema klarzumachen. Anschließend verwenden wir die (uns schon aus dem Beweis des Satzes von Savich bekannte) Technik des Ariadne Fadens, um die EXHAUSTIVE SEARCH platzeffizient zu gestalten.
Wir gehen aus von einer Eingabe der Form (24). Mit einem vollständig binären Entscheidungsbaum T der Tiefe m lassen sich alle 2^m denkbaren Belegungen von $v_1, \ldots, v_m$ durchprobieren. Wir markieren die inneren Knoten von T der Tiefe $i-1$ mit $\mathcal{Q}_i \in \{\exists, \forall\}$, um deutlich zu machen, wie die Variable v_i quantifiziert ist.

Einschub Die Entscheidungsbäume T_1 und T_2, die zu den Formeln qbf_1 und qbf_2 aus Beispiel 17.2 korrespondieren, sind in den Abbildungen 14 und 15 zu besichtigen. Der Fortgang des Beweises kann an diesen Abbildungen nachvollzogen werden.

Jedes Blatt von T korrespondiert zu einer vollständigen Belegung $(b_1, \ldots, b_m) \in \{0,1\}^m$ der Variablen $(v_1, \ldots, v_m)$. Ein innerer Knoten u der Tiefe i in T korrespondiert zu einer partiellen Belegung von $(v_1, \ldots, v_i)$ mit $(b_1, \ldots, b_i)$. Unser Ziel ist, die Knoten von T „bottom-up“ mit den Booleschen Wahrheitswerten 0 oder 1 zu markieren, wobei ein zur partiellen Belegung $(b_1, \ldots, b_i)$ korrespondierender Knoten u der Tiefe i genau dann mit 1 markiert werden soll, wenn die Bedingung

$$\mathcal{Q}_{i+1} a_{i+1} \in \{0,1\} \cdots \mathcal{Q}_m a_m \in \{0,1\} : F(b_1, \ldots, b_i, a_{i+1}, \ldots, a_m) \tag{26}$$

erfüllt ist. Zur Markierung eines zur vollständigen Belegung $b = (b_1, \ldots, b_m)$ korrespondierenden Blattes, brauchen wir lediglich die Formel F an b auszuwerten. Wenn wir induktiv annehmen, dass die Kinder u_0, u_1 eines inneren Knoten u bereits korrekt mit Wahrheitswerten w_0, w_1 markiert sind, dann führt folgende Regel zu einer korrekten Markierung von u:

- Wenn u mit Quantor $\exists$ markiert ist, dann markiere u mit Wahrheitswert $w_0 \vee w_1$.
- Wenn u mit Quantor $\forall$ markiert ist, dann markiere u mit Wahrheitswert $w_0 \wedge w_1$.

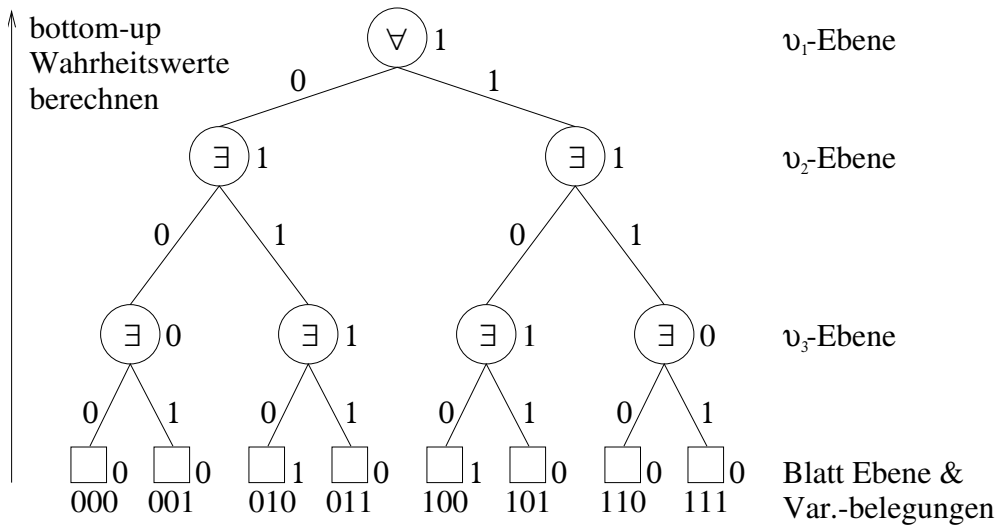


Abbildung 14: Der Entscheidungsbaum T_1 für die quantifizierte Boolesche Formeln qbf_1 aus Beispiel 17.2.

Auf diese Weise können wir, beginnend bei den Blättern, alle Knoten von T bottom-up mit Wahrheitswerten markieren bis wir schließlich bei der Wurzel anlangen. Aus der Invarianzbedingung (26) geht durch Inspektion des Spezialfalles $i = 0$ hervor, dass die Wurzel von T genau dann mit 1 markiert ist, wenn die gesamte quantifizierte Formel zu TQBF gehört. Wir akzeptieren also die Eingabe genau dann, wenn am Ende unserer Markierungsprozedur die Wurzel des Entscheidungsbaumes T mit Wahrheitswert 1 markiert ist.
Entscheidungsbaum T hat 2^m Blätter. Seine Größe ist i.A. nicht polynomiell in der Eingabelänge n beschränkt. Eine platzeffiziente EXHAUSTIVE SEARCH darf zu keinem Zeitpunkt den Baum T vollständig abspeichern. Um die eingangs beschriebene Markierungsprozedur platzeffizient durchzuführen, wird T in Präordnung durchlaufen, wobei wir zu jedem Zeitpunkt nur den Pfad von der Wurzel zum aktuellen Knoten (Ariadne-Faden) und die aktuelle partielle Belegung $(b_1, \ldots, b_i)$ kellerartig abspeichern. Zusätzlich speichern wir zu jedem Knoten auf dem Ariadne-Faden einen Record der folgenden Form[50]:

$\forall$	w_0	?	?
Quantor	Wahrheitswert linker Sohn	Wahrheitswert rechter Sohn	Wahrheitswert aktueller Knoten

Hieraus kann man ersehen, dass pro Knoten auf dem Ariadne Faden nur konstanter Platz erforderlich ist, und dass die Information in den Records ausreicht, um die oben beschriebene Markierungsprozedur am Laufen zu halten. Da in einem Blatt des Entscheidungsbaumes die Boolesche Formel F auf gegebenen Bits $b_1, \ldots, b_m$ auszuwerten ist, genügt Speicherplatz der

[50]Die hier gewählte Belegung des Record mit bekanntem Wahrheitswert des linken Sohns und unbekanntem Wahrheitswert des rechten entspricht der in Abbildung 16 gezeigten Momentaufnahme des Präordnungsdurchlaufes.

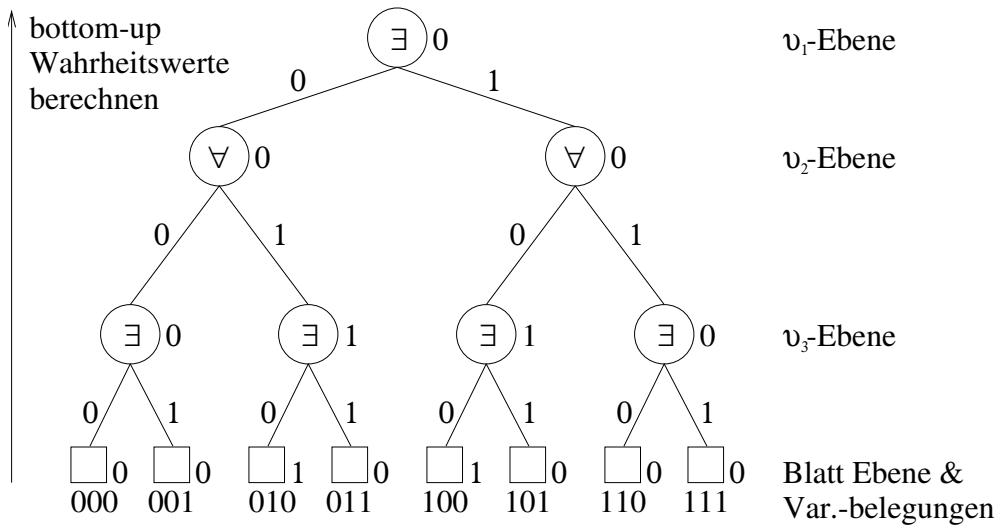


Abbildung 15: Der Entscheidungsbaum T_2 für die quantifizierte Boolesche Formel qbf_2 aus Beispiel 17.2.

Größenordnung $m + |F| = O(n)$, wobei $|F|$ die Kodierungslänge von F bezeichnet. Aus unserer Diskussion geht hervor, dass TQBF zur Klasse *PSpace* gehört. •

Satz 17.4 *TQBF ist PSpace-hart*

Beweis Der Beweis erfolgt über eine *generische* Karp-Reduktion, d.h., wir haben für eine *beliebige Sprache* $L \in PSpace$ zu zeigen, dass $L \leq_{pol}$ TQBF. Sei M eine DTM, die L erkennt und, für ein Polynom S, auf Eingaben w der Länge n nur $S(n)$ Zellen besucht. Dann hat die Rechnung von M auf w für eine geeignete Konstante c maximal $2^{cS(n)}$ Konfigurationen. Offensichtlich ist $w \in L$ äquivalent dazu, dass die Startkonfiguration $K_0(w)$ sich in $2^{cS(n)}$ Rechenschritten in die (oBdA eindeutige) akzeptierende Endkonfiguration K_+ transformieren lässt. Der Schlüssel zum Beweis ist daher die Aussage, dass sich die Relation

$$K \xrightarrow{s} K' :\Leftrightarrow \text{Konfiguration } K \text{ wird von } M \text{ in } 2^s \text{ Rechenschritten in } K' \text{ überführt}$$

durch eine effizient konstruierbare QBF F_s beschreiben lässt. Wir werden im folgenden eine Konfiguration K mit dem entsprechenden binären Codewort einer Länge $m = O(S(n))$ identifizieren. Belegungen einer Kollektion von m Booleschen Variablen, die korrekt gebildeten Codewörtern entsprechen, repräsentieren auf diese Weise Konfigurationen. Wie wir aus dem Beweis des Cook'schen Theorems wissen, existiert eine effizient konstruierbare Boolesche Formel F_0 (in $2m$ Booleschen Variablen) mit $F_0(K, K') = 1$ gdw K und K' Konfigurationen entsprechen (genauer: deren Codewörtern) und wenn $K \xrightarrow{0} K'$ (d.h. K' ist direkte Folgekonfiguration von K). Dies legt (ähnlich wie im Beweis des Theorems von Savich) die folgende Rekursion nahe:

$$F_s[K, K'] := (\exists K'') F_{s-1}[K, K''] \wedge F_{s-1}[K'', K'] \ . \tag{27}$$

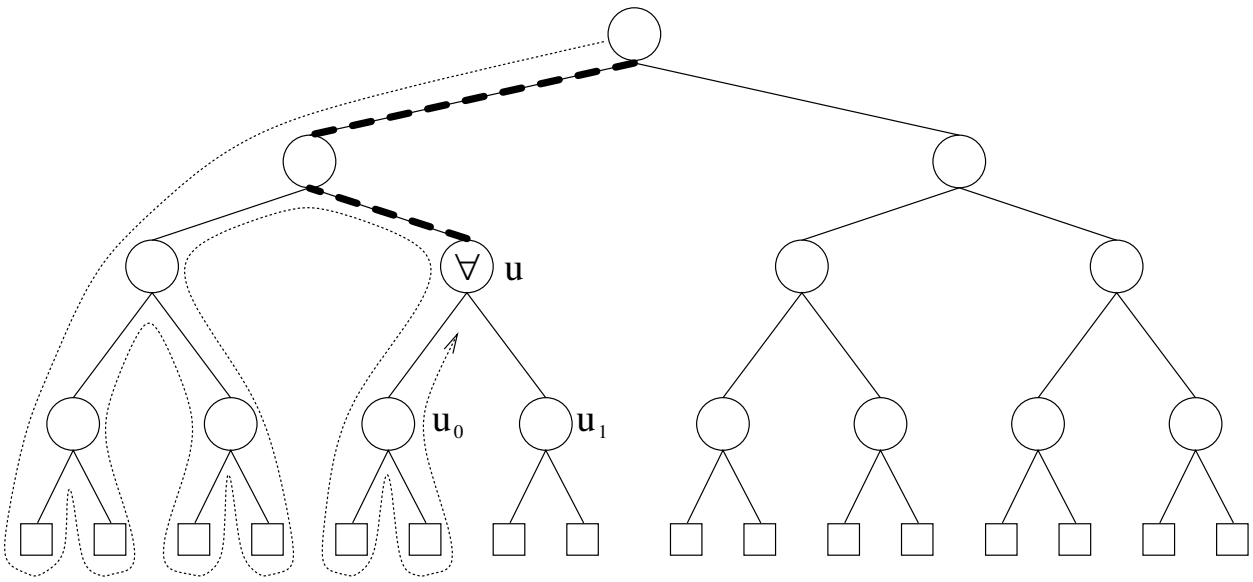


Abbildung 16: Eine Momentaufnahme des Präordnungs-Durchlaufs durch einen Entscheidungsbaum (u mit Söhnen u_0, u_1 der aktuell besuchte Knoten, Ariadne-Faden hervorgehoben).

Die schlechte Nachricht ist aber, dass die resultierende QBF für $K_0(x) \stackrel{cS(n)}{\longrightarrow} K_+(x)$ eine in n exponentielle Länge hat. Dies liegt daran, dass wir in (27) *zwei* rekursive Aufrufe vorfinden. Für $s = cS(n)$ entsteht somit ein vollständig binärer Rekursionsbaum der Tiefe $cS(n)$ mit $2^{cS(n)}$ Blättern.

Einschub Es ist nicht verwunderlich, dass der Plan noch nicht aufgeht. Bisher haben wir ausschließlich ∃-Quantoren verwendet. Wenn wir auf diese Weise eine Karp-Reduktion erhalten hätten, hätten wir *PSpace = NP* gezeigt (was eine Sensation wäre). Die noch ausstehende Kompression der Formellänge wird wohl Gebrauch von dem ∀-Quantor machen müssen.

Wir ändern jetzt die Rekursionsregel (27) unter Einsatz des ∀-Quantors so ab, dass nur noch *ein* rekursiver Aufruf erfolgt. Anstelle eine binären Rekursionsbaumes der Tiefe $cS(n)$ erhalten wir dann einen „Rekursionspfad“ der Tiefe $cS(n)$, der folgerichtig zu einer quantifizierten Booleschen Formel der Kodierungslänge $O(S(n))$ führen wird. In einem gewissen Sinn war bisher alles nur Vorgeplänkel und erst jetzt kommt die

Entscheidende Idee In die Rekursion (27) fügen wir zwei *neue* Variablenkollektionen X, Y ein. Wir werden mit *einem* rekursiven Aufruf testen, ob die Y-Variablenbelegung eine Konfiguration repräsentiert, die sich mit 2^{s-1} Rechenschritten aus der X-Variablenbelegung ergibt. Mit Hilfe eines ∀-Quantors werden wir erreichen, dass (X, Y) einmal in der Rolle von (K, K'') und ein weiteres Mal in der Rolle von (K'', K') auftreten. Dadurch ersetzt der *eine* rekursive Aufruf die *beiden* rekursiven Aufrufe in (27).

Die Gleichheit $X = V$ zweier Variablenkollektionen ist komponentenweise zu verstehen. Man beachte auch, dass Gleichheit zweier Boolescher Variablen oder Implikation zwischen Booleschen Variablen als Abkürzungen im Sinne von

$$\begin{aligned} a = b &\Leftrightarrow (a \wedge b) \vee (\bar{a} \wedge \bar{b}) \\ a \to b &\Leftrightarrow \bar{a} \vee b \end{aligned}$$

zu sehen sind. Nach diesen Vorbemerkungen geben wir nun die neue Rekursionsregel an, die (27) ersetzen soll:

$$\begin{aligned} &F_s[K, K'] := \\ &\quad (\exists K'')(\forall X)(\forall Y)\left(((X = K \wedge Y = K'') \vee (X = K'' \wedge Y = K')) \to F_{s-1}[X, Y]\right) \ . \end{aligned}$$

Nach etwas kontemplativer Versenkung sollte klar werden, dass die neue Rekursionsregel logisch äquivalent zu (27) ist. Wenn wir jetzt $F_{cS(n)}[K_0(w), K_+]$ gemäß der neuen Rekursion expandieren (und in die Pränexform bringen, so dass die Quantoren ganz am Anfang der Formel stehen), ergibt sich schließlich die angestrebte Karp-Reduktion von L nach TQBF. •

Wenn wir uns die Reduktionsabbildung im Beweis von Satz 17.4 nochmals aufmerksam ansehen, können wir die folgende Beobachtung machen:

Bemerkung 17.5 *$x \mapsto F_x$ produziert eine QBF, deren Quantorenkette von x nur indirekt über $|x|$ abhängt. Anders ausgedrückt: verschiedene Wörter x, x' derselben Länge n führen zur selben Quantorenkette in den QBFs F_x und $F_{x'}$.*

Sätze 17.3 und 17.4 liefern zusammen die

Folgerung 17.6 *TQBF ist PSpace-vollständig.*

17.2 Prädikatenlogische Charakterisierung von PSpace

Im folgenden Satz wird *PSpace* mit Hilfe alternierender Quantorenketten charakterisiert. Dabei ist $(\exists)_{pol}$ bzw. $(\forall)_{pol}$ die Abkürzung für einen Quantor der sich über poly(n) Bits erstreckt.

Satz 17.7 *Eine Sprache L gehört genau dann zu PSpace, wenn eine Sprache $L_0 \in P$ und ein Polynom p existieren, so dass*

$$x \in L \Leftrightarrow (\exists y_1)_{pol}(\forall y_2)_{pol}(\exists y_3)_{pol} \cdots (Q_{p(|x|)} y_{p(|x|)})_{pol} \langle y_1, \ldots, y_{p(|x|)}, x\rangle \in L_0 \ . \qquad (28)$$

Beweis Wir setzen zunächst $L \in$ *PSpace* voraus und weisen Bedingung (28) nach. Wegen $L \leq_{pol}$ TQBF gibt es für alle Wörter x eine (aus x polynomiell konstruierbare) Boolesche Formel F_x in $m = \text{poly}(|x|)$ Booleschen Variablen $v_1, \ldots, v_m$ und eine Quantorenkette $\mathcal{Q}_1, \ldots, \mathcal{Q}_m \in \{\exists, \forall\}$ mit der Eigenschaft

$$x \in L \Leftrightarrow (\mathcal{Q}_1 v_1) \ldots (\mathcal{Q}_m v_m) F_x(v_1, \ldots, v_m) \ . \qquad (29)$$

Wir erinnern daran, dass die Quantorenkette in (29) von x nur indirekt über $|x|$ abhängt. OBdA gelte $\mathcal{Q}_1 = \exists$. (Ansonsten könnten wir eine redundante Variable mit einem $\exists$-Quantor hinzufügen.) Wir können die Quantorenkette $\mathcal{Q}_1, \ldots, \mathcal{Q}_m$ in maximale Teilketten mit Quantoren nur eines Typs unterteilen. Die Anzahl der Teilketten sei mit $p(|x|)$ bezeichnet. Dann können wir (29) umschreiben wie folgt:

$$x \in L \Leftrightarrow (\exists y_1)_{pol}(\forall y_2)_{pol}(\exists y_3)_{pol} \cdots (Q_{q(|x|)} y_{q(|x|)})_{pol} F_x(y_1, \ldots, y_{p(|x|)}) \ . \tag{30}$$

Hierbei bezeichnet y_i die Teilkollektion der Booleschen Variablen aus $v_1, \ldots, v_m$, welche in der i-ten Teilkette von $(\mathcal{Q}_1 v_1) \ldots (\mathcal{Q}_m v_m)$ vorkommen. Mit

$$L_0 := \{\langle y_1, \ldots, y_{p(|x|)}, x\rangle : F_x(y_1, \ldots, y_{p(|x|)} = 1\} \in P$$

kann schließlich (30) in die Form (28) gebracht werden.
Nehmen wir umgekehrt an, dass L durch Bedingung (28) gegeben ist. Wir haben $L \in$ *PSpace* zu zeigen. Dieser Nachweis kann in völliger Analogie zum Nachweis von TQBF $\in$ *PSpace* (Beweis von Satz 17.3) geführt werden. Wir modellieren mit einem Entscheidungsbaum T die möglichen Belegungen der Variablenkollektionen $y_1, \ldots, y_{p(|x|)}$. Wir verwenden wieder eine platzeffiziente EXHAUSTIVE SEARCH durch T unter Einsatz der Ariadne-Faden Technik. Im Laufe der EXHAUSTIVE SEARCH kann eine Markierung der Knoten von T mit Wahrheitswerten vorgenommen werden, so dass x genau dann zu L gehört, wenn die Wurzel dabei mit Wahrheitswert 1 markiert wird. Die Ausfüllung der technischen Details überlassen wir der Leserin und dem Leser. •

17.3 Weitere PSpace-vollständige Probleme

In diesem Abschnitt nennen wir ohne Beweis ein paar weitere PSpace-vollständige Probleme, um einen Eindruck zu vermitteln, welcher Problemtypus in diese Kategorie gehört. Dabei setzen wir voraus, dass die Operationen der Konkatenation zweier formaler Sprachen (Operationszeichen „$\cdot$“) und der Kleene'sche Abschluss einer formalen Sprache (Operationszeichen „$*$“) aus der Vorlesung „Theoretische Informatik“ bekannt sind.

17.3.1 Probleme mit Automaten und Grammatiken

Definition 17.8 Reguläre Ausdrücke *über einem Alphabet* Σ *und die von ihnen induzierten formalen Sprachen sind induktiv definiert wie folgt:*

1. $\emptyset, \epsilon, \sigma$ *mit* $\sigma \in \Sigma$ *sind reguläre Ausdrücke. Die hiervon induzierten Sprachen sind (in dieser Reihenfolge) die leere Menge, die Menge* $\{\epsilon\}$ *und die Menge* $\{\sigma\}$.

2. *Wenn* α, β *reguläre Ausdrücke sind, dann sind auch* $(\alpha + \beta)$, $(\alpha \cdot \beta)$ *und* (α^*) *reguläre Ausdrücke. Wenn* $L(\alpha)$ *und* $L(\beta)$ *die von* α *und* β *induzierten Sprachen bezeichnen, dann sind die von* $(\alpha+\beta)$, $(\alpha\cdot\beta)$ *und* (α^*) *induzierten Sprachen (in dieser Reihenfolge)* $L(\alpha) \cup L(\beta)$, $L(\alpha) \cdot L(\beta)$ *und* $(L(\alpha))^*$.

Reguläre Ausdrücke werden zum Beispiel bei Editoren oder in der lexikalischen Analyse von Compilern angewendet. Das folgende Problem fragt danach, ob ein gegebener regulärer Ausdruck eine Sprache induziert, die zumindest ein Wort über dem Grundalphabet ausschließt.

Definition 17.9 *REGULAR EXPRESSION NON-UNIVERSALITY ist das folgende Problem: Zu gegebenem regulären Ausdruck α entscheide ob $L(\alpha) \neq \Sigma^*$.*

Satz 17.10 (Stockmeyer und Meyer, 1973) *Für jedes mindestens zweielementige Grundalphabet (also zum Beispiel für das binäre Alphabet $\{0,1\}$) gilt: REGULAR EXPRESSION NON-UNIVERSALITY ist PSPACE-vollständig.*

Der Beweis von Stockmeyer und Meyer erfolgt mit einer generischen Karp-Reduktion.

Ein LBA (ausgeschrieben: Linear Bounded Automaton) ist im wesentlichen eine $O(n)$-platzbeschränkte NTM.[51] Es ist bekannt, dass die Klasse der von LBAs akzeptierbaren Sprachen identisch ist mit der Klasse der sogenannten kontextsensitiven Sprachen (die aus der Vorlesung *Theoretische Informatik* bekannt sein könnten).

Definition 17.11 *LBA-ACCEPTANCE ist das Problem zu entscheiden, ob ein gegebener LBA M ein gegebenes Eingabewort x akzeptiert. Hierbei sind also sowohl (eine Kodierung von) M als auch x Bestandteil der Eingabe. LINEAR SPACE ACCEPTANCE ist das entsprechende Problem für $O(n)$-platzbeschränkte DTMs.*

Satz 17.12 (Karp, 1972) *LBA-ACCEPTANCE und LINEAR SPACE ACCEPTANCE sind beides PSpace-vollständige Probleme.*

Der Beweis von Karp, den wir als **Übung** empfehlen, erfolgt über eine (technisch einfache) generische Karp-Reduktion unter Verwendung eines „padding argument“. Wegen des engen Zusammenhangs zwischen LBAs und kontextsensitiven Grammatiken (auf deren formale Definition wir hier verzichten) ergibt sich leicht die **Übg.**

Folgerung 17.13 *Das Problem zu einer gegebenen kontextsensitiven Grammatik G und einem gegebenen Wort x zu entscheiden, ob x zu der von G induzierten kontextsensitiven Sprache gehört, ist PSpace-vollständig.*

17.3.2 Spielprobleme

Es gibt einen natürlichen Zusammenhang zwischen Zwei-Personen Spielen und alternierenden Quantorenketten. Nennen wir die Personen WEISS und SCHWARZ. Die Frage, ob WEISS (im Anzug) eine Gewinnstrategie über m Züge hat, liest sich in expandierter Form wie folgt:

($\exists$ Zug_1 für WEISS)	($\forall$ Zug_1 von SCHWARZ)
($\exists$ Zug_2 für WEISS)	($\forall$ Zug_2 von SCHWARZ)
...	...
($\exists$ Zug_m für WEISS)	Spielstellung ist gewonnen für WEISS

[51] Aus dem Kompressionssatz folgt, dass eine solche NTM oBdA genau n Zellen (also exakt die das Eingabewort enthaltenden Zellen) besucht.

Die Frage mit WEISS im Nachzug (also SCHWARZ im Anzug) lässt sich analog mit einer alternierenden Quantorenkette formulieren, die mit einem $\forall$-Quantor beginnt. Im Lichte von Satz 17.7 kann es daher nicht überraschen, dass sich über geeignet definierte Spiele PSpace-vollständige Probleme ergeben. Wir konkretisieren dies an folgenden Spielen:

QBF-Spiel: Ein „Spielbrett" des QBF-Spiels besteht aus einer Booleschen Formel $F = F(v_1, \ldots, v_m)$ in m Booleschen Variablen (wobei verschiedene Spielbretter verschiedene Größen m haben dürfen). Zwei Spieler sind abwechselnd am Zug. Spieler 1 (der „Verifizierer") hat das Ziel, F zu 1 auszuwerten, wohingegen Spieler 2 (der „Falsifizierer") die Auswertung 0 anstrebt. Spieler 1 belegt v_1 mit einem Bit seiner Wahl, Spieler 2 danach die Variable v_2, dann wieder Spieler 1 die Variable v_3, und so weiter. Es seien $a_1, \ldots, a_m$ die von den Spielern gewählten Bits. Spieler 1 gewinnt gdw $F(a_1, \ldots, a_m) = 1$.

GENERALIZED HEX: Sei $G = (V, E)$ ein Graph mit zwei ausgezeichneten Knoten $s, t \in V$. Das Spiel HEX[G] ist ein Zwei-Personen Spiel[52], bei welchem WEISS über weiße und SCHWARZ über schwarze Spielsteine verfügt. Beide Spieler ziehen abwechselnd mit WEISS im Anzug. Ein Zug besteht darin, einen eigenen Stein auf einen noch unbesetzten Knoten aus $V \setminus \{s, t\}$ zu setzen. Das Spiel ist vorüber, wenn alle Knoten aus $V \setminus \{s, t\}$ von (weißen oder schwarzen) Steinen besetzt sind. Die finale Spielstellung ist genau dann eine *Gewinnstellung* für WEISS, wenn die Knoten s, t zusammen mit den von weißen Steinen besetzten Knoten einen Verbindungspfad von s nach t in G enthalten. WEISS muss also mit der Strategie spielen, eine Verbindung von s nach t herzustellen; SCHWARZ ist bemüht, solche Verbindungen mit Hilfe von schwarzen Steinen zu vereiteln.

Offensichtlich ist die Frage nach einer Gewinnstrategie für Spieler 1 im QBF-Spiel gerade das Problem TQBF in einer (leicht zu durchschauenden) Verkleidung:

Bemerkung 17.14 *Zu entscheiden, ob Spieler 1 im QBF-Spiel eine Gewinnstrategie hat ist PSpace-vollständig.*

Das folgende Resultat geht in die gleiche Richtung, ist aber weniger offensichtlich:

Satz 17.15 (Even und Tarjan, 1976) *Die Frage, ob WEISS eine Gewinnstrategie im Spiel GENERALIZED HEX besitzt, ist PSpace-vollständig.*

Die Mitgliedschaft in *PSpace* sollte klar sein. Der Beweis der *PSpace*-Härte erfolgt durch Angabe einer Karp-Reduktion von TQBF auf GENERALIZED HEX.

In ähnlicher Weise kann man zeigen, dass (mehr oder weniger) natürliche Verallgemeinerungen[53] populärer Spiele (wie zum Beispiel SCHACH oder GO) ebenfalls *PSpace*-hart sind.

[52] Für spezielle Graphen in den USA ein handelsübliches Spiel

[53] Um sinnvoll über Komplexität reden zu können, benötigt man Spielbretter einer variablen Größe.

Weitere Übungsaufgabe Das japanische Brettspiel „Go-Moku“ wird von zwei Spielern auf einem (19×19)-Gitter gespielt. Die Spieler setzen abwechselnd ihre Steine. Derjenige, der zuerst fünf seiner Steine direkt aufeinanderfolgend in einer Zeile, Spalte oder Diagonale platziert, hat gewonnen. Betrachte dieses Spiel als auf ein $(n \times n)$-Gitter verallgemeinert. Es sei GM die Sprache aller Spielstellungen im verallgemeinerten Go-Moku-Spiel, für die Spieler 1 eine Gewinnstrategie besitzt. **Zeige**, dass GM zur Klasse *PSpace* gehört. **Übg.**

17.4 Offene Probleme

Über die (bisher unbewiesene) $P \neq NP$-Vermutung haben wir schon viel berichtet. Die Vermutung $P \neq PSpace$ ist vergleichsweise schwächer. Im Lichte der diversen prädikatenlogischen Charakterisierungen kann man diese offenen Fragen nun auch folgendermaßen ausdrücken:

P versus NP Sind prädikatenlogische Aussagen[54] die einen $(\exists)_{pol}$-Quantor verwenden dürfen, mächtiger als rein aussagenlogische (also quantorenfreie) Aussagen?

P versus PSpace Sind prädikatenlogische Aussagen, die eine alternierende Quantorenkette variabler Länge verwenden dürfen, mächtiger als rein aussagenlogische Aussagen?

Wir haben die Fragen bewusst informell gehalten, damit die Formulierungen griffig sind. Leider haben die prädikatenlogischen Formulierungen der dahinter liegenden berechnungstheoretischen Probleme bisher nicht wirklich zu einer Klärung der offenen Fragen beigetragen.

18 Die polynomielle Hierarchie

In diesem Kapitel beschäftigen wir uns mit einer von Meyer und Stockmeyer in den 1970ern eingeführte (vermutlich unendlichen) Hierarchie von Komplexitätsklassen zwischen P und *PSpace*. Abschnitt 18.1 führt in die Thematik ein. In Abschnitt 18.2 präsentieren wir die formale Definition und beweisen einige elementare Eigenschaften. Wir wir in Kapitel 19 noch sehen werden, lässt sich jede Stufe der Hierarchie in einer ähnlichen Weise prädikatenlogisch charakterisieren, wie wir das im Falle von *PSpace* schon kennengelernt haben.

18.1 Einleitende Betrachtungen

Wir gehen der Frage nach, ob die Klasse der Sprachen, welche (in einem zu präzisierenden Sinne) reduzierbar auf eine Sprache aus *NP* sind, mit *NP* zusammenfällt oder *NP* echt enthält. Im zweiten Fall würde die Reduktionstechnik gewissermaßen für eine „neue Qualität“ an Rechenkraft sorgen. Es wird sich zeigen, dass die Antwort auf diese Frage vom

[54] Gemeint ist stets pränexe Normalform mit Quantoren vom Typ $(\exists)_{pol}$ oder $(\forall)_{pol}$ und einer in Polynomialzeit entscheidbaren Aussage.

gewählten Reduktionsbegriff abhängt. Im Falle von Turing-Reduktionen entsteht eine Komplexitätsklasse die (voraussichtlich) NP echt enthält, wohingegen Karp-Reduktionen nicht aus NP hinausführen.

18.1.1 Polynomielle Reduktionen auf Sprachen aus NP

Falls $L \leq_{pol} L'$ und $L' \in NP$, dann folgt bekanntlich $L \in NP$. Wir erhalten nämlich eine polynomielle NTM M für L mit Hilfe der polynomiellen NTM M' für L' und der Karp-Reduktion $f : \Sigma^* \to \Sigma^*$ wie folgt:

1. Transformiere die Eingabe x zum Mitgliedschaftsproblem für die Sprache L in die korrespondierende Eingabe $x' = f(x)$ zum Mitgliedschaftsproblem für die Sprache L'.
2. Sezte M' auf x' an und akzeptiere genau dann, wenn M' akzeptiert.

Die Klasse der auf Sprachen aus NP Karp-reduzierbaren Sprachen führt also nicht aus NP heraus.

18.1.2 Turing-Reduktionen auf Sprachen aus NP

Bei Turing-Reduktionen ergibt sich ein völlig anderes Bild. Wir wollen dies durch ein paar Beobachtungen und Beispiele illustrieren.

Eine erste Beobachtung wäre, dass jede Sprache auf ihr Komplement Cook-reduzierbar ist:

Beobachtung 18.1 $\forall L \subseteq \Sigma^* : L \leq_T \bar{L}$.

Beweis Zu Eingabe x können wir das $\bar{L}$-Orakel befragen und seine Antwort umkehren. •

Aus Beobachtung 18.1 ergibt sich unmittelbar, dass jede Sprache aus $NP \cup$ co-NP auf das Erfüllbarkeitsproblem Cook-reduzierbar ist:

Beobachtung 18.2 $\forall L \in NP \cup co\text{-}NP : L \leq_T SAT$.

Beweis Für $L \in NP$ ergibt sich $L \leq_{pol}$ SAT und somit auch $L \leq_T$ SAT aus der NP-Vollständigkeit von SAT. Für $L \in$ co-NP folgt zunächst $\bar{L} \in NP$. Zusammen mit Beobachtung 18.1 erhalten wir die Reduktionskette $L \leq_T \bar{L} \leq_T$ SAT. Nun folgt $L \leq_T$ SAT aus der Transitivität von Turing-Reduktionen. •

Die Klasse der auf SAT Cook-reduzierbaren Sprachen führt also (unter der $NP \neq$ co-NP-Voraussetzung) aus der Klasse NP heraus. Eine DOTM mit „Rechenkraft“ P und ein Orakel der „Rechenkraft“ NP können demzufolge in Kooperation eine „Rechenkraft“ oberhalb von NP erlangen.

Ein Orakel für eine NP-vollständige Sprache nennen wir im folgenden einfach NP-Orakel. Wir wollen illustrieren, wie mächtig eine polynomiell zeitbeschränkte NOTM wird, wenn sie mit einem NP-Orakel ausgestattet ist. Dabei spielen die in den folgenden Beispielen beschriebenen Sprachen eine wichtige Rolle:

Beispiel 18.3 *MINIMUM EQUIVALENT CIRCUIT (kurz: MEC) sei die Sprache aller Paare (genauer: Kodierungen von Paaren) der Form (S, k), wobei gilt:*

- *S ist ein Boolescher Schaltkreis.*
- *k ist eine nicht-negative ganze Zahl.*
- *Es gibt einen Booleschen Schaltkreis S' mit höchstens k Bausteinen (aus der Basis AND, OR, NOT), der die selbe Boolesche Funktion wie S berechnet.*

Das zu MEC korrespondierende Minimierungsproblem gehört in den Bereich der Hardwareminimierung: suche die billigste Realisierung eines gegebenen Schaltkreises.

Beispiel 18.4 *UNSAT (ausgeschrieben: UNSATISFIABILITY) sei die Sprache der nicht erfüllbaren Booleschen CNF-Formeln C. Da UNSAT im Wesentlichen das Komplement von SAT ist, ist UNSAT co-NP-vollständig.*

Beispiel 18.5 *SAT* sei die Sprache aller Schaltkreise (genauer: Kodierungen von Schaltkreisen), die nicht die konstante Nullfunktion berechnen. Da SAT offensichtlich ein Teilproblem von SAT* ist und andererseits SAT* zu NP gehört, ist SAT* NP-vollständig.*

Wir werden im Folgenden nachweisen, dass MEC ein sehr hartes Problem ist: es ist co-NP-hart unter Karp-Reduktionen und NP-hart unter Turing-Reduktionen. Vermutlich gehört es nicht zu $NP \cup$ co-NP, aber diese Frage ist offen. Unter der $NP \neq$ co-NP-Voraussetzung kann man zumindest ausschließen, dass MEC zu NP gehört. Der Härte von MEC zum Trotz werden wir eine polynomiell zeitbeschränkte NOTM angeben, die mit Hilfe eines NP-Orakels als Akzeptor von MEC auftritt. Die kombinierte Rechenkraft zweier Agenten (NOTM und Orakel) mit „Rechenkraft“ NP reicht also aus, um die „harte Nuss“ namens MEC zu knacken.

Wir starten unsere Analyse von MEC mit der folgenden

Beobachtung 18.6 *$UNSAT \leq_{pol} MEC$.*

Beweis Wir verwenden im Prinzip die Technik der Spezialisierung. Sei C eine Boolesche CNF-Formel, f_C die von C berechnete Boolesche Funktion und S_C der zweistufige Schaltkreis, der C auf die offensichtliche Weise berechnet (Berechnung der Klauseln mit OR-Gattern auf der ersten Stufe und Berechnung der Konjunktion aller Klauseln mit einem AND-Gatter auf der zweiten Stufe). OBdA enthalte C mindestens eine nicht-tautologische Klausel.[55] Dann gilt erst recht $f_C \not\equiv 1$ und die folgenden Äquivalenzen sind offensichtlich:

$$\begin{aligned} C \in \text{UNSAT} &\Leftrightarrow f_C \equiv 0 \\ &\Leftrightarrow f_C \text{ ist eine konstante Funktion} \\ &\Leftrightarrow f_C \text{ ist von einem Schaltkreis mit 0 Bausteinen berechenbar} \\ &\Leftrightarrow (S_C, 0) \in \text{MEC} \end{aligned}$$

[55] Eine Klausel heiße *tautologisch*, wenn sie nicht falsifiziert werden kann (wie zum Beispiel die leere Klausel oder die Klausel $v_1 \vee \bar{v}_1$).

Da $(S_C, 0)$ leicht aus C berechenbar ist, erhalten wir eine Karp-Reduktion von UNSAT auf MEC. •

Folgerung 18.7 *1. SAT $\leq_T$ MEC.*

2. MEC ist co-NP-hart unter polynomiellen oder Turing-Reduktionen.

3. MEC ist NP-hart unter Turing-Reduktionen.

4. MEC $\notin$ NP unter der NP $\neq$ co-NP-Voraussetzung.

Beweis

1. Aus UNSAT $\leq_{pol}$ MEC (gemäß Beobachtung 18.6) folgt UNSAT $\leq_T$ MEC. Wegen SAT $\leq_T$ UNSAT (gemäß Beobachtung 18.1) ergibt sich dann SAT $\leq_T$ MEC aus der Transitivität der Turing-Reduktionen.

2. Die co-*NP*-Härte von UNSAT vererbt sich wegen UNSAT $\leq_{pol}$ MEC auf MEC.

3. Die *NP*-Härte von SAT (unter Karp-Reduktionen) impliziert die *NP*-Härte von SAT unter Turing-Reduktionen. Diese vererbt sich wegen SAT $\leq_T$ MEC auf MEC.

4. Würde MEC als ein (unter Karp-Reduktionen) co-*NP*-hartes Problem zu *NP* gehören, dann würde jede Sprache $L \in$ co-*NP* wegen $L \leq_{pol}$ MEC und MEC $\in$ *NP* ebenfalls zu *NP* gehören. (Vgl. Abschnitt 18.1.1.) Dies hätte co-*NP* $\subseteq$ *NP* und daher auch co-*NP* = *NP* zur Folge. Im Umkehrschluss ergibt sich MEC $\notin$ *NP* aus der *NP* $\neq$ co-*NP*-Voraussetzung.

•

Wie angekündigt hat sich ergeben, dass MEC ein sehr hartes Problem ist. Wir komplettieren das Bild jetzt durch folgende

Beobachtung 18.8 *Es gibt eine polynomiell zeitbeschränkte NOTM, die mit Hilfe eines SAT*-Orakels als Akzeptor von MEC auftritt.*

Beweis Die NOTM gehe auf einer Eingabe (S, k) vor wie folgt:

1. Rate einen Schaltkreis S' mit maximal k Bausteinen.

2. Seien f und f' die von S und S' berechneten Booleschen Funktionen. Synthetisiere auf die offensichtliche Weise die Schaltkreise S und S' zu einem neuen Schaltkreis $S^{\oplus}$, welcher die Funktion
$$f \oplus f' := (f \wedge \bar{f}') \vee (\bar{f} \wedge f')$$
berechnet.

3. Befrage das SAT*-Orakel nach der Erfüllbarkeit von $S^{\oplus}$.

4. Akzeptiere genau dann, wenn $S^{\oplus}$ nicht erfüllbar ist.

Diese Vorgehensweise ist wegen

$$\begin{aligned} f \equiv f' &\Leftrightarrow f \oplus f' \equiv 0 \\ &\Leftrightarrow S^{\oplus} \text{ ist nicht erfüllbar} \end{aligned}$$

korrekt. Offensichtlich existiert genau dann, eine akzeptierende Rechnung zur Eingabe (S, k), wenn (S, k) zur Sprache MEC gehört. Die Laufzeit der skizzierten NOTM ist sicherlich polynomiell beschränkt. •

18.2 Definition und elementare Eigenschaften der Hierarchie

Definition 18.9 *1. Sei $L \subseteq \Sigma^*$ eine Sprache.*

(a) $P[L]$ bezeichne die Klasse aller Sprachen, die von einer polynomiellen DOTM mit Orakel L akzeptiert werden können.

(b) $NP[L]$ bezeichne die Klasse aller Sprachen, die von einer polynomiell zeitbeschränkten NOTM mit Orakel L akzeptiert werden können.

2. Sei $\mathcal{C}$ eine Klasse von Sprachen über Alphabet Σ.

$$\begin{aligned} P[\mathcal{C}] &:= \bigcup_{L \in \mathcal{C}} P[L] \\ NP[\mathcal{C}] &:= \bigcup_{L \in \mathcal{C}} NP[L] \end{aligned}$$

(Notationen $P^L, NP^L, P^{\mathcal{C}}, NP^{\mathcal{C}}$ anstelle von $P[L], NP[L], P[\mathcal{C}], NP[\mathcal{C}]$ sind ebenso gebräuchlich.)

Zum Aufwärmen machen wir ein paar einfache Beobachtungen:

Lemma 18.10 *1. $\mathcal{C} \subseteq P[\mathcal{C}] \subseteq NP[\mathcal{C}]$.*

2. $P[P] = P$ und $NP[P] = NP$.

3. Falls L_0 $\mathcal{C}$-vollständig (unter Karp-Reduktionen) ist, dann gilt $P[L_0] = P[\mathcal{C}]$ und $NP[L_0] = NP[\mathcal{C}]$.

Beweis

1. $P[\mathcal{C}] \subseteq NP[\mathcal{C}]$ ist klar, da eine DOTM als eine spezielle NOTM angesehen werden kann. Um die Inklusion $\mathcal{C} \subseteq P[\mathcal{C}]$ nachzuweisen, genügt es, $L \in P[L]$ für eine beliebige Sprache $L \subseteq \Sigma^*$ zu zeigen. Dies ist aber klar, da eine mit einem L-Orakel ausgestattete DOTM die Mitgliedschaft von x in L durch Befragung des L-Orakels nach x unmittelbar testen kann.

2. Eine polynomielle DTM (oder gar NTM) kann die Arbeit eines P-Orakels auch gleich selber machen.

3. Wir beschränken uns auf die Aussage $P[L_0] = P[\mathcal{C}]$. Der Nachweis der Aussage $NP[L_0] = NP[\mathcal{C}]$ erfolgt analog.
   $P[L_0] \subseteq P[\mathcal{C}]$ ist die triviale Beweisrichtung. Zum Nachweis von $P[\mathcal{C}] \subseteq P[L_0]$ haben wir für eine beliebige Sprache $L \in \mathcal{C}$ die Inklusion $P[L] \subseteq P[L_0]$ zu zeigen. Hierzu nutzen wir die $\mathcal{C}$-Vollständigkeit von L_0 aus. Sei f die in Polynomialzeit berechenbare Abbildung bei der Karp-Reduktion von L nach L_0. Die an ein L-Orakel gerichtete Nachfrage nach einem Wort $x \in \Sigma^*$ kann dann durch eine an ein L_0-Orakel gerichtete Nachfrage nach dem Wort $f(x)$ ersetzt werden.

•

Aussage 1 von Lemma 18.10 besagt gewissermaßen, dass eine Klasse $\mathcal{C}$ durch Anwendung des „P-Operators“ zu einer Oberklasse $P[\mathcal{C}]$ „aufgeblasen“ wird. Die Anwendung des mächtigeren „NP-Operators“ bläst $\mathcal{C}$ zu einer (evtl.) noch größeren Klasse auf. In diesem Lichte bedeutet Aussage 2, dass der P-Operator auf P stationär ist und der NP-Operator P zu NP aufbläst. Aussage 3 bedeutet, dass es keinen Unterschied macht, ob wir einen der Operatoren auf $\mathcal{C}$ oder eine $\mathcal{C}$-vollständige Sprache anwenden. Die folgende Definition der polynomiellen Hierarchie, welche sich zwischen P bzw. NP und $PSpace$ aufspannt, beruht auf iterativer Anwendung des P- und des NP-Operators.

Definition 18.11 *Die Sprachklassen Σ_k und Δ_k sind für $k = 0$ bzw. $k \geq 1$ induktiv definiert wie folgt:*

$$\begin{aligned} \Sigma_0 &:= \Delta_0 := P \\ \Sigma_k &:= NP[\Sigma_{k-1}] \\ \Delta_k &:= P[\Sigma_{k-1}] \end{aligned}$$

Die Sprachlassen Π_k ergeben sich als Komplementärklassen zu den Sprachklassen Σ_k:

$$\Pi_k := co\text{-}\Sigma_k \ .$$

Die Klassen $\Sigma_k, \Delta_k, \Pi_k$ bilden (zusammen mit den zwischen ihnen geltenden Inklusionsbeziehungen) die sogenannte polynomielle Hierarchie.

Die untersten drei Stufen dieser Hierarchie ergeben sich aus dieser Definition wie folgt:

Stufe 0 $\Sigma_0 = \Delta_0 = P$ und $\Pi_0 = \text{co-}P = P$.

Stufe 1 $\Sigma_1 = NP[P] = NP$, $\Delta_1 = P[P] = P$ und $\Pi_1 = \text{co-}NP$.

Stufe 2 $\Sigma_2 = NP[NP]$, $\Delta_2 = P[NP]$ und $\Pi_2 = \text{co-}NP[NP]$.

Beobachtung 18.8 kann jetzt etwas gelehrsamer formuliert werden:

Lemma 18.12 *MEC* $\in \Sigma_2$.

Beweis Da SAT* *NP*-vollständig ist, gilt $\Sigma_2 = NP[NP] = NP[\text{SAT}^*]$. Beobachtung 18.8 ist äquivalent zu MEC $\in NP[\text{SAT}^*]$. •

Die folgenden Lemmas formulieren elementare Eigenschaften der polynomiellem Hierarchie und sind darüberhinaus eine geeignete Aufwärmübung, die die Vertrautheit mit Definition 18.11 erhöhen sollte. Wir geben bei jedem Beweis nur die entscheidende Idee an.

Das folgende Lemma macht deutlich, dass die polynomielle Hierarchie sich nicht verändert, wenn wir den *P*- oder *NP*-Operator auf Π_{k-1} anstelle von Σ_{k-1} anwenden.

Lemma 18.13 $\Sigma_k = NP[\Pi_{k-1}]$ *und* $\Delta_k = P[\Pi_{k-1}]$ *für alle* $k \geq 1$.

Beweis Nutze aus, dass ein L-Orakel gleichwertig zu einem $\bar{L}$-Orakel ist: statt nach $w \in L$ zu fragen, frage nach $w \in \bar{L}$ und negiere die Antwort. •

Das nächste Lemma besagt, dass die Klassen Δ_k abgeschlossen unter Komplementbildung sind.

Lemma 18.14 $\Delta_k = \text{co-}\Delta_k$ *für alle* $k \geq 0$.

Beweis Nutze aus, dass eine L akzeptierende DOTM durch Vertauschen von akzeptierenden und nicht-akzeptierenden Zuständen in eine $\bar{L}$ akzeptierende DOTM transformiert wird. •

Das nun folgende Lemma deckt die Inklusionsbeziehungen auf:

Lemma 18.15 $\Delta_k \subseteq \Sigma_k \cap \Pi_k$ *und* $\Sigma_k \cup \Pi_k \subseteq \Delta_{k+1}$ *für alle* $k \geq 0$.

Beweis Der Beweis erfolgt induktiv. Für $k = 0$ sind die Aussagen gültig. Für $k \geq 1$ ergibt sich zunächst

$$\Delta_k = P[\Sigma_{k-1}] \subseteq NP[\Sigma_{k-1}] = \Sigma_k \ .$$

Dies impliziert

$$\Delta_k = \text{co-}\Delta_k \subseteq \text{co-}\Sigma_k = \Pi_k$$

und daher auch $\Delta_k \subseteq \Sigma_k \cap \Pi_k$. Der Rest ergibt sich mit Hilfe der Lemmas 18.10 und 18.13:

$$\Sigma_k \subseteq P[\Sigma_k] = \Delta_{k+1} \text{ und } \Pi_k \subseteq P[\Pi_k] = \Delta_{k+1} \ .$$

•

Die Inklusionsbeziehungen in Lemma 18.15 sind in Abbildung 17 visualisiert. Es ergibt sich ausgehend von P und NP (auf den unteren Stufen) eine (voraussichtlich) unendliche Hierarchie von immer mächtigeren Klassen. Wir erhalten eine gemeinsame Oberklasse PH durch die Definition

$$PH := \bigcup_{k \geq 0} \Sigma_k \ . \tag{31}$$

Wir werden später zeigen, dass (wie in Abbildung 17 bereits zu sehen) PH in PSpace enthalten ist.

Offene Probleme Es ist nicht wirklich geklärt, ob PH eine unendliche Hierarchie ist oder (das andere Extrem) vielleicht zu NP oder gar P kollabiert. Wir werden später sehen, dass $P = NP$ die scheinbar stärkere Aussage $P = PH$ impliziert. Auf ähnliche Weise würde $NP =$ co-NP, also Abgeschlossenheit von NP unter Komplementbildung, $NP = PH$ implizieren. In Verbindung mit der $P \neq NP$-Vermutung bzw. der $NP \neq$ co-NP-Vermutung, gibt es den naheliegenden Verdacht, dass PH nicht auf eine feste Stufe kollabiert, sondern dass jede neue Stufe der Hierarchie eine echt erweiterte Sprachklasse repräsentiert. In diesem Fall ergäbe sich eine unendliche Hierarchie. Eine vertiefte Debatte dieser Fragen werden wir führen können, wenn wir in Kapitel 19 den Satz von Celia Wrathall kennengelernt haben, der die polynomielle Hierarchie zur prädikatenlogischen Beschreibungskomplexität in Beziehung setzt.

19 Der Satz von Celia Wrathall

Im Jahre 1977 fand Celia Wrathall eine prädikatenlogische Charaktersisierung der Klassen Σ_k und Π_k, welche wir auf der k-ten Stufe der polynomiellen Hierarchie vorfinden. Wir erinnern uns, dass Σ_k aus Σ_{k-1} durch Anwendung des NP-Operators hervorgeht: $\Sigma_k = NP[\Sigma_{k-1}]$. Durch anschließenden Übergang zur Komplementärklasse erhalten wir $\Pi_k =$ co-$NP[\Sigma_{k-1}]$. Anschaulich können wir die Hintereinanderausführung des NP-Operators und des Übergangs zur Komplementärklasse als co-NP-Operator auffassen. Die Operatoren NP und co-NP sind berechnungstheoretischer Natur: die aktuelle Komplexitätsklasse wird erweitert durch Hinzufügen von „Rechenpower“. Celia Wrathall setzt implizit[56] an die Stelle der Operatoren NP und co-NP Operatoren prädikatenlogischer Natur. Wir notieren diese Operatoren im Folgenden durch $(\exists)_{pol}$ und $(\forall)_{pol}$. Es wird sich zeigen, dass die iterative Anwendung der neuen Operatoren die selbe Hierarchie aufbaut wie die iterative Anwendung der ursprünglichen Operatoren. Der Aufbau der polynomiellen Hierarchie mit den Operatoren $(\exists)_{pol}$ uns $(\forall)_{pol}$ führt zu einer prädikatenlogischen Charakterisierung von Σ_k und Π_k vermöge alternierender Quantorenketten der Länge k. Der Satz von Celia Wrathall zeigt gewissermaßen wie sich die Berechnungskomplexität einer Sprache der polynomiellen Hierarchie in prädikatenlogische „Beschreibungskomplexität“ übersetzt (und umgekehrt).

[56] Wir weichen bei der Darstellung des Satzes von Celia Wrathall und seines Beweises von der Originalarbeit ab.

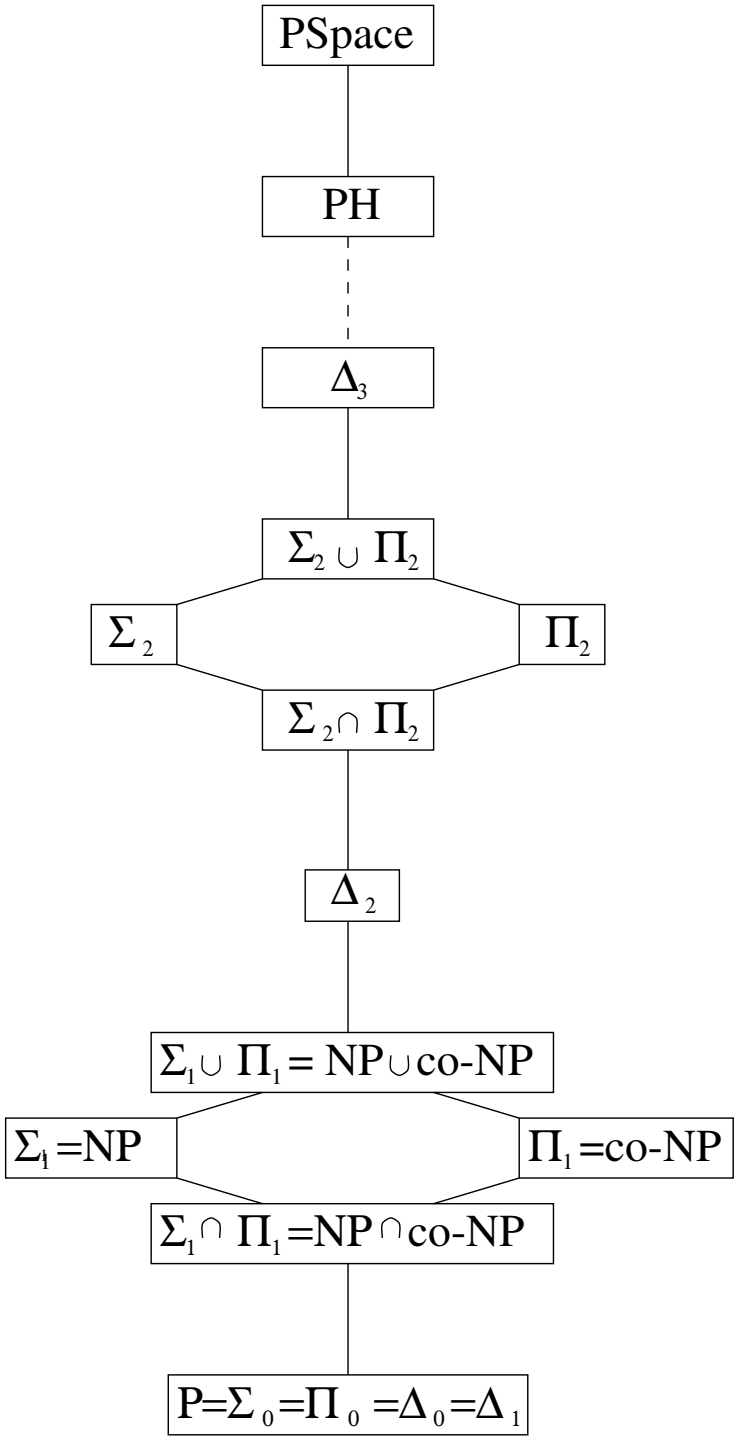


Abbildung 17: Die polynomielle Hierarchie von Meyer und Stockmeyer.

In Abschnitt 19.1 erfolgt eine prädikatenlogische Definition der polynomiellen Hierarchie. Abschnitt 19.2 ist dem Satz von Wrathall gewidmet, welcher besagt, dass die prädikatenlogische Definition zur selben Hierarchie führt wie unsere ursprüngliche Definition aus Abschnitt 18.2. Einige Anwendungen dieses Satzes sind in Abschnitt 19.3 zusammengestellt.

19.1 Die prädikatenlogischen Operatoren

Wir erinnern an die Notation $(\exists y)_{pol}$ bzw. $(\forall y)_{pol}$ aus Abschnitt 17.2 zur Quantifizierung über alle Wörter y einer Länge $p(|x|)$ über einem festen Alphabet, wobei p ein geeignet gewähltes Polynom ist und der Bezugsstring x stets aus dem Kontext hervorgeht.

Definition 19.1 *Sei $\mathcal{C}$ eine Klasse von Sprachen (über einem festen Alphabet).*

1. *$(\exists)_{pol}[\mathcal{C}]$ sei die Klasse aller Sprachen L, die sich für ein $L_0 \in \mathcal{C}$ in der Form*

$$L = \{x : (\exists y)_{pol}\langle y, x\rangle \in L_0\} \tag{32}$$

   schreiben lassen.

2. *$(\forall)_{pol}[\mathcal{C}]$ sei die Klasse aller Sprachen L, die sich für ein $L_0 \in \mathcal{C}$ in der Form*

$$L = \{x : (\forall y)_{pol}\langle y, x\rangle \in L_0\} \tag{33}$$

   schreiben lassen.

Da unsere Schreibkonventionen zwar bequem aber zugegebenermaßen auch etwas schlampig sind, geben wir hier (zum letzten Mal) zusätzlich die ausführliche Schreibweise an (um Missverständnissen vorzubeugen). Gleichung (32) bedeutet: Es existiert ein Polynom p, so dass die Sprache L genau diejenigen Wörter x enthält, zu denen ein Wort y der Länge höchstens $p(|x|)$ existiert, so dass der Kodierungsstring für das Paar (y, x) in L_0 zu liegen kommt. Gleichung (33) bedeutet: Es existiert ein Polynom p, so dass die Sprache L genau diejenigen Wörter x enthält, welche die Eigenschaft haben, dass für alle Wörter y der Länge höchstens $p(|x|)$ der Kodierunsstring für das das Paar (y, x) in L_0 zu liegen kommt. (Uff, es lebe die Schlampigkeit!)

Bei festem k werden wir ab jetzt $\langle y_1, \langle y_2 \ldots \langle y_{k-1}, y_k\rangle \ldots\rangle\rangle$ mit $\langle y_1, y_2, \ldots, y_k\rangle$ identifizieren. Das hat den Vorteil, dass k-fache Anwendung von Definition 19.1 einfach zu Quantorenketten der Länge k und einer Bedingung der Form $\langle y_1, \ldots, y_k, x\rangle \in L_0$ führt.

Für einen Quantor $Q \in \{\exists, \forall\}$ bezeichne seine Negation $\bar{Q}$ den jeweils anderen „dualen“ Quantor:

$$\bar{Q} := \begin{cases} \forall, & \text{falls } Q = \exists, \\ \exists, & \text{falls } Q = \forall. \end{cases} \tag{34}$$

Wir werden im Folgenden häufig mit alternierenden Quantorenketten arbeiten. Um diese bequem zu notieren, definieren wir

$$Q_k := \begin{cases} \exists, & \text{falls } k \text{ ungerade ist}, \\ \forall, & \text{falls } k \text{ gerade ist}. \end{cases} \tag{35}$$

Trivialerweise gilt dann

$$\bar{Q}_k = \begin{cases} \forall, & \text{falls } k \text{ ungerade ist,} \\ \exists, & \text{falls } k \text{ gerade ist.} \end{cases} \tag{36}$$

Eine alternierende Quantorenkette der Länge k hat dann die Form

$$(\exists)_{pol}(\forall)_{pol}(\exists)_{pol}\cdots(Q_k)_{pol} \ ,$$

wenn sie mit $(\exists)_{pol}$ beginnt. Beginnt sie mit $(\forall)_{pol}$ hat sie die Form

$$(\forall)_{pol}(\exists)_{pol}(\forall)_{pol}\cdots(\bar{Q}_k)_{pol} \ .$$

Die analogen Schreibweisen benutzen wir auch für $(\exists y)_{pol}$ und $(\forall y)_{pol}$.
Es folgen ein paar Aufwärmübungen zu den neuen Definitionen und Schreibweisen.

Lemma 19.2 *Beim Übergang zur Komplementärklasse gelten die folgenden Regeln:*

$$co\text{-}(\exists)_{pol}(\forall)_{pol}(\exists)_{pol}\cdots(Q_k)_{pol}[\mathcal{C}] = (\forall)_{pol}(\exists)_{pol}(\forall)_{pol}\cdots(\bar{Q}_k)_{pol}[co\text{-}\mathcal{C}] \ . \tag{37}$$

$$co\text{-}(\forall)_{pol}(\exists)_{pol}(\forall)_{pol}\cdots(\bar{Q}_k)_{pol}[\mathcal{C}] = (\exists)_{pol}(\forall)_{pol}(\exists)_{pol}\cdots(Q_k)_{pol}[co\text{-}\mathcal{C}] \ . \tag{38}$$

Beweis Wir beschränken uns auf den Nachweis von (37). Der Nachweis von (38) lässt sich völlig analog führen.
k-fache Anwendung von Definition 19.1 führt zu folgender Einsicht. Zu einer Sprache $L \in (\exists)_{pol}(\forall)_{pol}(\exists)_{pol}\cdots(Q_k)_{pol}[\mathcal{C}]$ gibt es eine Sprache $L_0 \in \mathcal{C}$, so dass L gegeben ist durch

$$L = \{x : (\exists y_1)_{pol}(\forall y_2)_{pol}(\exists y_3)_{pol}\cdots(Q_k y_k)_{pol}\langle y_1, \ldots, y_k, x\rangle \in L_0\} \ . \tag{39}$$

Die Komplementärklasse co-$\mathcal{C}$ enthält anstelle von L die Sprache $\bar{L}$, deren Definitionsbedingung sich durch logische Negation der Definitionsbedingung von L ergibt:

$$\bar{L} = \{x : (\forall y_1)_{pol}(\exists y_2)_{pol}(\forall y_3)_{pol}\cdots(\bar{Q} y_k)_{pol}\langle y_1, \ldots, y_k, x\rangle \in \bar{L}_0 \ . \tag{40}$$

Hierbei haben wir die de Morgan'schen Regeln angewendet, welche dazu führen, dass jeder Quantor durch sein duales Gegenstück ersetzt und die atomare Bedingung $\langle y_1, \ldots, y_k, x\rangle \in L_0$ am Ende der Quantorenkette zu $\langle y_1, \ldots, y_k, x\rangle \in \bar{L}_0$ logisch negiert wird. Durch k-fache Anwendung von Definition 19.1 erkennen wir jetzt, dass die Bedingung (40) gerade Sprachen aus $(\forall)_{pol}(\exists)_{pol}(\forall)_{pol}\cdots(\bar{Q}_k)_{pol}[\text{co-}\mathcal{C}]$ charakterisiert. •

Wegen co-$P = P$ ergibt sich unmittelbar die

Folgerung 19.3

$$co\text{-}(\exists)_{pol}(\forall)_{pol}(\exists)_{pol}\cdots(Q_k)_{pol}[P] = (\forall)_{pol}(\exists)_{pol}(\forall)_{pol}\cdots(\bar{Q}_k)_{pol}[P] \ .$$

$$co\text{-}(\forall)_{pol}(\exists)_{pol}(\forall)_{pol}\cdots(\bar{Q}_k)_{pol}[P] = (\exists)_{pol}(\forall)_{pol}(\exists)_{pol}\cdots(Q_k)_{pol}[P] \ .$$

Ähnlich leicht erhalten wir die

Folgerung 19.4 *$NP = (\exists)_{pol}[P]$ und co-$NP = (\forall)_{pol}[P]$.*

Beweis Die Aussage $NP = (\exists)_{pol}[P]$ entpuppt sich mit etwas Nachdenken als unsere gute alte Charakterisierung von NP mit Hilfe von polynomiell verifizierbaren Relationen (Stichwort: Rate- und Verifikationsprinzip). Die zweite Aussage ergibt sich dann mit Hilfe von Folgerung 19.3: co-NP = co-$(\exists)_{pol}[P] = (\forall)_{pol}[P]$. •

Wir stellen uns als nächstes die Frage, ob die Operatoren $(\exists)_{pol}$ und $(\forall)_{pol}$ zu einer (evtl. unechten) Erweiterung einer Sprachklasse führen. Vorsicht ist insofern geboten, als man künstliche Sprachklassen $\mathcal{C}$ **angeben kann**, für welche die Inklusionsbeziehung $\mathcal{C} \subseteq (\exists)_{pol}[\mathcal{C}]$ bzw. $\mathcal{C} \subseteq (\forall)_{pol}[\mathcal{C}]$ *nicht* gilt. Andererseits beweisen wir folgendes **Übg.**

Lemma 19.5 *$\mathcal{C}$ besitze die die folgende Abschluss-Eigenschaft. Falls $L \in \mathcal{C}$, dann gehöre auch die Sprache*

$$L_\epsilon := \{\langle \epsilon, x\rangle : x \in L\}$$

zu $\mathcal{C}$. (Hierbei bezeichne ϵ wie üblich das leere Wort.) Unter dieser Voraussetzung gilt $\mathcal{C} \subseteq (\exists)_{pol}[\mathcal{C}]$ und $\mathcal{C} \subseteq (\forall)_{pol}[\mathcal{C}]$.

Beweis Sei $L \in \mathcal{C}$ und somit auch $L_\epsilon \in \mathcal{C}$. Um $\mathcal{C} \subseteq (\exists)_{pol}[\mathcal{C}]$ einzusehen, genügt es $L \in (\exists)_{pol}[\mathcal{C}]$ einzusehen. Dies ist aber klar, da wir L in folgender Form schreiben können

$$L = \{x : (\exists y)_{pol}\langle y, x\rangle \in L_\epsilon\} \ .$$

Das geeignete Polynom, das sich hinter dem Index „pol“ von $(\exists y)_{pol}$ versteckt, kann in diesem Fall als das Nullpolynom $p \equiv 0$ gewählt werden. In der Tat existiert ja für die Wörter x von L (und nur für diese) ein Wort y einer Länge 0 (nämlich das leere Wort $y = \epsilon$) mit $\langle y, x\rangle \in L_\epsilon$. Der Nachweis von $\mathcal{C} \subseteq (\forall)_{pol}[\mathcal{C}]$ geschieht völlig analog (oder durch Dualisierung). •

Es ist **leicht einzusehen**, dass alle Sprachklassen der polynomiellen Hierarchie (wie überhaupt alle „vernünftig definierten“ Komplexitätsklassen) die in Lemma 19.5 geforderte Abschluss-Eigenschaft besitzen. Für Σ_k und Π_k zeigen wir aber darüberhinaus das **Übg.**

Lemma 19.6 *Für alle $k \geq 1$ gilt $\Sigma_k = (\exists)_{pol}[\Sigma_k]$ und $\Pi_k = (\forall)_{pol}[\Pi_k]$.*

Beweis Wir zeigen zunächst $\Sigma_k = (\exists)_{pol}[\Sigma_k]$. $\Sigma_k \subseteq (\exists)_{pol}[\Sigma_k]$ ergibt sich aus Lemma 19.5. Zum Nachweis von $(\exists)_{pol}[\Sigma_k] \subseteq \Sigma_k$ sei $L \in (\exists)_{pol}[\Sigma_k]$ für ein $L_0 \in \Sigma_k$ gegeben durch

$$L = \{x : (\exists y)_{pol}\langle y, x\rangle \in L_0\} \ .$$

Sei M_0 ein Akzeptor von $L_0 \in \Sigma_k$. Wegen $k \geq 1$ ist M_0 eine polynomiell zeitbeschränkte NOTM mit einem Orakel für eine Sprache $L_0' \in \Sigma_{k-1}$. (Dies schließt in einer etwas verklausulierten Form auch den Fall $k = 1$ ein.) Folgende NOTM M, versehen mit einem Orakel für $L_0' \in \Sigma_{k-1}$, ist ein polynomiell zeitbeschränkter Akzeptor von L:

1. Rate ein Wort y der Länge pol($|x|$).

2. Benutze die NOTM M_0 mit dem L_0'-Orakel, um $\langle y, x\rangle \in L_0$ zu verifizieren (falls möglich).

Man überlegt sich leicht, dass M auf einer Eingabe x genau dann eine akzeptierende Rechnung besitzt, wenn $x \in L$.
Mit unserem Hintergrundwissen ergibt sich nun $\Pi_k = (\forall)_{pol}[\Pi_k]$ leicht durch Dualisierung:

$$\Pi_k = \text{co-}\Sigma_k = \text{co-}(\exists)_{pol}[\Sigma_k] = (\forall)_{pol}[\text{co-}\Sigma_k] = (\forall)_{pol}[\Pi_k] \ .$$

•

Im Beweis von $\Sigma_k = (\exists)_{pol}[\Sigma_k]$ haben wir (salopp gesprochen) ausgenutzt, dass eine NOTM ihren Nichtdeterminismus dazu verwenden kann, den $(\exists)_{pol}$-Operator (durch Raten eines geeigneten Wortes y) überflüssig zu machen.

Aus beweistechnischen Gründen definieren wir jetzt die polynomielle Hierarchie ein zweites Mal (wie sich später herausstellen wird), wobei diesmal die neuen prädikatenlogischen Operatoren zum Einsatz kommen:

Definition 19.7 *Die Sprachklassen Σ_k' und Π_k' sind für $k = 0$ bzw. $k \geq 1$ induktiv definiert wie folgt:*

$$\begin{aligned} \Sigma_0' &:= \Pi_0' := P \ . \\ \Sigma_k' &:= (\exists)_{pol}[\Pi_{k-1}'] \ . \\ \Pi_k' &:= (\forall)_{pol}[\Sigma_{k-1}'] \ . \end{aligned}$$

Der Satz von Celia Wrathall besagt, dass $\Sigma_k' = \Sigma_k$ und $\Pi_k' = \Pi_k$, d.h., die „neue“ Hierachie ist die „alte“ Hierarchie in verkleideter Form. Der Beweis dieser Aussage ist nichttrivial und wird in Abschnitt 19.2 geführt.

Iterierte Anwendung von Definition 19.7 liefert sofort eine Charakterierung der Klassen Σ_k' und Π_k' vermöge alternierender Quantorenketten:

Lemma 19.8 *Für alle $k \geq 1$ gilt:*

$$\begin{aligned} L \in \Sigma_k' &\Leftrightarrow \exists L_0 \in P : L = \{x : (\exists y_1)_{pol}(\forall y_2)_{pol}(\exists y_3)_{pol} \cdots (Q_k y_k)_{pol}\langle y_1, \ldots, y_k, x\rangle \in L_0\} \ . \\ L \in \Pi_k' &\Leftrightarrow \exists L_0 \in P : L = \{x : (\forall y_1)_{pol}(\exists y_2)_{pol}(\forall y_3)_{pol} \cdots (\bar{Q} y_k)_{pol}\langle y_1, \ldots, y_k, x\rangle \in L_0\} \ . \end{aligned}$$

Aus der Charakterisierung der Klassen Σ_k', Π_k' mit alternierenden Quantorenketten lassen sich unmittelbar eine paar Schlussfolgerungen ziehen.

Folgerung 19.9

1. *Für alle $k \geq 0$ gilt $\Pi_k' = co\text{-}\Sigma_k'$.*
2. *Für alle $k \geq 1$ gilt $\Sigma_k' = (\exists)_{pol}[\Sigma_k']$.*
3. *Für alle $k \geq 1$ gilt $\Pi_k' = (\forall)_{pol}[\Pi_k']$.*
4. *Für alle $k \geq 0$ gilt $\Sigma_k' \subseteq \Sigma_{k+1}' \cap \Pi_{k+1}'$ und $\Pi_k' \subseteq \Pi_{k+1}' \cap \Sigma_{k+1}'$.*

Beweis Wir geben jeweils nur die entscheidende Idee an. (Die technischen Details sind leicht auszuarbeiten.)

1. Kombiniere Lemma 19.8 mit Lemma 19.2.
2. Die Grundidee ist, dass $(\exists)_{pol}(\exists)_{pol}$ gleichwertig zu $(\exists)_{pol}$ ist, da sich zwei aufeinanderfolgende Quantoren des selben Typs miteinander verschmelzen lassen.
3. Das geschilderte Verschmelzungsargument gilt natürlich auch für $(\forall)_{pol}$-Quantoren. Alternativ kann $\Pi'_k = (\forall)_{pol}[\Pi'_k]$ auch leicht durch Dualisierung gezeigt werden, wobei dann die bereits bewiesenen Aussagen $\Pi'_k = \text{co-}\Sigma'_k$ und $\Sigma'_k = (\exists)_{pol}[\Sigma'_k]$ benutzt werden.
4. Die „Beschreibungskraft" einer Quantorenkette kann nicht abnehmen, wenn wir sie um einen neuen Quantor verlängern.

•

Die Ausarbeitung der technischen Details zu dem „Verschmelzungsargument" empfehlen wir als **Übung**. Man kann sich dabei an der technischen Beweisführung des folgenden Lemmas orientieren. **Übg.**

Lemma 19.10 *Für jedes $k \geq 0$ sind die Klassen Σ'_k und Π'_k abgeschlossen unter Vereinigung und Durchschnitt von Sprachen.*

Beweis Für $k = 0$ ist die Aussage offensichtlich. Sei $k \geq 1$. Seien $L', L'' \in \Sigma'_k$. Gemäß Lemma 19.8 existieren dann Sprachen $L'_0, L''_0 \in P$ und Polynome $p'_1, \ldots, p'_k, p''_1, \ldots, p''_k$, so dass

$$\begin{aligned} L' &= \{x : (\exists y'_1)_{p'_1}(\forall y'_2)_{p'_2} \ldots (Q_k y'_k)_{p'_k} \langle y'_1, \ldots, y'_k, x\rangle \in L'_0\} \ , \\ L'' &= \{x : (\exists y''_1)_{p''_1}(\forall y''_2)_{p''_2} \ldots (Q_k y''_k)_{p''_k} \langle y''_1, \ldots, y''_k, x\rangle \in L''_0\} \ . \end{aligned}$$

Hieraus ergibt sich

$$\begin{aligned} L' \cup L'' &= \{x : (\exists y_1)_{pol}(\forall y_2)_{pol}(\exists y_3)_{pol} \cdots (Q_k y_k)_{pol} \text{Bedingung (A)}\} \\ L' \cap L'' &= \{x : (\exists y_1)_{pol}(\forall y_2)_{pol}(\exists y_3)_{pol} \cdots (Q_k y_k)_{pol} \text{Bedingung (B)}\} \ , \end{aligned}$$

wobei die Bedingungen (A) und (B) spezifiziert werden wie folgt:

- Für $i = 1, \ldots, k$ ist y_i ein Codewort für ein Wortpaar, sagen wir $y_i = \langle y'_i, y''_i\rangle$, und es gilt $|y'_i| \leq p'_i(|x|)$, $|y''_i| \leq p''_i(|x|)$. Weiterhin gilt

  $$\langle y'_1, \ldots, y'_k, x\rangle \in L'_0 \vee \langle y''_1, \ldots, y''_k, x\rangle \in L''_0$$

  im Falle der Bedingung (A) und

  $$\langle y'_1, \ldots, y'_k, x\rangle \in L'_0 \wedge \langle y''_1, \ldots, y''_k, x\rangle \in L''_0$$

  im Falle der Bedingung (B).

Man beachte, dass die Bedingungen (A) und (B) in Polynomialzeit getestet werden können.[57]
Eine Inspektion der obigen Definitionsbedingungen für $L' \cup L''$ bzw. $L' \cap L''$ verbunden mit einer erneuten Anwendung von Lemma 19.8 liefert $L' \cup L'', L' \cap L'' \in \Sigma'_k$.
Die entsprechenden Abschlusseigenschaften von Π'_k **ergeben sich leicht** durch Dualisierung. **Übg.**

•

19.2 Der Satz von Celia Wrathall und sein Beweis

Im vorangegangenen Abschnitt haben wir genug Vorarbeit geleistet, um jetzt dem Hauptresultat zu Leibe zu rücken:

Satz 19.11 (Wrathall, 1977) *Für alle $k \geq 0$ gilt $\Sigma'_k = \Sigma_k$ und $\Pi'_k = \Pi_k$.*

Beweis Für $k = 0$ ist die Behauptung richtig, da auf Stufe 0 alle beteiligten Sprachklassen mit P zusammenfallen. Wir können daher für ein beliebiges aber festes $k \geq 1$ induktiv $\Sigma'_{k-1} = \Sigma_{k-1}$ und $\Pi'_{k-1} = \Pi_{k-1}$ voraussetzen. Aus $\Sigma'_k = \Sigma_k$ ergibt sich $\Pi'_k = \Pi_k$ durch Dualisierung:

$$\Pi'_k = \text{co-co-}\Pi'_k = \text{co-}\Sigma'_k = \text{co-}\Sigma_k = \Pi_k \ .$$

Es genügt also der induktive Nachweis von $\Sigma'_k = \Sigma_k$.
Die Inklusion $\Sigma'_k \subseteq \Sigma_k$ ergibt sich wie folgt:

$$\Sigma'_k = (\exists)_{pol}[\Pi'_{k-1}] = (\exists)_{pol}[\Pi_{k-1}] \subseteq (\exists)_{pol}[\Sigma_k] = \Sigma_k \ .$$

Hierbei haben wir nacheinander die Definition von Σ'_k, die Induktionsvoraussetzung, die Inklusion $\Pi_{k-1} \subseteq \Sigma_k$ (die nach Anwendung des $(\exists)_{pol}$-Operators gültig bleibt) und schließlich Lemma 19.6 angewendet.
Zum Nachweis der Inklusion $\Sigma_k \subseteq \Sigma'_k$ werden wir härter arbeiten müssen. Sei $L \in \Sigma_k = NP[\Sigma_{k-1}]$. Sei weiter M die polynomiell zeitbeschränkte NOTM, die in Kooperation mit einem Orakel für $L' \in \Sigma_{k-1}$ als Akzeptor von L auftritt. M darf als nichtdeterministische Maschine „raten“ und als Orakel-Maschine zusätzlich pol($|x|$)-oft ihr L'-Orakel befragen. Zum Nachweis von $L \in \Sigma'_k = (\exists)_{pol}[\Pi'_{k-1}]$ müssen wir die Fähigkeiten von M irgendwie mit Hilfe des $(\exists)_{pol}$-Operators nachbilden. Die Korrespondenz von „Raten“ und $(\exists)_{pol}$-Quantifizierung schreckt uns nicht wirklich, da wir diese vom Rate- und Verifikationsprinzip her gewohnt sind. Bleibt aber das Problem, auch die Kommunikation zwischen M und dem L'-Orakel mit Hilfe des $(\exists)_{pol}$-Quantors nachzubilden. Wir verwenden dabei die folgende

Zentrale Beweisstrategie Antizipiere die Ratebits von M *und* die gesamte Kommunikation zwischen M und dem L'-Orakel durch einen einzigen (raffiniert entworfenen) String polynomiell beschränkter Länge und wende die $(\exists)_{pol}$-Quantifizierung auf diesen String an.

[57] Beim effizienten Testen von $|y'_i| \leq p'_i(|x|)$ bzw. $|y''_i| \leq p''_i(|x|)$ nutzen wir die Platzkonstruierbarkeit der betreffenden Polynome aus.

Es folgt die technische Umsetzung. M stoppe nach höchstens $T = \text{pol}(|x|)$ Rechenschritten und habe oBdA pro Schritt nur zwei Handlungsalternativen. Die Rechnung von M hängt von folgenden Faktoren ab:

- den nichtdeterministischen Entscheidungen repräsentiert durch einen Ratestring $y \in \{0,1\}^T$,
- den JA/NEIN-Antworten des L'-Orakels.

Für $r, s \leq T$ seien u, v Strings von der Form

$$u = \langle u_1, \ldots, u_r \rangle \text{ und } v = \langle v_1, \ldots, v_s \rangle \ ,$$

wobei $|u_i|, |v_j| \leq T$. Der String $\langle y, u, v, x \rangle$ heiße *konsistentes Rechenprotokoll*, falls folgendes gilt:

- Wenn M auf Eingabe x gemäß Ratestring y rechnet und wenn alle Orakelanfragen nach einem Wort aus $U := \{u_1, \ldots, u_r\}$ mit JA und alle Orakelanfragen nach einem Wort aus $V := \{v_1, \ldots, v_s\}$ mit NEIN beantwortet werden, dann stellt M nur Anfragen nach Wörtern aus $U \cup V$ und akzeptiert nach höchstens T Schritten.

Beachte, dass wir in dieser Definition *nicht* verlangen, dass die Antworten korrekt sind. Stattdessen sind die (evtl. falschen) Antworten durch die Strings u, v vorgegeben.[58] Die Sprache

$$L_0 := \{\langle y, u, v, x \rangle : \langle y, u, v, x \rangle \text{ ist ein konsistentes Rechenprotokoll}\}$$

gehört daher zu P.[59] Wir können nämlich auf Eingaben der Form $\langle y, u, v, x \rangle$ eine effiziente Simulation von M auf x starten[60], welche die konsistenten Rechenprotokolle (und nur diese) akzeptiert. Wir betrachten zwei Fälle:

Fall 1 M (mit Ratestring y) stelle während der ersten T simulierten Schritte nur Orakelanfragen aus $U \cup V$.
Dann können wir ohne Probleme T Schritte von M simulieren, da die Antwort auf Orakelanfragen effizient aus dem String u bzw. v abgelesen werden kann. Wenn M nach spätestens T Schritten (akzeptierend oder verwerfend) stoppt, dann stoppe die Simulation mit dem selben Ergebnis. Wenn M nach T Schritten nicht stoppt[61], dann stoppe die Simulation nach T simulierten Schritten verwerfend.[62]

[58] Dies ist eine subtile Angelegenheit: unsere Analyse der NOTM M beschäftigt sich mit dem Verhalten von M, für den Fall, dass wir sie vom Orakel abkoppeln und ihr auf andere Weise Antworten verabreichen. In mathematischen Beweisen ist alles erlaubt! Warten wir mal ab, wozu solche dubiosen Manöver gut sind.

[59] Dazu diente das Manöver! Der Nichtdeterminismus wird mit Hilfe des Strings y eliminiert; das Orakel wird mit Hilfe der Strings u, v überflüssig gemacht; bleibt eine deterministische polynomielle Rechnung.

[60] Eingaben, die schon syntaktisch von der Form $\langle y, u, v, x \rangle$ abweichen, können sofort verworfen werden.

[61] was man wegen der M zugespielten falschen Antworten nicht ausschließen kann

[62] An dieser Stelle verwenden wir die Zeitkonstruierbarkeit von $T = \text{pol}(|x|)$.

Fall 2 M (mit Ratestring y) stelle während der ersten T simulierten Schritte eine Orakelanfrage außerhalb $U \cup V$.
Bei der ersten Anfrage dieser Art stoppe die Simulation verwerfend.

Aus der Definition der konsistenten Rechenprotokolle ergibt sich direkt, dass unsere Simulation diese und nur diese akzeptiert. Somit gilt $L_0 \in P$.
Wir haben mit dem Nachweis von $L_0 \in P$ bereits einen Teilsieg errungen. Auf Dauer können wir uns aber nicht davor drücken, die Korrektheit von Antworten auf Orakelanfragen zu kontrollieren. Dieser Kontrolle dienen die folgenden beiden Sprachen:

$$\begin{aligned} L_1 &:= \{\langle y,u,v,x\rangle : u_1,\ldots,u_r \in L'\} \ . \\ L_2 &:= \{\langle y,u,v,x\rangle : v_1,\ldots,v_s \in \bar{L}'\} \ . \end{aligned}$$

Wegen $L' \in \Sigma_{k-1}$ und $\bar{L}' \in \text{co-}\Sigma_{k-1} = \Pi_{k-1}$ ist unter Verwendung der Induktionsvoraussetzung **leicht zu sehen**, dass folgende Aussagen gelten: **Übg.**

$$\begin{aligned} L_1 \in \Sigma_{k-1} &= \Sigma'_{k-1} \subseteq \Sigma'_k \ . \\ L_2 \in \Pi_{k-1} &= \Pi'_{k-1} \subseteq \Sigma'_k \ . \end{aligned}$$

Da $L_0 \in P \subseteq \Sigma'_k$ und Σ'_k abgeschlossen unter Durchschnittsbildung ist, gilt dann auch $L_0 \cap L_1 \cap L_2 \in \Sigma'_k$. Weiterhin sind L_0, L_1, L_2 gerade so entworfen, dass

$$x \in L \Leftrightarrow (\exists w)_{pol} w = \langle y,u,v\rangle \wedge \langle y,u,v,x\rangle \in L_0 \cap L_1 \cap L_2 \ .$$

Hieraus ergibt sich leicht $L \in (\exists)_{pol}[\Sigma'_k]$ und wegen $(\exists)_{pol}[\Sigma'_k] = \Sigma'_k$ (gemäß Folgerung 19.9) folgt schließlich $L \in \Sigma'_k$. Da L eine beliebige Sprache aus Σ_k ist, ergibt sich die angestrebte Inklusion $\Sigma_k \subseteq \Sigma'_k$. •

In Verbindung mit Lemma 19.8 erhalten wir unmittelbar die

Folgerung 19.12 *Für alle $k \geq 1$ gilt:*

$$\begin{aligned} L \in \Sigma_k &\Leftrightarrow \exists L_0 \in P : L = \{x : (\exists y_1)_{pol}(\forall y_2)_{pol}(\exists y_3)_{pol} \cdots (Q_k y_k)_{pol} \langle y_1,\ldots,y_k,x\rangle \in L_0\} \\ L \in \Pi_k &\Leftrightarrow \exists L_0 \in P : L = \{x : (\forall y_1)_{pol}(\exists y_2)_{pol}(\forall y_3)_{pol} \cdots (\bar{Q} y_k)_{pol} \langle y_1,\ldots,y_k,x\rangle \in L_0\} \end{aligned}$$

Die „Rechenkraft“ einer zu Σ_k korrespondierenden polynomiell zeitbeschränkten NOTM (mit einem Orakel aus Σ_{k-1}) entspricht also exakt der „Beschreibungskraft“ einer alternierenden mit $(\exists)_{pol}$ beginnenden Quantorenkette der Länge k, gefolgt von einer in Polynomialzeit überprüfbaren Aussage. Die analoge Entsprechung finden wir für die Klasse Π_k und alternierende mit $(\forall)_{pol}$ beginnende Quantorenketten der Länge k. Es hat sich somit eine erstaunliche Querbeziehung zwischen Berechnungskomplexität und prädikatenlogischer Beschreibungskomplexität ergeben.

19.3 Anwendungen des Satzes von Celia Wrathall

Für die harte Arbeit, die zum Beweis des Satzes von Celia Wrathall nötig war, werden wir in diesem Abschnitt entlohnt. Einige elegante Anwendungen dieses Satzes werden uns nun zufallen wie reife Früchte. Eine erste Anwendung besteht darin, die Einordnung einer Sprache in die polynomielle Hierarchie nicht durch „Programmierung“ einer geeigneten OTM vorzunehmen, sondern durch Angabe einer geeigneten „Definition“ der Sprache. Die Länge der alternierenden Quantorenkette in der Definitionsbedingung wird uns dann verraten, zu welcher Stufe der polynomiellen Hierchie die Sprache gehört. Eine zweite Anwendung ist die Analyse, unter welchen Bedingungen die polynomielle Hierachie auf eine endliche Stufe kollabiert. Eine dritte Anwendung besteht in der Angabe von Σ_k- oder Π_k-vollständigen Problemen und im Nachweis der Vollständigkeit.

19.3.1 Anwendung 1: Deskriptiver Nachweis der Mitgliedschaft in PH

Ein „algorithmischer“ Nachweis von $L \in PH$ besteht in der Angabe einer passenden OTM. Ein „deskriptiver“ Nachweis von $L \in PH$ macht sich Folgerung 19.12 zunutze und beschreibt L mit einer passenden prädikatenlogischen Formel. Wir illustrieren dies an einem

Beispiel 19.13 *Wir betrachten erneut die Sprache MEC (Minimum Equivalent Circuit). Wir haben früher bereits einen algorithmischen Nachweis von $MEC \in \Sigma_2$ geführt, und zwar durch Angabe einer polynomiellen NOTM, die mit Hilfe eines SAT*-Orakels als Akzeptor von MEC auftrat. Es folgt nun ein deskriptiver Nachweis für die selbe Aussage:*

$$MEC = \{\langle S, k\rangle : (\exists S')_{pol}(\forall a)_{pol}\langle S', a, S, k\rangle \in MEC_0\} \ ,$$

wobei MEC_0 die Sprache aller Strings der Form $\langle S', a, S, k\rangle$ sei, so dass zusätzlich die folgenden Bedingungen gelten:

1. *S repräsentiert einen Booleschen Schaltkreis. Die Anzahl der Eingangsknoten in S sei mit n bezeichnet.*
2. *k repräsentiert eine nicht-negative ganze Zahl.*
3. *S' repräsentiert einen Booleschen Schaltkreis mit n Eingangsknoten und höchstens k Bausteinen.*
4. *$a \in \{0,1\}^n$ und die Schaltkreise S und S' berechnen auf Eingabe a das gleiche Ausgabebit.*

Offensichtlich gilt $MEC_0 \in P$. Aus Folgerung 19.12 ergibt sich unmittelbar $MEC \in \Sigma_2$.

19.3.2 Anwendung 2: Hinreichende Bedingungen für einen Kollaps von PH

Es wird vermutet, dass die polynomielle Hierarchie aus unendlich vielen voneinander verschiedenen Stufen besteht. Der folgende Satz formuliert eine hinreichende Bedingung für die Negation dieser Vermutung.

Satz 19.14 *Falls $\Sigma_k = \Pi_k$ für ein $k \geq 1$, dann kollabiert PH auf den k-ten Level, d.h., dann gilt*

$$PH = \bigcup_{i=0}^{\infty} \Sigma_i = \bigcup_{i=0}^{k} \Sigma_i = \Sigma_k \ .$$

Beweis Wegen des Satzes von Wrathall gilt $\Sigma_{k+1} = (\exists)_{pol}\Pi_k$. Aus $\Sigma_k = \Pi_k$ folgt dann (unter Verwendung von Lemma 19.6):

$$\Sigma_{k+1} = (\exists)_{pol}\Pi_k = (\exists)_{pol}\Sigma_k = \Sigma_k \ .$$

Iteration dieses Argumentes (genauer: vollständige Induktion) liefert $\Sigma_k = \Sigma_{k+j} = \Pi_{k+j}$ für alle $j \geq 0$. Es folgt $PH = \cup_{i=0}^{k}\Sigma_k$ und wegen $\Sigma_0 \subseteq \Sigma_1 \subseteq \Sigma_2 \subseteq \cdots$ natürlich auch $PH = \Sigma_k$. •

Aus Satz 19.14 ergibt sich ein kleiner Beitrag zu der $P \neq NP$-Debatte:

Folgerung 19.15 *$P \neq NP \Leftrightarrow P \neq PH$.*

Beweis Wegen $P \subseteq NP \subseteq PH$ folgt aus $P \neq NP$ natürlich $P \subset NP \subseteq PH$ und somit auch $P \neq PH$. Zum Nachweis von $P \neq PH \implies P \neq NP$ genügt es freilich $P = NP \implies P = PH$ zu zeigen. Wie früher bereits nachgewiesen, impliziert $P = NP$ die Aussage $NP =$ co-NP. Wegen $NP = \Sigma_1$ und co-$NP = \Pi_1$ ergibt sich mit Satz 19.14 ein Kollaps von PH auf den ersten Level, d.h., $PH = NP$. Zusammen mit der Voraussetzung $P = NP$ ergibt sich nun auch $PH = P$. •

Folgerung 19.15 bedeutet, dass wir die berühmte $P \neq NP$-Vermutung im Prinzip durch den Nachweis von $P \neq PH$ beweisen könnten. Da PH eine viel mächtigere Komplexitätsklasse ist als NP, da sie ja NP bereits auf Stufe 1 zur Teilklasse hat, könnte die Separation von P und PH (durch Nachweis der Existenz einer Sprache in $PH \setminus P$) evtl. leichter zu erreichen sein als die Separation von NP und P.

19.3.3 Anwendung 3: Vollständige Probleme in PH

Das NP-vollständige Problem SAT lässt sich mit einer $(\exists)_{pol}$-quantifizierten Booleschen Formel (in konjunktiver Normalform) beschreiben, da wir nach der Existenz einer erfüllenden Belegung zu einer gegebenen CNF-Formel fragen. Um zu Σ_k- oder Π_k-vollständigen Problemen zu gelangen ist es naheliegend mit quantifizierten Booleschen Formeln zu operieren, wobei die Quantifizierung über eine alternierende Quantorenkette der Länge k erfolgt. Um solche Formeln bequem zu notieren, teilen wir die Booleschen Variablen in k Typen ein:

Definition 19.16 *Sei F eine Boolesche Formel und m die Anzahl der in F verwendeten Booleschen Variablen. Eine doppelt indizierte Boolesche Variable der Form v_{ij} heiße* Variable vom Typ *i. Wir sagen F ist eine Boolesche Formel* mit k Variablentypen*, wenn $m_1, \ldots, m_k$ existieren, so dass $m = m_1 + \cdots + m_k$ und F verwendet genau m_i Variablen vom Typ i. Dies seien oBdA die Variablen $\bar{v}_i = (v_{i1}, \ldots, v_{im_i})$ für $i = 1, \ldots, k$. Wir schreiben dann auch $F(\bar{v}_1, \ldots, \bar{v}_k)$ statt F, um die Abhängigkeit von den Variablen hervorzuheben.*

Die folgenden Sprachen sind eine naheliegende Verallgemeinerung der Sprache SAT, wobei an die Stelle eines einzelnen $(\exists)_{pol}$-Quantors eine alternierende Quantorenkette tritt und wir nicht mehr darauf bestehen, dass die Formel in konjunktiver Normalform (also eine CNF-Formel) ist.

Definition 19.17 *Die Sprache $\mathcal{B}_k$ enthalte alle (Kodierungen von) Booleschen Formeln mit k Variablentypen, welche die Bedingung*

$$\exists \bar{a}_1 \in \{0,1\}^{m_1} \forall \bar{a}_2 \in \{0,1\}^{m_2} \exists \bar{a}_3 \in \{0,1\}^{m_3} \ldots Q_k \bar{a}_k \in \{0,1\}^{m_k} : F(\bar{a}_1, \ldots, \bar{a}_k) = 1 \quad (41)$$

erfüllen.

Lemma 19.18 *Für alle $k \geq 1$: $\mathcal{B}_k \in PSpace$.*

Beweis $\mathcal{B}_k$ ist ein Teilproblem von TQBF und, wie wir wissen, TQBF $\in PSpace$. •

Eine zu $\mathcal{B}_k$ duale Klasse ergibt sich wie folgt:

Definition 19.19 *Die Sprache $\tilde{\mathcal{B}}_k$ enthalte alle (Kodierungen von) Booleschen Formeln mit k Variablentypen, welche die Bedingung*

$$\forall \bar{a}_1 \in \{0,1\}^{m_1} \exists \bar{a}_2 \in \{0,1\}^{m_2} \forall \bar{a}_3 \in \{0,1\}^{m_3} \ldots \bar{Q}_k \bar{a}_k \in \{0,1\}^{m_k} : F(\bar{a}_1, \ldots, \bar{a}_k) = 1 \quad (42)$$

erfüllen.

Der folgende Satz ist das Pendant zum Cook'schen Theorem:

Satz 19.20 *Für alle $k \geq 1$ gilt: $\mathcal{B}_k$ ist Σ_k-vollständig und $\tilde{\mathcal{B}}_k$ ist Π_k-vollständig.*

Beweis Der Beweis ähnelt dem Beweis des Cook'schen Theorems. Wir skizzieren die wesentlichen Ideen und weisen von Zeit zu Zeit auf die Analogie zum Beweis des klassischen Cook'schen Theorems hin. Genauere technische Details lassen sich leicht einarbeiten.
Wir diskutieren zunächst den Fall einer ungeraden Anzahl k von Variablentypen. In diesem Fall gilt $Q_k = \exists$, d.h., die alternierende Quantorenkette in (41) endet mit einem $\exists$-Quantor. Sei $L \in \Sigma_k$. Gemäß Folgerung 19.12 gilt (unter Beachtung von k ungerade):

$$L = \{x : (\exists y_1)_{pol} (\forall y_2)_{pol} \cdots (\exists y_k)_{pol} \langle y_1, \ldots, y_k, x \rangle \in L_0\}$$

für eine geeignete Sprache $L_0 \in P$. Wir können oBdA annehmen, dass $y_1, \ldots, y_k$ Wörter über dem Alphabet $\{0,1\}$ sind. Sei M_0 die polynomielle DTM, welche als Akzeptor von L_0 auftritt. Wie im Beweis des Cook'schen Theorems lässt sich die polynomielle deterministische Rechnung von M_0 auf $\langle y_1, \ldots, y_k, x \rangle$ mit Hilfe einer Booleschen CNF-Formel F beschreiben. F verwendet Boolesche Variable zur Kodierung der Strings $y_1, \ldots, y_k$ und weitere Hilfvariablen. Für $i = 1, \ldots, k$ und $m_i = |y_i|$ seien $\bar{v}_i = (v_{i1}, \ldots, v_{im_i})$ die Booleschen Variablen vom Typ i, deren Belegung zum Binärstring y_i korrespondieren soll. Die Hilfsvariablen sind

alle vom Typ k und dienen (wie im klassischen Beweis des Cook'schen Theorems) der Beschreibung von Rechnungen von M_0 auf Eingaben der Form $\langle y_1, \ldots, y_k, x\rangle$. (Es handelt sich also um Variablen, deren Belegungen den jeweiligen Zustand, die jeweilige Kopfposition und die jeweilige Bandinschrift kodieren.) Sei m'_k die Anzahl dieser Hilfsvariablen, welche wir als $\bar{v}'_k = (v'_{k1}, \ldots, v'_{km'_k})$ notieren. Da M_0 eine *polynomielle* DTM ist, ist m'_k polynomiell in $m_1, \ldots, m_k, |x|$ beschränkt. Da $m_i = |y_i|$ polynomiell in $|x|$ beschränkt ist, ist dann schließlich auch m'_k wie auch die Gesamtanzahl $m = m_1 + \cdots + m_k + m'_k$ aller in F vorkommender Boolescher Variablen polynomiell in $|x|$ beschränkt. In Analogie zum Beweis des klassischen Cook'schen Theorems lässt sich nun in pol($|x|$) Schritten eine Formel $F(\bar{v}_1, \ldots, \bar{v}_{k-1}, \bar{v}_k, \bar{v}'_k)$ konstruieren, so dass für alle $x \in L$

$$\begin{aligned} x \in L \;&\Leftrightarrow\; (\exists y_1)_{pol}(\forall y_2)_{pol} \cdots (\exists y_k)_{pol} \langle y_1, \ldots, y_k, x\rangle \in L_0 \\ &\Leftrightarrow\; (\exists y_1)_{pol}(\forall y_2)_{pol} \cdots (\exists y_k)_{pol} M_0 \text{ akzeptiert } \langle y_1, \ldots, y_k, x\rangle \\ &\Leftrightarrow\; \exists \bar{a}_1 \in \{0,1\}^{m_1} \forall \bar{a}_2 \in \{0,1\}^{m_2} \ldots \exists \bar{a}_k \in \{0,1\}^{m_k} \exists \bar{a}'_k \in \{0,1\}^{m'_k} : \\ &\quad F(\bar{a}_1, \ldots, \bar{a}_k, \bar{a}'_k) = 1 \ . \end{aligned}$$

Da wir die letzten beiden $\exists$-Quantoren vor $F(\bar{a}_1, \ldots, \bar{a}_k, \bar{a}'_k)$ verschmelzen können, ist offensichtlich die Abbildung $x \mapsto F$ eine polynomielle Reduktion von L auf $\mathcal{B}_k$. Folglich ist $\mathcal{B}_k$ Σ_k-vollständig. Die Π_k-Vollständigkeit von $\tilde{\mathcal{B}}_k$ ergibt sich (für ungerades k) durch Dualisierung.
Bleibt der Fall eines geraden k zu diskutieren. Hier ist es technisch leichter zunächst die Π_k-Vollständigkeit von $\tilde{B}_k$ zu verifizieren und die Σ_k-Vollständigkeit von B_k durch Dualisierung zu folgern. Es ist nämlich praktischer, wenn die alternierende Quantorenkette erneut mit einem $\exists$-Quantor endet, damit der $\exists$-Quantor für die Hilfsvariablen der Booleschen Formel F mit dem vorangehenden $\exists$-Quantor verschmolzen werden kann. Der Nachweis der Π_k-Vollständigkeit von $\tilde{B}_k$ für gerades k geschieht analog zum Nachweis der Σ_k-Vollständigkeit von $\mathcal{B}_k$ für ungerades k. •

Eine genauere Inspektion des Beweises von Satz 19.20 führt zu **folgender Beobachtung**: **Übg.**

Folgerung 19.21 *Die Einschränkung von $\mathcal{B}_k$ auf CNF-Formeln bei ungeradem k und DNF-Formeln bei geradem k ist Σ_k-vollständig. Dual dazu ist die Einschränkung von $\tilde{B}_k$ auf DNF-Formeln bei ungeradem k und CNF-Formeln bei geradem k Π_k-vollständig.*

Folgerung 19.21 liefert im Falle $k = 1$ gerade das Cook'sche Theorem.

Weiterhin ergibt sich aus $\mathcal{B}_k \in$ *PSpace* und der Σ_k-Härte von $\mathcal{B}_k$ unmittelbar die

Folgerung 19.22 *PH $\subseteq$ PSpace.*

Offene Probleme im Zusammenhang mit PH

P versus PH Sind prädikatenlogische Aussagen, die eine alternierende Quantorenkette konstanter Länge verwenden dürfen, mächtiger als rein aussagenlogische Aussagen?

Kollaps versus Echtheit von PH Sind alternierende Quantorenketten der Länge $k+1$ stets mächtiger als alternierende Quantorenketten der Länge k?

PH versus PSpace Sind alternierende Quantorenketten variabler Länge mächtiger als alternierende Quantorenketten konstanter Länge?

20 Uniforme versus nicht-uniforme Komplexität

Bei einem uniformen Maschinenmodell wenden wir ein einheitliches Programm auf alle möglichen Eingabewörter über einem gegebenen Alphabet an. Wir haben also nicht die Möglicheit, das Programm an die Länge n des Eingabewortes anzupassen. Software-orientierte Maschinenmodelle (wie zum Beispiel die Turing-Maschinen) sind in der Regel von diesem Typ.

Bei einem nicht-uniformen Maschinenmodell darf die „Maschine“ von der Eingabelänge n abhängig sein. Streng genommen hat man dann eine mit n parametrisierte Familie von Maschinen zur Lösung eines vorgegebenen Problems. Hardware-orientierte Maschinenmodelle (wie zum Beispiel Schaltkreisfamilien) sind in der Regel von diesem Typ.

Wir gehen im Folgenden davon aus, dass das Konzept eines Booleschen Schaltkreises (etwa über der Basis $\{\neg, \vee, \wedge\}$) bekannt ist. Zur Konstruktion eines Schaltkreises wird eine endliche Anzahl von Hardwarebausteinen azyklisch miteinander verdrahtet. Jeder Baustein berechnet dabei eine elementare Boolsche Funktionen (wie etwa die logische Negation eines Bits bzw. die logische Disjunktion oder Konjunktion zweier Bits). Durch die Verdrahtung zu einem Schaltkreis kann dann im Prinzip jede Boolesche Funktion $f : \{0,1\}^n \to \{0,1\}$ berechnet werden. Im Folgenden bezeichne

$$C := (C_n)_{n \geq 0}$$

eine Schaltkreisfamilie (kurz: SKF), wobei C_n eine Boolesche Funktion in n Variablen berechnet. Diese Funktion notieren wir der Einfachheit halber als $C_n(x)$ mit $x \in \{0,1\}^n$. Wir sagen C *realisiert* die Sprache $L \subseteq \{0,1\}^*$, wenn

$$\forall n \geq 0, \forall x \in \{0,1\}^n : C_n(x) = 1 \Leftrightarrow x \in L \ .$$

Die *Größe* eines Schaltkreises C_n ist die Anzahl seiner Bausteine und wird als $size(C_n)$ notiert. $C = (C_n)$ heißt *polynomielle SKF*, wenn $size(C_n)$ polynomiell in n beschränkt ist.

Die Nicht-Uniformität eines Maschinenmodells hat weit reichende Folgen. Zum Beispiel gibt es für jede Sprache L (also auch für nicht-entscheidbare oder gar nicht-semi-entscheidbare) eine SKF, welche L realisiert (einfach darum, weil **jede** Boolesche Funktion realisiert werden kann).[63] Beim Vergleich von SKF's und TM's ist es manchmal hilfreich, nur sogenannte „uniforme“ SKF's zuzulassen. Dabei heißt eine SKF $C = (C_n)$ *uniform*, wenn die Abbildung

$$1^n \mapsto desc(C_n)$$

in „logspace“ (also von einer $\log(n)$-platzbeschränkten DTM) berechenbar ist, wobei $desc(C_n)$ eine „natürliche“ Kodierung von C_n bezeichnet.[64] Wegen $\mathcal{L} \subseteq P$ kann eine uniforme SKF

[63] Dies wird freilich bedeuten, dass die Funktion $n \mapsto C_n$ i.A. nicht berechenbar ist.

[64] „desc“ ist Abkürzung von „description“.

nicht superpolynomiell viele Bausteine haben und ist daher polynomiell.
Dieses Kapitel ist aufgebaut wie folgt:

- In den Abschnitten 20.1 und 20.2 zeigen wir, dass jede polynomielle DTM in eine äquivalente uniforme SKF transformiert werden kann und umgekehrt.

- In Abschnitt 20.3 diskutieren wir die Familie $P/poly$ der mit polynomiellen SKF's realisierbaren Sprachen.

- In Abschnitt 20.4 gehen wir erneut auf spärliche (= dünne) Sprachen ein. Es wird sich zeigen, dass $P/poly$ übereinstimmt mit der Klasse der auf spärliche Sprachen Cook-reduzierbaren Sprachen. Außerdem lassen sich die (vermutlich falschen) Aussagen $NP \subseteq P/poly$ bzw. $P = NP$ mit Hilfe von spärlichen und sogenannten kospärlichen Sprachen charakterisieren.

- Im abschließenden Abschnitt 20.5 beweisen wir den Satz von Karp und Lipton, welcher besagt, dass $NP \subseteq P/poly$ zur Folge hätte, dass die polynomielle Hierarchie auf ihren zweiten Level kollabiert.

Hinweis auf laufende Forschung Da vermutlich $NP \not\subseteq P/poly$, ist es naheliegend, nach Sprachen aus NP zu suchen, die sich beweisbar nicht durch polynomielle SKF's berechnen lassen. Wegen $P \subseteq P/poly$ würde dies $P \neq NP$ implizieren. Leider hat sich das Problem, superpolynomielle untere Schranken für (Sprachen aus NP realisierende) SKF's zu beweisen, als außerordentlich hart erwiesen.[65] Aus „Notwehr“ diskutiert man daher hauptsächlich in ihrer Rechenkraft eingeschränkte SKFs (wie zum Beispiel motonone oder tiefenbeschränkte SKFs). Im Rahmen dieser Vorlesung können wir aber darauf nicht näher eingehen.

20.1 Konversion von Software in Hardware

Es sei M eine exakt $S(n)$-platz- und exakt $T(n)$-zeitbeschränkte DTM mit Zustandsmenge Z und Bandalphabet $\Gamma \supset \Sigma$. Zu einer gegebenem Eingabewort $x \in \Sigma^n$ stehe M das Bandsegment mit den Zellen $0, 1, \ldots, S(n), S(n)+1$ zur Verfügung, wobei auf den Zellen 0 und $S(n)+1$ Randmarkierungen stehen, die von M nicht überschritten (und auch nicht gelöscht) werden. Der eigentliche Arbeitsbereich besteht aus den Zellen $1, \ldots, S(n)$. Diese Situation ist in Abbildung 18 illustriert. Anfangs befindet sich M im Startzustand z_0, ihr Kopf ist auf Zelle 0 positioniert und das Eingabewort $x = x_1 \cdots x_n \in \{0,1\}^n$ steht in den Zellen $1, \ldots, n$. Nach exakt $T(n)$ Schritten befindet sich M im Zustand z_+, sofern $x \in L_M$, bzw. im Zustand z_-, sofern $x \notin L_M$. Zu diesem Zeitpunkt ist das Band (bis auf die Randmarkierungen) gesäubert (also leer) und der Kopf befindet sich auf Zelle 0.[66] Jede Konfiguration von M kann in der offensichtlichen Weise durch einen String aus $K^{2+S(n)}$ über dem Zeichenvorrat

$$K := \Gamma \cup (Z \times \Gamma)$$

[65] Bisher sind noch nicht einmal superlineare Schranken bekannt.

[66] Durch diese Normierungen haben wir nicht nur eine eindeutige Anfangskonfiguration, sondern auch eine eindeutige (akzeptierende bzw. verwerfende) Endkonfiguration.

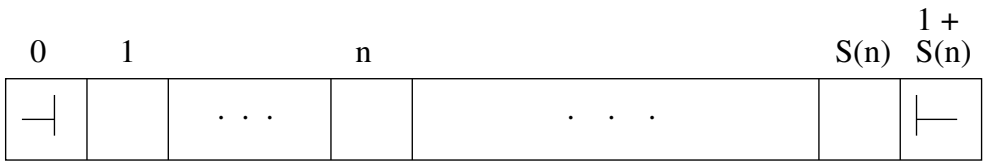


Abbildung 18: Der Arbeitsbereich einer $S(n)$-platzbeschränkten DTM.

beschrieben werden. Das Zeichen aus $Z \times \Gamma$ gibt dabei neben dem aktuellen Zustand auch die Kopfposition an.

Beispiel 20.1 *Die Anfangskonfiguration (zu Eingabewort x) ist beschrieben durch*

$$(z_0, \dashv)x_1 \cdots x_n B \cdots B \vdash$$

und die akzeptierende Endkonfiguration durch

$$(z_+, \dashv)B \cdots B \vdash \ ,$$

wobei „B“ den Blank (= Leerzeichen) bezeichnet.

Die Rechnung von M auf Eingabewort x kann durch die *Rechnungstabelle*

$$R := (R_{i,j})_{0 \le i \le T(n), 0 \le j \le S(n)+1}$$

beschrieben werden. Dabei ist $R_{i,j}$ das j-te Zeichen der i-ten Konfiguration.

Zentrale Beobachtung Wegen der lokalen Arbeitsweise von DTMs ist $R_{i+1,j}$ nur von $R_{i,j-1}, R_{i,j}, R_{i,j+1}$ abhängig, d.h.,

$$R_{i+1,j} = f_\delta(R_{i,j-1}, R_{i,j}, R_{i,j+1}) \ ,$$

wobei f_δ eine durch die Überführungsfunktion δ implizit gegebene Funktion ist.[67]

Jedes Zeichen aus K kann auf einfache Weise mit

$$k := \lceil \log |K| \rceil$$

Bits kodiert werden. Es bezeichne

$$R'_{i,j} \in \{0,1\}^k$$

die Kodierung von $R_{i,j}$. Dann gilt

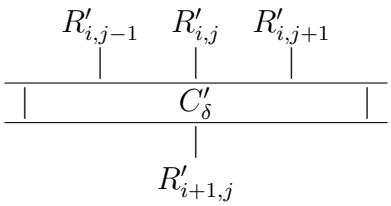
$$R'_{i+1,j} = f'_\delta(R'_{i,j-1}, R'_{i,j}, R'_{i,j+1})$$

für eine (durch δ implizit gegebene) Boolesche Funktion

$$f'_\delta : \{0,1\}^{3k} \to \{0,1\}^k \ .$$

Es bezeichne C'_δ einen Schaltkreis (konstanter Größe), der f'_δ berechnet:

[67] Argument $R_{i,j-1}$ entfällt für $j = 0$ (linker Rand); Argument $R_{i,j+1}$ entfällt für $j = S(n) + 1$ (rechter Rand).



Es sollte klar sein, dass durch Parallelschaltung von $2+S(n)$ Duplikaten von C'_δ (mit Besonderheiten am Rand) ein Schaltkreis C_δ entsteht, der aus der i-ten Zeile der Rechnungstabelle R' die $(i+1)$-te Zeile berechnet. Durch Serienschaltung von $T(n)$ Duplikaten von C_δ entsteht ein Schaltkreis C'_M, der die $T(n)$-te Zeile von R' aus der 0-ten Zeile berechnet. Wenn wir einen Schaltkreis C_{in} voranschalten, der aus $x_1 \cdots x_n$ die $k(2+S(n))$ Bits

$$R'_{0,0} \cdots R'_{0,k(S(n)+2)-1}$$

(also die 0-te Zeile von R') berechnet, und einen Schaltkreis C_{out} hinterherschalten, der die Abbildung

$$R'_{T(n),0} \cdots R'_{T(n),k-1} \mapsto \begin{cases} 1 & \text{falls } R_{T(n),0} = (z_+, \vdash) \\ 0 & \text{falls } R_{T(n),0} = (z_-, \vdash) \end{cases}$$

berechnet, so erhalten wir einen Schaltkreis

$$C_M = C_{out} \circ C'_M \circ C_{in} \ ,$$

der L_M realisiert, d.h.,

$$C_M(x) = 1 \Leftrightarrow M \text{ akzeptiert } x \Leftrightarrow x \in L_M \ .$$

C'_M enthält insgesamt

$$(2 + S(n))T(n) = O(S(n)T(n))$$

Duplikate von C'_δ und hat daher die Größe $O(ST) = O(T^2)$. Es ist nicht schwer zu sehen, dass auch der Gesamtschaltkreis C_M die Größe $O(ST) = O(T^2)$ besitzt und in $O(ST) = O(T^2)$ Schritten sowie im Rahmen einer logarithmischen Platzschranke konstruiert werden kann. Wir ziehen aus der gesamten Diskussion das folgende Fazit:

Satz 20.2 *Falls $L \in P$, dann kann L durch eine uniforme (und somit auch polynomielle) SKF $C = (C_n)$ realisiert werden.*

20.2 Konversion von Hardware in Software

Ziel dieses Abschnittes ist die Umkehrung von Satz 20.2:

Satz 20.3 *Es sei $C = (C_n)$ eine uniforme SKF und*

$$L_C := \{x \in \{0,1\}^* |\ C_{|x|}(x) = 1\}$$

die von ihr realisierte Sprache. Dann gilt $L_C \in P$.

Beweis Es bezeichne CIRCUIT EVALUATION die Sprache

$$\{\langle desc(C_n), a\rangle |\ n \geq 0, a \in \{0,1\}^n \text{ und } C_n(a) = 1\}\ .$$

Das Mitgliedschaftsproblem zu dieser Sprache besteht im Wesentlichen darin, einen gegebenen Schaltkreis C_n (mit n Boolschen Eingabewerten) an einem gegebenen Booleschen Vektor $a \in \{0,1\}^n$ auszuwerten, was natürlich in Polynomialzeit geschehen kann. Es gilt also

$$\text{CIRCUIT EVALUATION} \in P\ .$$

Der Beweis $L_C \in P$ kann daher geführt werden, indem wir wir eine logspace-Reduktion von L_C auf CIRCUIT EVALUATION angeben. Die dazu passende Reduktionsabbildung ist

$$x \mapsto \langle desc(C_{|x|}), x\rangle\ .$$

Sie kann wegen der Uniformität von C in „logspace“ berechnet werden und erfüllt (per Definition der beteiligten Sprachen) die Bedingung

$$x \in L_C \Leftrightarrow \langle desc(C_{|x|}), x\rangle \in \text{CIRCUIT EVALUATION}\ .$$

•

Aus den Sätzen 20.2 und 20.3 ziehen wir die

Folgerung 20.4 *P ist die Klasse aller durch uniforme SKFs realisierbaren Sprachen.*[68]

Weiterhin enthält der Beweis von Satz 20.3 implizit eine logspace-Reduktion von L_C auf CIRCUIT EVALUATION. Dabei steht C für eine beliebige uniforme SKF. In Verbindung mit Folgerung 20.4 ergibt sich daraus leicht:

Folgerung 20.5 *CIRCUIT EVALUATION ist P-vollständig unter logspace-Reduktionen.*

20.3 P/poly: die nicht-uniforme „Schwester“ von P

Definition 20.6 *$P/poly$ bezeichne die Klasse aller Sprachen, die durch polynomielle (aber nicht notwendig uniforme) SKFs realisiert werden können.*

Aus Folgerung 20.4 ergibt sich unmittelbar die die Inklusion $P \subseteq P/poly$. Am Ende dieses Abschnittes werden wir nachweisen, dass diese Inklusion echt ist. Die eigentliche (und bis heute ungeklärte) Frage lautet: wie mächtig ist die Klasse $P/poly$? Wir werden in diesem und den beiden folgendenen Abschnitten etwas Licht auf diese Frage werfen und beginnen mit dem

[68] Streng genommen müssten wir ein binäres Alphabet $\Sigma = \{0,1\}$ zugrunde legen. Für Sprachen über einem erweiterten Alphabet Σ gilt die Folgerung aber für die davon induzierte Sprache über $\{0,1\}$ (nach Festlegung eines Binärcodes für die Symbole von Σ).

Satz 20.7 *Eine Sprache L gehört zur Klasse $P/poly$ gdw ein Polynom q, eine Sprache $L_0 \in P$ und eine Folge $(a_n)_{n\geq 0}$ von binären Strings $a_n \in \{0,1\}^*$ einer durch $q(n)$ beschränkten Länge existieren, so dass*

$$L = \{x|\ \langle a_{|x|}, x\rangle \in L_0\}\ .$$

Beweis Wir beweisen zunächst die Richtung von links nach rechts und setzen daher $L \in P/poly$ voraus. Es sei $C = (C_n)$ eine L realisierende polynomielle SKF. Mit

$$a_n := desc(C_n) \text{ und } L_0 := \{\langle desc(C_{|x|}), x\rangle|\ C_{|x|}(x) = 1\}$$

ergibt sich die gewünschte Charakterisierung von L: die Strings $a_n = desc(C_n)$ haben nämlich eine polynomiel beschränkte Länge, da C eine polynomielle SKF ist und L_0 gehört (als Teilproblem von CIRCUIT EVALUATION) zur Klasse P.
Für die Beweisrichtung von rechts nach links setzen wir jetzt die im Satz beschriebene Charakterisierung von L voraus und weisen $L \in P/poly$ nach. Sei also $q(n) \geq n$ ein Polynom, a_n eine Folge von Strings mit $|a_n| \leq q(n)$, $L_0 \in P$ und $L = \{x|\ \langle a_{|x|}, x\rangle \in L_0\}$. Betrachte eine Boolesche Eingabe $x \in \{0,1\}^n$. Wir können einen Schaltkreis C'_n zur Berechnung von

$$x \mapsto \langle a_n, x\rangle$$

und einen Schaltkreis $C''_{n'}$ mit $n' = |\langle a_n, x\rangle|$ zum Entscheiden von

$$\langle a_n, x\rangle \in L_0?$$

in Serie schalten und erhalten so eine SKF $C = (C_n)$ mit $C_n = C''_{n'} \circ C'_n$, welche L realisiert. Wegen $|a_n| \leq q(n)$ gilt sowohl $|\langle a_n, x\rangle| = O(q(n))$ als auch $size(C'_n) = O(q(n))$. C' ist also eine polynomielle SKF. Zusammen mit $L_0 \in P$ und Satz 20.2 ergibt sich nun, dass C'' und somit auch C eine polynomielle SKF ist. Insgesamt hat sich also $L \in P/poly$ ergeben. •

Die Bits in a_n heißen auch *Hinweisbits (advice bits)*. Satz 20.7 lässt sich dann auch folgendermaßen lesen: die Sprachen aus $P/poly$ sind dadurch charakterisiert, dass sie von einer DTM in Polynomialzeit akzeptiert werden können, wenn diese neben der eigentlichen Eingabe $x \in \Sigma^n$ polynomiell viele zusätzliche Hinweisbits a_n (die von x nur über $n = |x|$ abhängen dürfen) bekommt. So gesehen beruht der Beweis von Satz 20.7 auf zwei Grundideen:

- Gegeben eine uniforme L realisierende SKF: dann kann $desc(C_n)$ einer (C_n simulierenden) DTM in Form von Hinweisbits zugänglich gemacht werden.

- Gegeben der polynomielle Algorithmus zum Erkennen von Strings aus L mit Hilfe zusätzlicher Hinweisbits a_n: dann können die Bits aus a_n in einen (den Erkennungsalgorithmus realisierenden) Schaltkreis C_n „gehardwired“ werden.

Das Verifizieren der Aussage $x \in L$ mit Hinweisbits (wie bei Sprachen aus $P/poly$) und das Verifizieren von $x \in L$ mit Ratebits (wie bei Sprache aus NP) scheinen verwandt zu sein. Wir machen aber auf zwei wichtige Unterschiede aufmerksam:

NP	*P/poly*
Zertifikat y zum Nachweis von $x \in L$ kann in Abhängigkeit von x gewählt werden.	Hinweisbits a_n zum Nachweis von $x \in L$ hängen nur von $n = \|x\|$ ab (also gleiche Hinweisbits für Eingabewörter gleicher Länge).
Für $x \notin L$ existiert kein falsches Zertifikat (kein Ratestring y, der zum Akzeptieren von x führt).	Es könnte falsche Hinweise geben (Strings a'_n mit $\langle a'_n, x\rangle \in L_0$ obwohl $x \notin L$ bzw. Strings a'_n mit $\langle a'_n, x\rangle \notin L_0$ obwohl $x \in L$)

Wir werden sehen, dass *NP* und *P/poly* sehr unterschiedliche Klassen sind. Zum Beispiel sind wegen $NP \subseteq PSpace$ alle Sprachen in *NP* entscheidbar. *P/poly* hingegen enthält u.a. auch nicht entscheidbare bzw. noch nicht einmal semi-entscheidbare Sprachen, wie aus dem nächsten Resultat hervorgeht:

Satz 20.8 *Jede unäre Sprache $L \subseteq \{0\}^*$ gehört zu P/poly.*

Beweis Das Hinweisbit

$$a_n := \begin{cases} 1 & \text{falls } 0^n \in L \\ 0 & \text{falls } 0^n \notin L \end{cases}$$

reicht aus, um L in Polynomialzeit zu entscheiden: die Dekodierung von a_n aus $\langle a_n, x\rangle$ liefert die Antwort. •

In der Vorlesung *Theoretische Informatik* wurden zahlreiche nicht entscheidbare (bzw. nicht einmal semi-entscheidbare) Sprachen $L \subseteq \{0,1\}^*$ präsentiert. Zu $x \in \{0,1\}^*$ bezeichne $N(x) \in \mathbb{N}$ die durch die Bitfolge $1x$ binär kodierte natürliche Zahl. Da die Abbildung

$$x \mapsto 0^{N(x)} \text{ (unäre Kodierung von } N(x))$$

berechenbar ist, ist mit L auch die unäre Sprache

$$L' := \{0^{N(x)} |\ x \in L\}$$

nicht (semi-)entscheidbar. Somit gibt es zu jeder nicht (semi-)entscheidbaren Sprache ein unäres Pendant.

Folgerung 20.9

1. *P/poly enthält insbesondere alle unären nicht (semi-)entscheidbaren Sprachen.*
2. *$P \subset P/poly$.*
3. *P/poly ist keine Teilmenge der semi-entscheidbaren Sprachen und kann daher in keiner uniformen (durch eine Platz- oder Zeitschranke gegebenen) Komplexitätsklasse enthalten sein.*

20.4 Spärliche und kospärliche Sprachen

Wir erinnern an Definition 12.7: die *Dichte* einer Sprache L ist gegeben durch

$$dens_L(n) := |\{x \in L : \ |x| \leq n\}|$$

und L heißt *dünn* oder auch *spärlich*, wenn ihre Dichte polynomiell in n beschränkt ist. In diesem Abschnitt benötigen wir weiterhin die

Definition 20.10 *Eine Sprache $L \subseteq \{0,1\}^*$ heißt* kospärlich*, wenn ihr Komplement $\bar{L} = \{0,1\}^* \setminus L$ spärlich ist.*

Der folgende Satz charakterisiert *P/poly* mit Hilfe von spärlichen Sprachen:

Satz 20.11 *P/poly stimmt überein mit der Klasse der auf spärliche Sprachen Cook-reduzierbaren Sprachen.*

Beweis Wir setzen zunächst $L \in P/poly$ voraus und wollen eine spärliche Sprache S mit $L \leq_T S$ ausfindig machen. Gemäß Satz 20.7 existiert ein Polynom q (mit Koeffizienten aus $\mathbb{N}$), Hinweise $(a_n)_{n\geq 0}$ mit $|a_n| \leq q(n)$ und eine Sprache $L_0 \in P$, so dass

$$L = \{x| \ \langle a_{|x|}, x\rangle \in L_0\} \ .$$

Idee Entwerfe S so, dass das S-Orakel nach den Bits von a_n (wobei $n = |x|$) befragt werden kann. Eine DOTM $M[S]$ kann dann $x \in L$ testen wie folgt:

1. Ermittle a_n bitweise durch $q(n)$ entsprechende Anfragen an das S-Orakel.
2. Setze die polynomiell zeitbeschränkte DTM M_0 für die Sprache L_0 auf $\langle a_n, x\rangle$ an und akzeptiere gdw M_0 akzeptiert.

Kommen wir zur Umsetzung der geschilderten Idee und definieren für alle $n \geq 0$ und alle $i = 1, \ldots, q(n)$ die Strings

$$s_i^n := 0^{i-1}10^{q(n)-i} \in \{0,1\}^{q(n)} \ .$$

Die Sprache S sei gegeben durch

$$S := \{s_i^n| \ n \geq 0, 1 \leq i \leq q(n) \text{ und das } i\text{-te Bit von } a_n \text{ ist } 1\} \ .$$

Auf Anfrage s_i^n muss das S-Orakel mit dem i-ten Bit von a_n antworten.[69] S ist spärlich, da offensichtlich

$$|S \cap \{0,1\}^m| \leq m \ .$$

Damit wäre gezeigt, dass jede Sprache aus *P/poly* auf eine spärliche Sprache Cook-reduzierbar ist.
Für die umgekehrte Beweisrichtung setzen wir $L \leq_T S$ voraus, wobei S eine spärliche Sprache bezeichnet, d.h., es gibt ein Polynom $p(n)$, so dass $|S \cap \{0,1\}^n| \leq p(n)$. Wir haben zu zeigen, dass sich das Wortproblem für L mit Hilfe geeigneter (polynomiell längenbeschränkter) Hinweise $(a_n)_{n\geq 0}$ in Polynomialzeit lösen lässt.

[69] Hierbei nutzen wir aus, dass $q(n)$ (als Polynom mit Koeffizienten aus $\mathbb{N}$) mit $n \in \mathbb{N}_0$ streng monoton wächst. Parameter m hat also maximal ein Urbild n mit $m = q(n)$.

Idee Gib eine Liste L_S aller Elemente von S (bis zu einer geeigneten Maximallänge) als Hinweis. Mit Hilfe von L_S kann das S-Orakel simuliert werden.

Kommen wir zur Umsetzung der Idee. Es sei $M'[S]$ die polynomiell zeitbeschränkte DOTM, die für jedes $x \in \Sigma^n$ in $q(n)$ Schritten entscheiden kann, ob $x \in L$. M' kann maximal $q(n)$ viele maximal $q(n)$ lange Anfragen an das S-Orakel stellen. Sei $s_1, \ldots, s_{r(n)}$ eine Auflistung aller Elemente von S der Maximallänge $q(n)$. Wegen der Spärlichkeit von S ist $r(n) \leq p(q(n))$ polynomiell beschränkt. Setze

$$a_n := \langle s_1, \ldots, s_{r(n)} \rangle$$

und beachte, dass

$$|a_n| = O(q(n)p(q(n))) \ .$$

Nun kann (wie geplant) das Wortproblem für L entschieden werden, indem eine DTM M (ohne Orakel), angesetzt auf Eingabe $\langle a_n, x \rangle$ mit $x \in \Sigma^n$, die Rechnung der DOTM M' auf Eingabe x effizient simuliert. Eine etwaige Anfrage an das S-Orakel kann durch Inspektion von a_n beantwortet werden. •

Wir wissen bereits, dass es vermutlich keine spärliche Sprache gibt, die *NP*-hart (unter Karp-Reduktion) ist, denn nach dem Satz von Mahaney (s. Satz 12.9) hätte dies $P = NP$ zur Folge. Dies schließt aber noch nicht die Existenz von spärlichen Sprachen aus, die *NP*-hart unter (den mächtigeren) Turing-Reduktionen sind. Das folgende Resultat legt jedoch die Vermutung nahe, dass dies nicht der Fall ist:

Folgerung 20.12 *Es gilt $NP \subseteq P/poly$ gdw es eine spärliche Sprache S gibt, die NP-hart unter Turing-Reduktionen ist.*

Beweis Die Voraussetzung $NP \subseteq P/poly$ impliziert, dass SAT $\in P/poly$, was gemäß Satz 20.11 die Existenz einer spärlichen Sprache S mit SAT $\leq_T S$ zur Folge hat. Folglich ist S *NP*-hart unter Turing-Reduktionen.
Sei umgekehrt vorausgesetzt es gäbe eine spärliche und zugleich unter Turing-Reduktionen *NP*-harte Sprache S. Somit gilt $L \leq_T S$ für jede Sprache $L \in NP$. Gemäß Satz 20.11 folgt hieraus $NP \subseteq P/poly$. •

$NP \subseteq P/poly$ würde bedeuten, dass alle Sprachen aus *NP* (inklusive der *NP*-vollständigen) durch SKFs polynomieller Größe realisiert werden können. Es wird jedoch vermutet, dass dies nicht der Fall ist. Wir fassen die diversen Vermutungen noch einmal zusammen:

Vermutung 1 Es gibt keine spärliche Sprache, die *NP*-hart (unter Karp-Reduktionen) ist. (Ansonsten wäre $P = NP$.)

Vermutung 2 Es gibt keine spärliche Sprache, die *NP*-hart unter Turing-Reduktionen ist. (Ansonsten wäre $NP \subseteq P/poly$.)

Die gleichen Vermutungen werden auch für kospärliche Sprachen gehegt. Dies wird (zumindest für Karp-Reduktionen) gestützt durch folgenden

Satz 20.13 *Es gilt $P = NP$ gdw es eine kospärliche Sprache S gibt, die NP-hart (unter Karp-Reduktionen) ist.*

Beweis Wir setzen zunächst $P = NP$ voraus. Die Menge $S := \{0,1\}^* \setminus \{0\}$ ist trivialerweise kospärlich. Darüberhinaus ist S aber auch NP-hart, da jede Sprache $L \in NP = P$ mit der Reduktionsabbildung

$$x \mapsto \begin{cases} 1 & \text{falls } x \in L \\ 0 & \text{falls } x \notin L \end{cases}$$

polynomiell auf S reduzierbar ist.
Für die umgekehrte Beweisrichtung gehen wir von einer kospärlichen und zugleich NP-harten Sprache S aus und weisen $P = NP$ nach, indem wir einen Polynomialzeitalgorithmus A für SAT entwerfen. Wegen der NP-Härte von S gilt SAT $\leq_{pol} S$, sagen wir mit Reduktionsabbildung $R : \Sigma^* \to \Sigma^*$. Der Algorithmus A für SAT, den wir im Folgenden entwerfen, ist ähnlich gestrickt wie der Algorithmus für SAT, den wir im Beweis des Satzes von Berman (s. Satz 12.10) verwendet haben. Wir „recyclen“ ein paar wesentliche Bestandteile dieses Beweises:

- Es bezeichne F eine CNF-Formel über n Booleschen Variablen $v_1, \ldots, v_n$ und $N \geq n$ bezeichne ihre Kodierungslänge. Zu einer partiellen Belegung $a \in \{0,1\}^i$ mit $i \in \{0, \ldots, n\}$ assoziieren wir wieder die vereinfachte CNF-Formel $F[a]$, die sich aus F durch Elimination aller bereits erfüllter Klauseln und aller bereits falsifizierter Literale ergibt. Wir erinnern daran, dass $F[\epsilon] = F$ und $F[a] \in \{0,1\}$ für jedes $a \in \{0,1\}^n$ (die Grenzfälle $i = 0$ bzw. $i = n$). Wir erinnern weiterhin daran, dass $F[a]$ erfüllbar ist gdw $F[a0]$ oder $F[a1]$ erfüllbar ist.

- Es bezeichne T den vollständigen binären Baum der Tiefe n, dessen 2^n Blätter in einer 1-zu-1 Beziehung zu den möglichen Belegungen $a \in \{0,1\}^n$ der Booleschen Variablen $v_1, \ldots, v_n$ stehen. Jeder Knoten der Tiefe i entspricht eindeutig einer partiellen Belegung $a \in \{0,1\}^i$. Wir verwenden für diesen Knoten die Notation $u[a]$.

- Ein Exponentialzeitalgorithmus könnte in einem „top-down pass“ den Baum T aufbauen und in einem „bottom-up pass“ die Knoten des Baumes mit Booleschen Wahrheitswerten (0 oder 1) beschriften, so dass der Knoten $u[a]$ mit 1 beschriftet wird gdw die Formel $F[a]$ erfüllbar ist. Damit wäre die ursprünglich gegebene Formel F erfüllbar gdw die Wurzel von T (also der Knoten $u[\epsilon]$) mit 1 beschriftet ist.

- Anstatt nun aber T (in Exponentialzeit) vollständig aufzubauen, versuchen wir die gleiche algorithmische Grundidee auf einem „Anfangsstück“ von T (einer polynomiell in der Kodierungslänge N der Eingabeformel F beschränkten Größe) durchzusetzen. Dabei werden die Konzepte der „Lazy Evaluation“ und des „Hashing“ eingesetzt.

- Zur Implementierung von „lazy evaluation“ erfolgen die Arbeitsschritte während eines Durchlaufes von T in Präordnung. Wenn aber die (zuerst inspizierte) Formel $F[a0]$ sich als erfüllbar erwiesen hat (der Knoten $u[a0]$ ist dann mit 1 markiert), können wir $F[a]$

als erfüllbar deklarieren (also den Knoten $u[a]$ mit 1 markieren), ohne den von $u[a1]$ induzierten Teilbaum aufzubauen (Idee des „Pruning“).

- Als Hashfunktion verwenden wir die Reduktionsabbildung R. Wenn sich eine Formel $F[a]$ als unerfüllbar erwiesen hat, dann sind auch alle Formeln $F[a']$ mit $R(F[a']) = R(F[a])$ unerfüllbar. Wenn daher in T ein Knoten $u[a]$ mit 0 markiert wird, dann können alle vorher bereits erzeugten Knoten $u[a']$ mit $R(F[a']) = R(F[a])$ ebenfalls mit 0 markiert werden. Neue Knoten $u[a']$, welche die Bedingung $R(F[a']) = R(F[a])$ erfüllen, brauchen gar nicht erst in T' aufgenommen zu werden. (Das Ergebnis „0“ kann sofort im „Record“ des Vaterknotens vermerkt werden.)

Wir verzichten auf eine formale Beschreibung des resultierenden Algorithmus A. Anhand der geschilderten Vorgehensweise sollte klar sein, dass A erstens F akzeptiert (also die Wurzel von T mit 1 markiert) gdw F erfüllbar ist (Korrektheit). Weiterhin sollte klar sein, dass die Laufzeit polynomiell in N beschränkt ist, sofern nur ein polynomiell kleines Anfangsstück T' von T aufgebaut wird. Zu diesem Zweck betrachten wir die „Hashtabelle“

$$H := \{t \in \bar{S} |\ \exists u[a] \in T' :\ u[a] \text{ ist mit 0 markiert und } R(F[a]) = t\}\ .$$

Beachte, dass diese Tabelle wegen der Kospärlichkeit von S eine in $n \leq N$ polynomiell beschränkte Größe hat, sagen wir $|H| \leq q(n)$ für ein Polynom q. Die polynomielle Größenbeschränkung von T' ergibt sich nun leicht aus folgender

Hilfsbehauptung Der Durchlauf in Präordnung habe einen Knoten $u[a] \in T'$ mit $a \in \{0,1\}^i$ und $i \in \{0,\ldots,n-1\}$ erreicht und befinde sich in „top-down“ Richtung. Dann wird während des Aufenthaltes im von $u[a]$ induzierten Unterbaum nach Aufnahme von höchstens $n-i$ neuen Knoten in T' einer der folgenden beiden Fälle eingetreten sein:

Fall 1 $u[a]$ hat eine Boolesche Markierung (0 oder 1) erhalten.

Fall 2 $u[a]$ hat noch keine Boolesche Markierung erhalten, aber die Hashtabelle H hat sich um mindestens 1 vergrößert.

Wir zeigen zunächst, wie unser Beweis mit der Hilfsbehauptung abgeschlossen werden kann. Zu diesem Zweck zerlegen wir den Präordnungsdurchlauf in Arbeits- und Backtracking-Phasen. Zu Beginn einer Arbeitsphase sei der Durchlauf beim Knoten $u[a]$ angelangt (anfangs $u[\epsilon]$) und befinde sich in „top-down“ Richtung. Die Arbeitsphase endet, mit dem Eintreten von Fall 1 bzw. Fall 2. Wenn Fall 1 eintritt, dann belasten wir (beweistechnisch) den Knoten $u[a]$ mit „Kosten“ n. Beachte, dass im Fall 1 die in der Arbeitsphase neu in T' aufgenommenen Knoten (auch in Zukunft) nicht mit Kosten belastet werden, da der Durchlauf den von $u[a]$ induzierten Unterbaum nicht mehr betreten wird. Aus dem selben Grund wird $u[a]$ nur ein einziges Mal mit Kosten n belastet. Wenn Fall 2 eintritt, belasten wir die Hashtabelle H mit Kosten n und nennen die während der Arbeitsphase in T' neu aufgenommenen Knoten *H-generiert*. Nach einer Arbeitsphase beginnt eine (evtl. leere) Backtracking-Phase, die solange

währt, bis der Durchlauf wieder die „top-down“-Richtung anvisiert.[70] Aus der geschilderten Vorgehensweise ergibt sich nun Folgendes:

- Die Anzahl der Knoten in T' ist nach oben beschränkt durch die Summe der Belastungen von Knoten bzw. von der Hashtabelle H.
- Die Hashtabelle kann maximal $q(n)$-mal belastet werden, da sie bei jeder Belastung „zur Belohnung“ einen neuen Eintrag erhält und insgesamt maximal $q(n)$ viele Einträge besitzt.
- Es kann maximal $q(n)n$ viele H-generierte Knoten geben, da jede H-Generierung einer Kollektion von maximal n neuen Knoten mit einem neuen Eintrag in H einhergeht.
- Mit Ausnahme der Wurzel werden nur H-generierte Knoten belastet, und die Belastung pro Knoten beträgt maximal n.

Aus diesen Beobachtungen ergibt sich leicht, dass die Anzahl der Knoten in T' nach oben durch $q(n)n + (1 + q(n)n)n = \text{poly}(n) = \text{poly}(N)$ bechränkt ist.
Der Beweis des Satzes wird nun durch den Beweis der Hilfsbehauptung abgeschlossen. Wir verwenden vollständige Induktion nach $i = n, n-1, \ldots, 1, 0$. Für $i = n$ ist der Durchlauf bei einem Knoten $u[a]$ mit $a \in \{0,1\}^n$ angelangt, d.h., $u[a]$ ist ein Blatt in T. Dann wird $u[a]$ mit 0 oder 1, markiert, d.h., es ist Fall 1 eingetreten. Da $u[a]$ ein Blatt ist, enthält der induzierte Unterbaum keine weiteen Knoten, d.h., es wurden in der betrachteten Arbeitsphase keine neuen Knoten generiert. Die Hilfsbehauptung ist somit für den Knoten $u[a]$ korrekt. Wir betrachten nun einen Index $i < n-1$ und setzen induktiv voraus, dass die Hilfsbehauptung sich für $i+1$ bereits als richtig erwiesen hat. Der Durchlauf ist also bei einem Knoten $u[a]$ mit $a \in \{0,1\}^i$ angelangt und befindet sich in „top-down“ Richtung. Wir betrachten zunächst den Fall, dass $u[a0]$ in T' aufgenommen wird. Dann gilt per Induktionsvoraussetzung, dass nach Aufnahme von höchstens $n - (i+1)$ weiteren Knoten in T' für $u[a0]$ Fall 1 oder Fall 2 eingetreten ist. Wenn für $u[a0]$ Fall 2 eintritt (neues Element in H), dann ist Fall 2 somit auch für den Knoten $u[a]$ nach Aufnahme von höchstens $n - i$ Knoten eingetreten. Wenn für $u[a0]$ Fall 1 eingetreten ist, dann unterscheiden wir zwei Unterfälle. Hat $u[a0]$ die Boolesche Markierung 1, dann wird auch $u[a]$ mit 1 markiert und für $u[a]$ ist dann ebenfalls Fall 1 eingetreten. Hat aber $u[a0]$ die Boolesche Markierung 0, dann ist $R(F[a0])$ ein neues Element in H[71] und für $u[a]$ ist Fall 2 der Hilfsbehauptung eingetreten. Betrachten wir abschließend den Fall, dass $u[a0]$ nicht in T' aufgenommen wird. Dies kann nur daran liegen, dass ein Abgleich mit H ergeben hatte, dass $F[a0]$ nicht erfüllbar ist. In diesem Fall versucht der Präordungsdurchlauf nun sein Glück beim Knoten $u[a1]$. Es gelten nun die analogen Betrachtungen wie zuvor beim Knoten $u[a0]$. Damit ist der induktive Beweis der Hilfsbehauptung abgeschlossen. •

[70] Beachte, dass während des „backtracking“ keine neuen Knoten in T' aufgenommen werden.
[71] Sonst wäre der Knoten $u[a0]$ nicht erzeugt worden.

20.5 Der Satz von Karp und Lipton

Dieser Abschnitt ist dem Beweis des folgenden Resultates gewidmet:

Satz 20.14 (Karp und Lipton, 1980) $NP \subseteq P/poly \Rightarrow PH = \Sigma_2 = \Pi_2$.

Beweis Wir merken zunächst an, dass es ausreicht, $\Pi_2 \subseteq \Sigma_2$ zu zeigen. Hieraus folgt nämlich durch Dualisierung $\Sigma_2 = \text{co-}\Pi_2 \subseteq \text{co-}\Sigma_2 = \Pi_2$ und somit $\Sigma_2 = \Pi_2$, was wiederum (Anwendung 2 des Satzes von Wrathall) $PH = \Sigma_2$ zur Folge hat.
Sei nun L eine beliebige aber fest ausgewählte Sprache aus Π_2. Somit existieren ein Polynom p und eine Sprache $L' \in \Sigma_1 = NP$, so dass

$$L = \{x |\ \forall y \in \{0,1\}^{p(|x|)} : \langle y, x\rangle \in L'\}\ . \tag{43}$$

Wir haben $L \in \Sigma_2$ zu zeigen und gemäß dem Satz von Wrathall reicht es dazu aus, eine Beschreibung der Sprache L vom „$\exists\forall$"-Typ anzufertigen.
Gemäß unserer Voraussetzung $NP \subseteq P/poly$ gehört die Sprache 3-SAT zu $P/poly$. Im Folgenden repräsentiere F_n eine CNF-Formel über den Booleschen Variablen $v_1, \ldots, v_n$ mit Klauseln der Länge höchstens 3 (kurz: 3-Klauseln).[72] Insgesamt gibt es nur $O(n^3)$ solcher 3-Klauseln, weswegen die binäre Kodierungslänge von F_n polynomiell in n, sagen wir durch $q(n)$, nach oben beschränkt ist. Im folgenden identifizieren wir die Formel F_n mit ihrer binären (auf Länge $q(n)$ ausgepolsterten) Kodierung, d.h., $F_n \in \{0,1\}^{q(n)}$. Wegen 3-SAT $\in P/poly$ gibt es eine 3-SAT realisierende polynomielle SKF (C_n^{3-SAT}). Schaltkreis C_n^{3-SAT} erhält also als Eingabe eine CNF-Formel F_n und gibt eine 1 aus gdw F_n erfüllbar ist. Auch C_n^{3-SAT} identifizieren wir mit seiner binären Kodierung, d.h., $C_n^{3-SAT} \in \{0,1\}^{r(n)}$ für ein geeignet gewähltes Polynom $r(n)$. Der Schlüssel zum Beweis liegt in folgender

Definition Eine Kollektion $(C_0, \ldots, C_m)$ von Schaltkreisen, wobei $C_i \in \{0,1\}^{r(i)}$ einen Schaltkreis mit $q(i)$ Eingangsknoten repräsentiert, heißt *3-SAT Tester*, falls

$$\forall i = 0, \ldots, m, \forall F_i \in \{0,1\}^{q(i)} : C_i(F_i) = 1 \Leftrightarrow F_i \in \text{3-SAT}\ .$$

Offensichtlich ist zum Beispiel $(C_0^{3-SAT}, \ldots, C_m^{3-SAT})$ ein 3-SAT Tester.
Wir verwenden jetzt das Konzept des 3-SAT Testers, um, ausgehend von der Beschreibung (43), zu einer neuen Beschreibung (vom „$\exists\forall$-Typ") der Sprache L zu gelangen. Hierzu nutzen wir $L' \leq_{pol}$ 3-SAT aus und bedienen uns einer Reduktionsabbildung

$$\langle y, x\rangle \mapsto F_{s(n)}^{x,y}\ ,$$

die einem String $\langle y, x\rangle$ eine CNF-Formel $F_{s(n)}^{x,y}$ über den Booleschen Variablen $v_1, \ldots, v_{s(n)}$ mit Klauseln der Länge höchstens 3 zuordnet, so dass

$$\langle y, x\rangle \in L' \Leftrightarrow F_{s(n)}^{x,y} \in \text{3-SAT}\ .$$

[72] Es ist hier ausnahmsweise einmal praktischer auch Klauseln mit weniger als 3 Literalen zuzulassen.

Die Beschreibung (43) von L lässt sich dann umschreiben wie folgt:

$$\begin{aligned} L \;=\; \Big\{ x \;\Big|\; & \exists (C_0, \ldots, C_{s(|x|)}) \in \times_{i=0}^{s(|x|)} \{0,1\}^{r(i)}, \forall y \in \{0,1\}^{p(|x|)} : \\ & C_{s(|x|)}\left(F_{s(|x|)}^{x,y}\right) = 1 \text{ und } (C_0, \ldots, C_{s(|x|)}) \text{ ist ein 3-SAT Tester} \Big\} \ . \end{aligned}$$

Die Bedingung $C_{s(|x|)}\left(F_{s(|x|)}^{x,y}\right) = 1$ lässt sich in Polynomialzeit testen (CIRCUIT EVALUATION). Falls die Sprache der 3-SAT Tester zu co-$NP = (\forall)_{pol}[P]$ gehört, dann können wir aus der neuen Beschreibung von L folgern, dass

$$L \in (\exists)_{\mathrm{pol}}(\forall)_{pol}(\forall)_{pol}[P] = (\exists)_{\mathrm{pol}}(\forall)_{pol}[P] \ ,$$

womit der Beweis für $L \in \Sigma_2$ erbracht wäre. Bleibt also nur noch zu zeigen, dass

$$\text{3-SAT Tester} \in (\forall)_{pol}[P] \ .$$

Zu diesem Zweck nutzen wir wieder aus, dass

$$F_n \in \text{3-SAT} \Leftrightarrow F_n[0] \in \text{3-SAT} \vee F_n[1] \in \text{3-SAT} \ ,$$

wobei $F_n[a]$ wieder für die CNF-Formel stehe, die aus F_n vermöge einer partiellen Belegung $a \in \{0,1\}^j$ hervorgeht. Für $(C_0, \ldots, C_m) \in \times_{i=0}^{m} \{0,1\}^{r(i)}$ erhalten wir folgende äquivalente Bedingungen:

$(C_0, \ldots, C_m)$ ist ein 3-SAT Tester

$$\begin{aligned} &\Leftrightarrow \quad \big(\forall F_m \in \{0,1\}^{q(m)} : C_m(F_m) = C_{m-1}(F_m[0]) \vee C_{m-1}(F_m[1])\big) \\ &\qquad \wedge \left((C_0, \ldots, C_{m-1}) \text{ ist ein 3-SAT Tester}\right) \\ &\Leftrightarrow \\ &\ldots \\ &\Leftrightarrow \quad \left(\forall (F_1, \ldots, F_m) \in \times_{i=1}^{m} \{0,1\}^{q(i)}, \forall i \in \{1, \ldots, m\} : \overbrace{C_i(F_i) = C_{i-1}(F_i[0]) \vee C_{i-1}(F_i[1])}^{\text{in Polynomialzeit testbar}} \right) \\ &\qquad \wedge \underbrace{(C_0 \text{ ist ein 3-SAT Tester})}_{\text{in konstanter Zeit testbar}} \end{aligned}$$

Wir haben damit für die Sprache der 3-SAT Tester eine Beschreibung vom „∀-Typ“ gefunden, womit der gesamte Beweis abgeschlossen ist. •

21 Probabilistische Komplexitätsklassen

Eine *probabilistische Turing-Maschine (PTM)* arbeitet wie eine NTM mit zwei Entscheidungsalternativen pro Schritt. Der Unterschied zwischen NTMs und PTMs macht sich technisch erst bei der Definition der von M erkannten Sprache L_M bemerkbar. Eine NTM M akzeptiert einen String $x \in \Sigma^*$ (d.h. $x \in L_M$) gdw M mindestens eine akzeptierende Rechnung auf Eingabe x besitzt. Bei einer PTM hingegen werden wir i.A. fordern, dass es

- auf Eingaben $x \in L_M$ „viele“

- auf Eingaben $x \notin L_M$ „wenige“

akzeptierende Rechnungen gibt. Um die Begriffe „viele“ und „wenige“ zu quantifizieren, sehen wir die Rechnung einer PTM M als zufällig an: in jedem Schritt wird eine der beiden Handlungsalternativen mit Wahrscheinlichkeit $1/2$ gewählt. Wie schon bei NTMs gibt es bei PTMs ein „online“- und ein „offline“-Modell:

online-Nichtdeterminismus In jedem Schritt wählt M nicht-deterministisch eine von (oBdA) zwei Handlungsalternativen aus, sagen wir „Aktion 0“ oder „Aktion 1“.

online-Randomisierung In jedem Schritt bestimmt M ein perfektes Zufallsbit $b \in \{0,1\}$ (etwa durch den Wurf einer perfekten Münze) und wählt dann Aktion b.

offline-Nichtdeterminismus Für eine $T(n)$-zeitbeschränkte NTM M nehmen wir die $T(n)$ nicht-deterministischen Entscheidungen vorweg, indem wir die Eingabe x um ein polynomiell verifizierbares Zertifikat $y \in \{0,1\}^{T(n)}$ ergänzen. M rechnet danach deterministisch auf $\langle y, x\rangle$ (Rate-Verifikationsschema).

offline-Randomisierung Wir stellen uns vor, die $T(n)$-zeitbeschränkte PTM M hätte Zugriff auf einen (bezüglich der uniformen Verteilung) zufälligen String $y \in \{0,1\}^{T(n)}$. Auf $\langle y, x\rangle$ rechnet M dann deterministisch, wobei (in Analogie zu NTMs) im i-ten Schritt die Aktion y_i gewählt wird.

Wir wählen im Folgenden das Modell der offline-Randomisierung und setzen zudem oBdA voraus, dass eine $T(n)$-zeitbeschränkte PTM M auf jeder Eingabe $\langle y, x\rangle$ mit $x \in \Sigma^n$ und $y \in \{0,1\}^{T(n)}$ exakt $T(n)$ Schritte rechnet. Da y bezüglich der uniformen Verteilung auf $\{0,1\}^{T(n)}$ ausgewählt wurde, lassen sich die Wahrscheinlichkeiten zum Akzeptieren bzw. Verwerfen von x durch „Zählen“ ermitteln (Anzahl der günstigen Fälle dividiert durch die Anzahl aller Fälle):

$$\begin{aligned} \Pr_y[M(y,x)=1] &= \frac{|\{y \in \{0,1\}^{T(n)} | M(y,x)=1\}|}{2^{T(n)}} \\ \Pr_y[M(y,x)=0] &= \frac{|\{y \in \{0,1\}^{T(n)} | M(y,x)=0\}|}{2^{T(n)}} \end{aligned}$$

Wir merken kurz an, dass alle auftretenden Wahrscheinlichkeiten Vielfache von $2^{-T(n)}$ sind. Die Schreibweise „$\Pr_y$“ soll stets implizieren, dass y bezüglich der uniformen Verteilung zufällig aus $\{0,1\}^{T(n)}$ ausgewählt wurde (so dass einer $T(n)$-zeitbeschränkten PTM $T(n)$ unabhängige perfekte Zufallsbits zur Verfügung stehen).

Beim Erkennen einer Sprache L kann M folgende Fehler begehen:

Fehler 1. Art M akzeptiert x, obwohl $x \notin L$.

Fehler 2. Art M verwirft x, obwohl $x \in L$.

Die diversen probabilistischen Komplexitätsklassen unterscheiden sich darin, welche Fehlerwahrscheinlichkeiten toleriert werden. Wir führen nun eine allgemeine generische Notation ein, aus welcher sich die gängigen Klassen leicht ableiten lassen:

Definition 21.1 *Seien $0 \leq \alpha < \beta \leq 1$, $\alpha, \beta \in \mathbb{Q}$ und $L \subseteq \{0,1\}^*$. Wir sagen L gehört zur Klasse $R_{\leq\alpha,\geq\beta}$ gdw eine PTM M mit polynomieller Zeitschranke $T(n)$ existiert, so dass für alle $n \geq 0$ und alle $x \in \{0,1\}^n$ folgende Bedingungen gelten:*

$$\begin{aligned} x \notin L \quad &\Longrightarrow \quad \Pr_y[M(y,x) = 1] \leq \alpha \\ x \in L \quad &\Longrightarrow \quad \Pr_y[M(y,x) = 1] \geq \beta \end{aligned}$$

Die Klassen $R_{<\alpha,\geq\beta}$, $R_{\leq\alpha,>\beta}$ und $R_{<\alpha,>\beta}$ sind analog definiert. Für diese Klassen ist auch der Fall $\alpha = \beta$ (mehr oder weniger) sinnvoll.

Im Folgenden nennen wir eine PTM mit einer polynomiellen Zeitschranke kurz eine PPTM.

Dass sich die uns vertrauten Klassen P und NP als Grenzfälle von probabilistischen Komplexitätsklassen darstellen lassen, lehrt folgendes

Beispiel 21.2 *$P = R_{\leq 0,\geq 1}$ und $NP = R_{\leq 0,>0}$.*

Weiterhin merken wir kurz an:

Lemma 21.3 *Alle von Definition 21.1 induzierten probabilistischen Komplexitätsklassen sind Teilklassen von PSpace.*

Beweis Eine Sprache $L \in R_{\leq\alpha,\geq\beta}$ mit einer PPTM M (polynomielle Zeitschranke T) als Akzeptor kann deterministisch mit polynomiell beschränktem Platzverbrauch erkannt werden wie folgt:

1. Gegeben Eingabe $x \in \{0,1\}^n$, starte M auf $\langle y, x\rangle$ für jedes $y \in \{0,1\}^{T(n)}$, benutze aber jeweils das selbe Bandsegment.
2. Zähle nebenbei die Anzahl N_+ der akzeptierenden Rechnungen.
3. Akzeptiere schließlich x gdw $N_+ \geq \beta 2^{T(n)}$.

Für die Klassen $R_{<\alpha,\geq\beta}$, $R_{\leq\alpha,>\beta}$ und $R_{<\alpha,>\beta}$ ist die Argumentation völlig analog. •

In den folgenden Abschnitten diskutieren wir einige Standardbeispiele für probabilistische Komplexitätsklassen:

$$\begin{aligned} PP &= R_{<1/2,>1/2} \\ BPP &= R_{\leq 1/3,\geq 2/3} \\ RP &= R_{\leq 0,\geq 1/2} \\ ZPP &= RP \cap \text{co-}RP \end{aligned}$$

Wir werden später ZPP etwas anders definieren, aber die Definition wird sich als äquivalent zu $ZPP = RP \cap \text{co-}RP$ erweisen.

Heutzutage gilt BPP (mehr noch als die Klasse P) als die Klasse der „praktisch lösbaren" Probleme (unter Einsatz von Randomisierung). Algorithmen, die die Mitgliedschaft eines Problems in der Klasse RP (bzw. ZPP) bezeugen, sind auch unter dem Namen „Monte-Carlo Algorithmen" (bzw. „Las-Vegas Algorithmen") bekannt.

21.1 Die Klasse PP

Es ist nicht sinnvoll, für den Fehler 1. und 2. Art jeweils eine Fehlerwahrscheinlichkeit von 1/2 zuzulassen: dann könnten wir nämlich die Frage

$$x \in L \ ?$$

mit dem Wurf einer perfekten Münze entscheiden.[73] Es ist daher eine Minimalforderung, dass die Fehlerwahrscheinlichkeit für den Fehler 1. und 2. Art zumindest kleiner als 1/2 sein sollte. Dies führt zur Definition der Klasse PP (Probabilistic Polynomial Time):

$$PP = R_{<1/2,>1/2}$$

Die Minimalforderungen, die wir an die Klasse PP richten, sind allerdings zu schwach. Der Hauptgrund hierfür ist, dass eine Maschine, die einen lediglich „exponentiell kleinen Vorsprung vor zufälligem Raten" besitzt, in ihrem statistischen Ein/Ausgabeverhalten nicht effizient von zufälligem Raten unterschieden werden kann.[74] Ein weiteres Indiz dafür, dass PP keine Klasse mit praktikablen Erkennungsalgorithmen ist, werden wir im Satz 21.5 liefern, welcher besagt, dass NP eine Teilklasse von PP ist.

Wir beweisen als kleine Aufwärmübung zunächst das folgende

Lemma 21.4 $PP := R_{<1/2,>1/2} = R_{\leq 1/2,>1/2}$.

Beweis Die Inklusion $R_{<1/2,>1/2} \subseteq R_{\leq 1/2,>1/2}$ ist trivial. Wir zeigen, dass auch die Inklusion

$$R_{\leq 1/2,>1/2} \subseteq R_{<1/2,>1/2}$$

gültig ist. Es sei $L \in R_{\leq 1/2,>1/2}$ und M eine entsprechende PPTM mit Zeitschranke $T(n)$, wobei T ein Polynom mit Koeffizienten aus $\mathbb{N}$ bezeichnet. Da die Wahrscheinlichkeit für das Ereignis „M akzeptiert Eingabe $x \in \{0,1\}^n$" ein Vielfaches von $2^{-T(n)}$ ist, folgern wir

$$\Pr_y[M(y,x)=1] > \frac{1}{2} \Leftrightarrow \Pr_y[M(y,x)=1] \geq \frac{1}{2} + 2^{-T(n)} \ . \tag{44}$$

Wir erweitern M zu einer PPTM M', die zusätzliche Zufallsbits $b = (b_0, b_1, \ldots, b_{T(n)}) \in_R \{0,1\}^{1+T(n)}$ verwendet und arbeitet wie folgt:

[73] Deshalb hatten wir die Klasse $R_{\leq\alpha,\geq\beta}$ nur im Falle $\alpha < \beta$ zugelassen. $R_{\leq 1/2,\geq 1/2}$ wäre die Klasse aller Sprachen!

[74] Wir verzichten an dieser Stelle auf eine Konkretisierung und einen mathematischen Beweis dieser Aussage. Im Prinzip läuft die Argumentation darauf hinaus, dass man viel zu viele Experimente machen müsste, um eine infinitesimal unfaire Münze zuverlässig von einer fairen Münze zu unterscheiden.

1. Falls $b = \vec{0}$ (ein Ereignis der Wahrscheinlichkeit $2^{-(1+T(n))}$)) dann verwerfe x und stoppe. Andernfalls mache weiter.

2. Starte M auf $\langle y, x\rangle$ und entscheide wie M.

Wir verwenden die folgende Rechenregel[75] aus der Wahrscheinlichkeitstheorie:

$$\Pr[A] = \Pr[A|B] \cdot \Pr[B] + \Pr[A|\bar{B}] \cdot \Pr[\bar{B}] \ .$$

Hierbei bezeichnen A, B Ereignisse, $\bar{B}$ ist das Komplementärereignis zu B und Ausdrücke wie $\Pr[A|B]$ sind bedingte Wahrscheinlichkeiten. Dann gilt für alle $x \notin L$:

$$\begin{aligned} \Pr_{yb}[M'(yb,x)=1] &= \overbrace{\Pr_{yb}[M'(yb,x)=1|b=\vec{0}]}^{=0} \cdot \Pr[b=\vec{0}] + \overbrace{\Pr_{yb}[M'(yb,x)=1|b\neq\vec{0}]}^{=\Pr[M(y,x)=1]\leq 1/2} \cdot \overbrace{\Pr[b\neq\vec{0}]}^{<1} \\ &< \frac{1}{2} \ . \end{aligned}$$

Weiterhin gilt (wegen der Subadditivität von Wahrscheinlichkeitsmaßen) für alle $x \in L$:

$$\begin{aligned} \Pr_{yb}[M'(yb,x)=0] &\leq \Pr_{y}[M(y,x)=0] + 2^{-(1+T(n))} \\ &\overset{(44)}{\leq} \frac{1}{2} - 2^{-T(n)} + 2^{-(1+T(n))} \\ &= \frac{1}{2} - 2^{-(1+T(n))} < \frac{1}{2} \end{aligned}$$

und somit

$$\Pr_{yb}[M(yb,x)=1] > \frac{1}{2} \ .$$

PPTM M' bezeugt, dass $L \in R_{<1/2,>1/2}$. •

Wir beschließen diesen Abschnitt mit dem folgenden

Satz 21.5 *$NP \subseteq PP$.*

Beweis Wegen Lemma 21.4 genügt es

$$NP \subseteq R_{\leq 1/2,>1/2}$$

nachzuweisen. Es sei L eine beliebige aber fest ausgewählte Sprache aus $NP = R_{\leq 0,>0}$ und M eine entsprechende PPTM. Wir erweitern M zu einer PPTM M', die ein zusätzliches Zufallsbit b verwendet und arbeitet wie folgt:

1. Falls $b = 1$ (ein Ereignis der Wahrscheinlichkeit $1/2$), dann akzeptiere und stoppe. Andernfalls mache weiter.

[75] genannt Satz der bedingten Wahrscheinlichkeiten

2. Starte M auf $\langle y, x\rangle$ und entscheide wie M.

Offensichtlich gilt für alle $x \notin L$ (welche von M niemals akzeptiert werden):

$$\Pr_{yb}[M(yb, x) = 1] = \frac{1}{2}$$

Unter Verwendung des „Satzes der bedingten Wahrscheinlichkeiten“ erhalten wir für alle $x \in L$ (die von M mit einer positiven, obschon evtl. sehr kleinen, Wahrscheinlichkeit akzeptiert werden):

$$\begin{aligned}\Pr_{yb}[M(yb, x) = 1] &= \frac{1}{2}\Pr_{yb}[M(yb, x) = 1|b = 1] + \frac{1}{2}\Pr_{yb}[M(yb, x) = 1|b = 0] \\ &= \frac{1}{2} \cdot 1 + \frac{1}{2}\underbrace{\Pr_{y}[M(y, x) = 1]}_{>0} > \frac{1}{2}\end{aligned}$$

PPTM M' bezeugt, dass $L \in R_{\leq 1/2, >1/2}$. •

21.2 Die Klasse BPP

Die Lehre aus dem Abschnitt 21.1 ist, dass die Wahrscheinlichkeiten für den Fehler 1. und 2. Art „deutlich“ kleiner als 1/2 sein sollten, damit die randomisierten Entscheidungen (Akzeptieren versus Verwerfen) sich signifikant von zufälligem Raten unterscheiden. Dies ist bei der Definition der Klasse BPP (Bounded-away from 1/2 Probabilistic Polynomial Time) berücksichtigt:

$$BPP = R_{\leq 1/3, \geq 2/3}$$

Um mit dieser technischen Definition vertraut zu werden, zeigen wir zunächst, dass die Auswahl der Konstanten $1/3, 2/3$ willkürlich ist und es nur darauf ankommt die Fehlerwahrscheinlichkeit „deutlich“ von 1/2 abzugrenzen. Anschließend zeigen wir, dass BPP auf dem 2. Level der polynomiellen Hierarchie (oder noch tiefer)[76] angesiedelt ist.

Um die Grenzen der Fehlerwahrscheinlichkeiten auszuloten, welche wir an die Stelle der Konstanten $1/3, 2/3$ setzen können, benötigen wir folgende natürliche Verallgemeinerung von Definition 21.1:

Definition 21.6 *Es seien $\alpha = (\alpha_n)_{n\geq 0}$ und $\beta = (\beta_n)_{n\geq 0}$ zwei Folgen mit $0 \leq \alpha_n < \beta_n \leq 1$ und $\alpha_n, \beta_n \in \mathbb{Q}$ für alle $n \geq 0$. Wir sagen L gehört zur Klasse $R_{\leq\alpha,\geq\beta}$ gdw eine PTM M mit einer polynomiellen Zeitschranke $T(n)$ (also eine PPTM) existiert, so dass für alle $n \geq 0$ und alle $x \in \{0, 1\}^n$ folgende Bedingungen gelten:*

$$\begin{aligned}x \notin L &\implies \Pr_{y}[M(y, x) = 1] \leq \alpha_n \\ x \in L &\implies \Pr_{y}[M(y, x) = 1] \geq \beta_n\end{aligned}$$

Die Klassen $R_{<\alpha,\geq\beta}$, $R_{\leq\alpha,>\beta}$ und $R_{<\alpha,>\beta}$ sind analog definiert.

[76] Sogar $BPP = P$ lässt sich beim derzeitigen Stand des Wissens nicht ausschließen.

Wir zeichnen zwei Nullfolgen $\epsilon = (\epsilon_n)$ und $\delta = (\delta_n)$ aus:

$$\epsilon_n = 2^{-n^l} \text{ und } \delta_n = n^{-k} \tag{45}$$

Hierbei seien $k, l \in \mathbb{N}$ natürlich-zahlige Konstanten. Offensichtlich gilt

$$R_{\leq\epsilon,\geq 1-\epsilon} \subseteq \overbrace{R_{\leq 1/3,\geq 2/3}}^{=BPP} \subseteq R_{\leq 1/2-\delta,\geq 1/2+\delta} \ . \tag{46}$$

ϵ_n ist eine phantastisch kleine Fehlerrate, die mit exponentieller Geschwindigkeit gegen Null konvergiert. $1/2 - \delta_n$ hingegen kommt verdächtig nahe an die Fehlerrate $1/2$ von zufälligem Raten heran.[77] Obwohl bei oberflächlicher Betrachtung eine „Galaxie“ zwischen den Fehlerraten ϵ_n und $1/2 - \delta_n$ zu liegen scheint, werden wir jetzt nachweisen, dass alle in (46) aufgeführten Klassen zu BPP identisch sind. Hierzu genügt es freilich, die Inklusion

$$R_{\leq 1/2-\delta,\geq 1/2+\delta} \subseteq R_{\leq\epsilon,\geq 1-\epsilon}$$

nachzuweisen. In Worten: Eine PPTM M, die lediglich einen polynomiellen Vorteil über zufälliges Raten erzielt, lässt sich in eine PPTM M' mit einer exponentiell kleinen Fehlerrate transformieren. Wir beweisen zu diesem Zweck die folgende (etwas allgemeinere) Aussage:

Satz 21.7 *Es sei $\tau = (\tau_n)_{n\geq 0}$ eine in poly(n) Schritten berechenbare Folge rationaler Zahlen im Intervall von 0 bis 1.*[78] *Weiter sei M eine PPTM, die für alle $n \geq 0$ und alle $x \in \{0,1\}^n$ die folgenden Bedingungen erfüllt:*

$$\begin{aligned} x \notin L \quad &\Longrightarrow \quad \Pr_y[M(y,x) = 1] \leq \tau_n - \delta_n \\ x \in L \quad &\Longrightarrow \quad \Pr_y[M(y,x) = 1] \geq \tau_n + \delta_n \end{aligned}$$

Dann existiert eine PPTM M' für L mit Fehlerrate ϵ (was $L \in R_{\leq\epsilon,\geq 1-\epsilon}$ bezeugt).

Beweis PPTM M' gehe auf Eingabe $x \in \{0,1\}^n$ vor wie folgt:

1. Berechne τ_n aus x. (Hierfür spielt nur die Länge n von x eine Rolle.)
2. Für eine hinreichend große (Präzisierung erfolgt aus didaktischen Gründen später) natürliche Zahl R lasse M R-mal auf Eingabe x laufen (mit jeweils neuen, unabhängigen Zufallsbits) und bestimme dabei die absolute Häufigkeit R_+, mit welcher x von M akzeptiert wird.
3. Akzeptiere x gdw $R_+ \geq \tau_n R$.

Wir machen folgende Beobachtungen:

[77] Da dieser Term jedoch noch um den Kehrwert eines Polynoms von $1/2$ weg-separiert ist, spricht man von „polynomiellem Vorteil über zufälliges Raten“).

[78] D.h., Aus 0^n (der unären Kodierung von n) lässt sich τ_n in poly(n) Schritten berechnen.

- R_+ ist eine binomial verteilte Zufallsvariable (Anzahl der Erfolge bei R unabhängig durchgeführten Bernoulli-Experimenten).
- Falls $x \in L$ (und M daher mit einer Wahrscheinlichkeit von mindestens $\tau_n + \delta_n$ akzeptiert), dann gilt
$$\mathbb{E}[R_+] \geq (\tau_n + \delta_n)R \ .$$
- Falls $x \notin L$ (und M daher mit einer Wahrscheinlichkeit von höchstens $\tau_n - \delta_n$ akzeptiert), dann gilt
$$\mathbb{E}[R_+] \leq (\tau_n - \delta_n)R \ .$$
- M' begeht einen Fehler höchstens dann, wenn die Zufallsvariable R_+ von ihrem Erwartungswert um mindestens $\delta_n R$ abweicht.

Mit Hilfe der (aus der Statistik bekannten)[79] Chernoff-Schranken folgt, dass die Abweichung einer binomial-verteilten Zufallsvariable von ihrem Erwartungswert (bei hinreichend großer Anzahl von Versuchen) ein eher unwahrscheinliches Ereignis ist. Genauer:

$$\Pr[|R_+ - \mathbb{E}[R_+]| \geq \delta_n R] \leq 2e^{-2\delta_n^2 R}$$

Eine kleine Rechnung ergibt

$$2e^{-2\delta_n^2 R} \leq \epsilon_n \Leftrightarrow R \geq \frac{\ln(2/\epsilon_n)}{2\delta_n^2} \ .$$

Wegen $\delta_n = n^{-k}$ und $\epsilon_n = 2^{-n^l}$ ergibt sich nun, dass

$$R := n^{2k+l+1}$$

hinreichend groß ist, um für M' die Fehlerschranke ϵ_n zu garantieren. •

Der Spezialfall $\tau_n = 1/2$ liefert nun die

Folgerung 21.8 *Die Klassen $R_{\leq\epsilon,\geq 1-\epsilon}$ und $R_{\leq 1/2-\delta,\geq 1/2+\delta}$ stimmen beide mit der Klasse $BPP = R_{\leq 1/3,\geq 2/3}$ überein.*

Die Technik, eine hohe Fehlerrate durch wiederholte Anwendung eines Zufallsexperimentes signifikant zu verkleinern, wird „Boosting“ genannt. Bei PPTMs mit beidseitigem Fehler haben wir dabei eine Art „Majoritätsvotum“ in Verbindung mit den „Chernoff-Schranken“ eingesetzt. Bei PPTMs mit einseitigem Fehler wird sich später ein einfacheres Boosting-Argument ergeben.

Wir versuchen im Folgenden, *BPP* in die polynomielle Hierarchie einzuordnen. Zu diesem Zweck definieren wir eine neue Komplexitätsklasse:

[79] s. auch Begleitmaterial zur Vorlesung

Definition 21.9 *Wir sagen eine Sprache $L \subseteq \Sigma^*$ gehört zur Komplexitätsklasse Φ_2 gdw eine Sprache $L_0 \in P$ existiert, so dass für alle $n \geq 0$ und alle $x \in \Sigma^n$ die folgenden Bedingungen gelten:*

$$x \in L \quad \Leftrightarrow \quad (\exists y)_{pol}(\forall z)_{pol} : \langle x, y, z\rangle \in L_0 \tag{47}$$

$$x \notin L \quad \Leftrightarrow \quad (\exists z)_{pol}(\forall y)_{pol} : \langle x, y, z\rangle \notin L_0 \tag{48}$$

Die Klasse Φ_2 wird von Δ_2 und $\Sigma_2 \cap \Pi_2$ „gesandwiched“:

Lemma 21.10 $\Delta_2 \subseteq \Phi_2 \subseteq \Sigma_2 \cap \Pi_2$.

Beweis Wir beweisen zunächst $\Phi_2 \subseteq \Sigma_2 \cap \Pi_2$. Wähle eine beliebige Sprache L aus Φ_2 fest aus. Aus Bedingung (47) lässt sich unmittelbar $L \in \Sigma_2$ ablesen. Aus Bedingung (48) ergibt sich analog $\bar{L} \in \Sigma_2$ und somit $L \in \Pi_2$. Es hat sich insgesamt $L \in \Sigma_2 \cap \Pi_2$ und daher $\Phi_2 \subseteq \Sigma_2 \cap \Pi_2$ ergeben.
Der Rest des Beweises ist der Inklusion $\Delta_2 \subseteq \Phi_2$ gewidmet. Wir wählen eine beliebige Sprache $L \in \Delta_2 = P[NP]$ fest aus. Dann gibt es eine Sprache $L' \in NP$ und eine polynomielle DOTM $M[L']$, welche Eingaben aus L erkennen kann. Wir wählen eine beliebige Eingabe $x \in \Sigma^n$ fest aus. Es seien

$$(g_1, b_1), \ldots, (g_s, b_s)$$

mit

$$b_i = \begin{cases} 0 & \text{falls } g_i \notin L' \\ 1 & \text{falls } g_i \in L' \end{cases}$$

die Fragen und Antworten in der Kommunikation zwischen $M[L']$ und ihrem L'-Orakel. Für alle i mit $g_i \in L'$ bezeichne h_i ein (in Polynomialzeit prüfbares) Zertifikat, das die Mitgliedschaft von g_i in L' bezeugt. Um $L \in \Phi_2$ nachzuweisen, suchen wir nach einer Darstellung von L, die den Bedingungen (47) und (48) genügt. Zu diesem Zweck definieren wir ein *Kommunikationscodewort* W gemäß

$$W := \langle w_1, \ldots, w_s\rangle \ ,$$

wobei

$$w_i := \begin{cases} \langle g_i, 0\rangle & \text{falls } g_i \notin L' \\ \langle g_i, 1, h_i\rangle & \text{falls } g_i \in L' \end{cases} \ .$$

W kodiert die (ggf. um Zertifikate erweiterte) Kommunikation zwischen $M[L']$ und ihrem Orakel. Wir nutzen im Folgenden aus, dass W eine effiziente Simulation von $M[L']$ auf Eingabe x erlaubt. Genauer: wir definieren eine polynomielle DTM M_0 (die im Wesentlichen $M[L']$ simuliert) und zeigen, dass die von ihr induzierte Sprache L_0 den an eine Sprache $L \in \Phi_2$ gerichteten Bedingungen (47) und (48) genügt:

- Eine Eingabe von M_0 heiße *zulässig*, falls sie die Form $\langle x, y, z\rangle$ hat, wobei $y = W$ oder $z = W$ gelten soll.
- M_0 soll eine zulässige Eingabe $\langle x, y, z\rangle$ mit $y = W$ oder $z = W$ akzeptieren gdw $x \in L$.[80]

[80] Bei unzulässigen Eingaben machen wir M_0 keine Vorschriften.

Man überlegt sich leicht, dass „syntaktisch inkorrekte“ Eingaben, die nicht von der Form $\langle x, y, z\rangle$ sind (wobei mindestens einer der Strings y, z von der Syntax her ein „potenzielles“ Kommunikationscodewort sein muss), in Polynomialzeit entlarvt werden. Wir können daher annehmen, dass eine syntaktisch korrekte Eingabe der Form $\langle x, y, z\rangle$ (mit zumindest einem potenziellen Kommunikationscodewort) vorliegt. Ein offensichtliches Dilemma für M_0 besteht darin, dass sie nicht weiß, ob y oder z (oder evtl. keiner von beiden) das korrekte Kommunikationscodewort ist. Vergleichsweise harmlos ist dabei der Fall, dass einer der Strings y, z (oder gar beide) eines der folgenden (in Polynomialzeit erkennbaren) „Fouls“ begeht:

Foul 1 Er weicht bereits syntaktisch von einem Kommunikationscodewort ab.

Foul 2 Er hat die syntaktische Form eines Kommunikationscodewortes, enthält aber nicht exakt die Fragen, die $M[L']$ an ihr Orakel richtet.

Falls weder $y = W$ noch $z = W$ (insbesondere also, falls beide Strings ein Foul begehen), dann darf M_0 sowieso machen, was sie will. Nehmen wir also an, dass eine zulässige Eingabe mit $y = W$ oder $z = W$ vorliegt. Falls einer der Strings ein Foul begeht, dann ist (bei einer zulässigen Eingabe) der andere String das Kommunikationscodewort und M kann die Simulation von $M[L']$ problemlos durchführen.

Böse, böse Was aber, wenn weder y noch z ein Foul begeht?

Man überlegt sich leicht, dass M_0 auch unter diesen (maximal widrigen) Umständen korrekt arbeiten kann, indem sie ein paar Regeln beherzigt:

- Falls y, z zur Anfrage g_i das gleiche Antwortbit enthalten, dann setze die Simulation mit diesem Antwortbit fort.
- Im Konfliktfall benutze das Zertifikat h_i, um

  $$g_i \in L' \ ?$$

  zu testen. Falls die Verfikation gelingt, dann setze die Simulation mit Antwortbit 1 fort. Falls nicht, dann mit Antwortbit 0.

Es hat sich also gezeigt, dass eine DTM M_0 mit den gewünschten Eigenschaften existiert. Es bezeichne L_0 die zugehörige Sprache. Wir wollen nun argumentieren, dass L, L_0 in der Beziehung (47) und (48) zueinander stehen:

- Die Richtung „$\Rightarrow$“ ergibt sich im Falle (47) mit $y = W$ und im Falle (48) mit $z = W$.
- Die Richtung „$\Leftarrow$“ in (47) ergibt sich leicht aus der Richtung „$\Rightarrow$“ in (48). Analog ergibt sich die Richtung „$\Leftarrow$“ in (48) leicht aus der Richtung „$\Rightarrow$“ in (47). **Übg**

Somit hat sich $L \in \Phi_2$ und daher auch $\Delta_2 \subseteq \Phi_2$ ergeben. •

Zum Beweis des Hauptresultates benötigen wir noch das folgende

Lemma 21.11 *Es sei $A \subseteq \{0,1\}^N$ mit $|A| \geq \frac{2}{3}2^N$ und $k = 18N$. Dann existieren $y_1, \ldots, y_k \in \{0,1\}^N$, so dass für alle $z \in \{0,1\}^N$ die Bedingung*

$$|\{i \in \{1,\ldots,k\} : \ y_i \oplus z \in A\}| > \frac{k}{2} \tag{49}$$

erfüllt ist (wobei „$\oplus$“ die komponentenweise Addition modulo 2 bezeichnet).

Beweis Wir verwenden die probabilistische Methode, d.h., wir zeigen, dass es eine echt positive Wahrscheinlichkeit gibt, eine „passende“ Sequenz $\bar{y} := (y_1, \ldots, y_k) \in \{0,1\}^{kN}$ zufällig zu generieren. Wir nennen die Sequenz $\bar{y}$ „gut“ für $z \in \{0,1\}^N$, falls $\bar{y}$ und z die Bedingung (49) erfüllen. Ansonsten heiße $\bar{y}$ „schlecht“ für $z \in \{0,1\}^N$. In dieser Sprechweise ist zu zeigen, dass eine Sequenz $\bar{y} \in \{0,1\}^{kN}$ existiert, die simultan für alle $z \in \{0,1\}^N$ gut ist. Betrachte eine Sequenz $\bar{y}$ die (bezüglich der uniformen Verteilung) zufällig aus $\{0,1\}^{kN}$ gewählt ist. Für ein beliebig aber fest ausgewähltes $z \in \{0,1\}^N$ sei Z_i die Bernoulli-Variable

$$Z_i := \begin{cases} 1 & \text{falls } y_i \oplus z \in A \\ 0 & \text{falls } y_i \oplus z \notin A \end{cases} .$$

Dann sind $Z_1, \ldots, Z_k$ identisch verteilte unabhängige Bernoulli-Variable mit Erfolgswahrscheinlichkeit $p \geq 2/3$. Damit $\bar{y}$ schlecht für z ist, müsste $Z_1 + \cdots + Z_k$ um mindestens $k/6$ vom Erwartungswert $pk \geq 2k/3$ abweichen. Mit den (uns inzwischen bekannten) Chernoff-Schranken (und mit $k = 18N$) ergibt sich, dass die Wahrscheinlichkeit hierfür durch $e^{-2(1/6)^2 k} = e^{-N}$ nach oben beschränkt ist. Mit der Subadditivität von Wahrscheinlichkeitsmaßen folgern wir weiter: die Wahrscheinlichkeit, eine zufällige Sequenz $\bar{y}$ zu generieren, so dass ein $z \in \{0,1\}^N$ existiert, für welche $\bar{y}$ schlecht ist, ist nach oben durch $2^N e^{-N} < 1$ beschränkt. Somit gibt es eine positive Wahrscheinlichkeit, eine Sequenz $\bar{y}$ zu realisieren, die für alle $z \in \{0,1\}^N$ gut ist. •

Nun zum Finale des laufenden Kapitels:

Satz 21.12 *$BPP \subseteq \Phi_2$.*

Beweis Wähle eine beliebige Sprache $L \in BPP$ fest aus. Es sei M eine PPTM für L mit einer exakten polynomiellen Zeitschranke $T(n)$ und einer durch $1/3$ beschränkten Fehlerrate (für die Fehler jeweils beider Arten).

Ziel Darstellung von L gemäß der Definition von Φ_2.

Wähle ein $n \geq 0$ und eine Eingabe $x \in \Sigma^n$ beliebig aber fest aus. Setze $N := T(n)$ und $k := 18N$. Es bezeichne χ_L die charakteristische Funktion für die Sprache L. Die Menge

$$A := \{y \in \{0,1\}^N | \ M(y,x) = \chi_L(x)\}$$

hat eine Mächtigkeit $|A| \geq \frac{2}{3}2^N$, da $|A|$ die Anzahl der korrekten Rechnungen der PPTM M (mit einer durch $1/3$ beschränkten Fehlerwahrscheinlichkeit) auf Eingabe x ist. Es bezeichne L_0 die Sprache bestehend aus allen $\langle x, y, z\rangle$ mit folgenden Eigenschaften:

1. y hat die Form $\langle y_1, \ldots, y_k \rangle$ mit $y_i \in \{0,1\}^N$ für $i = 1, \ldots, k$.

2. z hat die Form $\langle z_1, \ldots, z_k \rangle$ mit $z_i \in \{0,1\}^N$ für $i = 1, \ldots, k$.

3. Eine Mehrheit der k^2 Rechnungen $M(y_i \oplus z_j, x)$ ist akzeptierend.

Da A die Bedingungen von Lemma 21.11 erfüllt, folgt leicht, dass L und L_0 in den Beziehungen (47) und (48) zueinander stehen. Somit gilt $L \in \Phi_2$ und daher auch $BPP \subseteq \Phi_2$.

•

21.3 Die Klasse RP

Die Komplexitätsklassen PP und BPP lassen prinzipiell einen beidseitigen Fehler zu. In diesem Abschnitt lernen wir eine Klasse kennen, bei welcher lediglich ein einseitiger Fehler zugelassen ist. Die Klasse RP (Random Polynomial Time) ist gegeben durch

$$RP = R_{\leq 0, \geq 1/2} \ .$$

Eingaben $x \notin L$ werden also stets verworfen und Eingaben $x \in L$ werden mit einer Wahrscheinlichkeit von mindestens $1/2$ akzeptiert. Eine PPTM, die diesem Kriterium genügt, ist also auf Eingaben $x \notin L$ fehlerfrei. Wir merken kurz an, dass RP wegen

$$RP = R_{\leq 0, \geq 1/2} \subseteq R_{\leq 0, > 0} = NP$$

in NP enthalten ist.

Ähnlich wie im Falle BPP werden wir zeigen, dass die Wahl der Konstante $1/2$ willkürlich ist:

Lemma 21.13 *Für die in (45) definierten Nullfolgen $\epsilon = (\epsilon_n)_{n \geq 0}$ und $\delta = (\delta_n)_{n \geq 0}$ gilt:*

$$R_{\leq 0, \geq 1-\epsilon} = \overbrace{R_{\leq 0, \geq 1/2}}^{=:RP} = R_{\leq 0, \geq \delta} \ .$$

Beweis Wegen der offensichtlichen Inklusion

$$R_{\leq 0, \geq 1-\epsilon} \subseteq R_{\leq 0, \geq 1/2} \subseteq R_{\leq 0, \geq \delta}$$

genügt es,

$$R_{\leq 0, \geq \delta} \subseteq R_{\leq 0, \geq 1-\epsilon}$$

zu beweisen. In Worten: jede PPTM mit einseitigem Fehler, aber einer Fehlerrate von höchstens $1 - \delta$ auf Eingaben aus L, kann in eine PPTM mit einseitigem Fehler transformiert werden, die auch auf Eingaben in L eine exponentiell kleine Fehlerrate aufweist. Sei nun eine

Sprache $L \in R_{\leq 0,\geq \delta}$ mit einer hierzu passenden PPTM M vorgegeben, so dass für alle $n \geq 0$ und alle $x \in \Sigma^n$ gilt:

$$\begin{aligned} x \notin L &\Rightarrow \Pr_y[M(y,x) = 1] = 0 \\ x \in L &\Rightarrow \Pr_y[M(y,x)] = 1] \geq \delta_n \end{aligned}$$

Dann sei M' die PPTM, die auf Eingabe $x \in \Sigma^n$ arbeitet wie folgt:

1. Wende M R-mal (jeweils mit neuen unabhängigen Zufallsbits) auf Eingabe x an (wobei wir R aus didaktischen Gründen erst später festlegen).

2. Falls x in mindestens einer der R Rechnungen akzeptiert wurde, so stoppe akzeptierend. Andernfalls stoppe verwerfend.

Falls $x \notin L$, dann wird x von M R-mal verworfen. Somit stoppt auch M' schließlich verwerfend. Falls $x \in L$, dann macht M' einen Fehler (verwerfendes Stoppen) gdw alle R Rechnungen von M auf Eingabe x verwerfend sind. Da eine einzelne Rechnung von M auf x mit einer Wahrscheinlichkeit von höchstens $1 - \delta_n$ verwerfend ist, ergibt sich für die Wahrscheinlichkeit, dass alle R (unabhängig voneinander ausgeführten) Rechnungen von M auf Eingabe x verwerfend sind (gemäß der Produktformel für die Konjunktion unabhängiger Ereignisse) die obere Schranke

$$(1 - \delta_n)^R < e^{-\delta_n R} \ .$$

(Hierbei wurde die Formel $1 + a \leq e^a$ mit Gleichheit nur für $a = 0$ benutzt, die für alle $a \in \mathbb{R}$ gültig ist.) Eine elementare Rechnung ergibt

$$e^{-\delta_n R} \leq \epsilon_n \Leftrightarrow R \geq \frac{\ln(1/\epsilon_n)}{\delta_n} \ .$$

Wegen $\epsilon_n = 2^{-n^l}$ und $\delta_n = n^{-k}$ ist $R := n^{k+l}$ eine hinreichend große Wahl von Parameter R. Unsere Diskussion hat ergeben, dass M' eine PPTM ist, welche $L \in R_{\leq 0,\geq 1-\epsilon}$ bezeugt. •

Folgerung 21.14 *Für jede Konstante* $0 < c < 1$ *gilt:* $RP = R_{\leq 0,\geq c}$.

21.4 Die Klasse ZPP

Die bisher betrachteten PPTMs lassen für den menschlichen Benutzer einen Rest Unsicherheit übrig. Selbst bei PPTMs mit einseitigem Fehler können wir nicht sicher sein (obschon die Fehlerwahrscheinlichkeit via „Boosting“ vernachlässigbar klein gemacht werden kann), dass eine verworfene Eingabe nicht vielleicht doch zur Sprache gehört und hätte akzeptiert werden sollen. Eine Fehlerwahrscheinlichkeit von, sagen wir, 2^{-1000} ist sicher nicht größer als die Wahrscheinlichekeit, dass alle Kernkraftwerke dieser Erde gleichzeitig ihren Supergau erleben. Allein: auch eine noch so kleine Fehlerwahrscheinlichkeit bleibt ein „Stachel im Fleisch“ des wahren Perfektionisten! Voilá, hier ist nun die Definition der PPTM, die Balsam auf die Wunden aller Puristen und Perfektionisten träufelt:

Definition 21.15 *Eine* fehlerfreie *PPTM M, ist eine PPTM, deren Ausgaben* 0 *(für verwerfende Rechnungen) und* 1 *(für akzeptierende Rechnungen) stets absolut verlässlich sind. Sie verfügt über eine dritte Ausgabe „?“ (für „weiß nicht“), die aber auf jeder Eingabe mit einer Wahrscheinlichkeit von höchstens* $1/2$ *produziert wird.*

Definition 21.16 *Die Klasse ZPP (Zerro Error Probabilistic Polynomial Time) besteht aus allen Sprachen, die eine fehlerfreie PPTM zum Akzeptor haben.*

Wie im Falle von RP ist die Wahl der Konstanten $1/2$ in der Definition von fehlerfreien PPTMs willkürlich. Sie kann durch jede Konstante $0 < c < 1$ (oder auch durch $1 - \epsilon_n$ bzw. δ_n) ersetzt werden, ohne dass die davon induzierte Klasse ZPP sich ändert.

Wenn wir bei einer fehlerfreien PPTM für die Sprache L Ausgabe 0 und Ausgabe 1 vertauschen, erhalten wir eine fehlerfreie PPTM für die Komplementärsprache $\bar{L}$. Somit gilt das

Lemma 21.17 *$ZPP = co\text{-}ZPP$.*

Folgendes Resultat klärt das Verhältnis von RP und ZPP:

Satz 21.18 *$ZPP = RP \cap co\text{-}RP$.*

Beweis Wir weisen zunächst

$$ZPP \subseteq RP$$

nach. Sei $L \in ZPP$ und M eine dazu passende fehlerfreie PPTM. Die PPTM M' arbeite wie M, außer dass statt „?“ stets „0“ ausgegeben werde. Es folgt:

$$\begin{aligned} x \notin L &\Rightarrow \Pr_y[M'(y,x) = 1] = \Pr_y[M(y,x) = 1] = 0 \\ x \in L &\Rightarrow \Pr_y[M'(y,x) = 1] = \Pr_y[M(y,x) = 1] \geq \frac{1}{2} \end{aligned}$$

M' bezeugt, dass $L \in RP$. Somit gilt $ZPP \subseteq RP$.
Durch Dualisierung erhalten wir

$$ZPP = \text{co-}ZPP \subseteq \text{co-}RP,$$

womit dann auch

$$ZPP \subseteq RP \cap \text{co-}RP$$

nachgewiesen wäre.
Bleibt also zu zeigen, dass

$$RP \cap \text{co-}RP \subseteq ZPP \ .$$

Sei $L \in RP \cap \text{co-}RP$. Es sei weiter M die PPTM, welche $L \in RP$ bezeugt und entsprechend $\bar{M}$ die PPTM, welche $L \in \text{co-}RP$ und somit $\bar{L} \in RP$ bezeugt. Wir betrachten die PPTM M', die arbeitet wie folgt:

1. Wende M und $\bar{M}$ auf Eingabe x an.

2. Falls M akzeptiert, gib 1 aus, falls $\bar{M}$ akzeptiert gib 0 aus, und falls weder M noch $\bar{M}$ akzeptiert, gib „?“ aus.

Wegen

$$\begin{aligned} x \notin L &\implies \left(\Pr_y[M(y,x)=1] = 0 \text{ und } \Pr_y[\bar{M}(y,x)=1] \geq \frac{1}{2}\right) \\ x \in L &\implies \left(\Pr_y[\bar{M}(y,x)=1] = 0 \text{ und } \Pr_y[M(y,x)=1] \geq \frac{1}{2}\right) \end{aligned}$$

entscheidet sich M' immer korrekt und gibt mit einer Wahrscheinlichkeit von maximal $1/2$ ein Fragezeichen aus. M' bezeugt, dass $L \in ZPP$, womit auch $RP \cap \text{co-}RP \subseteq ZPP$ bewiesen wäre. •

21.5 Abschluss unter Komplement

Wir hatten bereits angemerkt, dass $ZPP = \text{co-}ZPP$. In diesem Abschnitt gehen wir der Frage des Abschlusses unter Komplement etwas systematischer nach.

Es sei M eine PPTM, die $L \in R_{\leq\alpha,\geq\beta}$ bezeugt und $\bar{M}$ die PPTM, die aus M durch Vertauschen der Ausgaben 0 und 1 hervorgeht. Hieraus ergibt sich

$$x \in \bar{L} \Rightarrow \Pr_y[M(y,x)=1] \leq \alpha \Rightarrow \Pr_y[M(y,x)=0] \geq 1-\alpha \Rightarrow \Pr_y[\bar{M}(y,x)=1] \geq 1-\alpha$$

und

$$x \notin \bar{L} \Rightarrow x \in L \Rightarrow \Pr_y[M(y,x)=1] \geq \beta \Rightarrow \Pr_y[M(y,x)=0] \leq 1-\beta \Rightarrow \Pr_y[\bar{M}(y,x)=1] \leq 1-\beta \ .$$

Offensichtlich bezeugt $\bar{M}$, dass $L \in R_{\leq 1-\beta,\geq 1-\alpha}$. Aus diesen Überlegungen lassen sich folgende Schlüsse ziehen:

Folgerung 21.19 *Es gilt:*

$$\begin{aligned} L \in R_{\leq\alpha,\geq\beta} &\Rightarrow \bar{L} \in R_{\leq 1-\beta,\geq 1-\alpha} \\ L \in R_{\leq\alpha,\geq 1-\alpha} &\Rightarrow \bar{L} \in R_{\leq\alpha,\geq 1-\alpha} \end{aligned}$$

Analoge Aussagen gelten für die Klassen $R_{<\alpha,\geq\beta}$, $R_{\leq\alpha,>\beta}$ und $R_{<\alpha,>\beta}$. Insbesondere gilt

$$PP = \text{co-}PP \text{ und } BPP = \text{co-}BPP \ .$$

Unter den von uns näher diskutierten Klassen ist RP die einzige „asymmetrisch definierte“ Klasse, die vermutlich nicht unter Komplement abgeschlossen ist.

21.6 Die Landschaft der Komplexitätsklassen

Wir wollen ein Bild entwerfen, das die probabilistischen und deterministischen Komplexitätsklassen (sagen wir zwischen $\mathcal{L}$ und *PSpace*) und ihre Querbeziehungen darstellt. Aus der trivialen (und auch schon mehrfach ausgenutzten) Beziehung

$$0 \leq \alpha \leq \alpha' < \beta' \leq \beta \leq 1 \Rightarrow R_{\leq\alpha,\geq\beta} \subseteq R_{\leq\alpha',\geq\beta'}$$

(und den analogen Beziehungen für die Klassen $R_{<\alpha,\geq\beta}$, $R_{\leq\alpha,>\beta}$, $R_{<\alpha,>\beta}$) und aus

$$P = R_{\leq 0,\geq 1},\ RP = R_{\leq 0,\geq 1/2} = R_{\leq 0,\geq 3/4},\ BPP = R_{\leq 1/3,\geq 2/3},\ PP = R_{<1/2,>1/2},\ NP = R_{\leq 0,>0}$$

lässt sich sofort

$$P \subseteq RP \subseteq BPP \subseteq PP \text{ und } RP \subseteq NP$$

ableiten. Wegen

$$ZPP = RP \cap \text{co-}RP,\ P = \text{co-}P,\ BPP = \text{co-}BPP$$

folgt weiterhin

$$P \subseteq ZPP \subseteq RP \subseteq RP \cup \text{co-}RP \subseteq BPP \text{ und } ZPP \subseteq NP \cap \text{co-}NP\ .$$

Da alle probabilistischen Klassen, die unserer generischen Definition genügen, in *PSpace* liegen, gilt insbesondere

$$PP \subseteq PSpace\ .$$

Schließlich sei an die Inklusionen

$$\Delta_2 \subseteq \Phi_2 \subseteq \Sigma_2 \cap \Pi_2 \text{ und } BPP \subseteq \Phi_2$$

erinnert. Wenn wir diese Puzzlesteine zusammensetzen, erhalten wir die „Landschaft der Komplexitätsklassen“ wie sie in Abbildung 19 dargestellt ist.

22 Problemliste

SAT: Satisfiability (Erfüllbarkeitsproblem der Aussagenlogik)

Eingabe: Kollektion $C_1, \ldots, C_m$ von Booleschen Klauseln in n Booleschen Variablen $x_1, \ldots, x_n$. (Eine Boolesche Klausel ist eine Disjunktion von Booleschen Literalen. Ein Boolesches Literal ist eine negierte oder unnegierte Boolesche Variable.)

Frage: Existiert eine Belegung von $x_1, \ldots, x_n$ mit 0 oder 1, die alle Klauseln erfüllt, d.h., die dazu führt, dass $C_1, \ldots, C_m$ zu 1 ausgewertet werden ?

Zertifikat: eine Belegung $a \in \{0,1\}^n$ der n Booleschen Variablen

Verifikation: Überprüfung, dass alle Klauseln durch a erfüllt werden

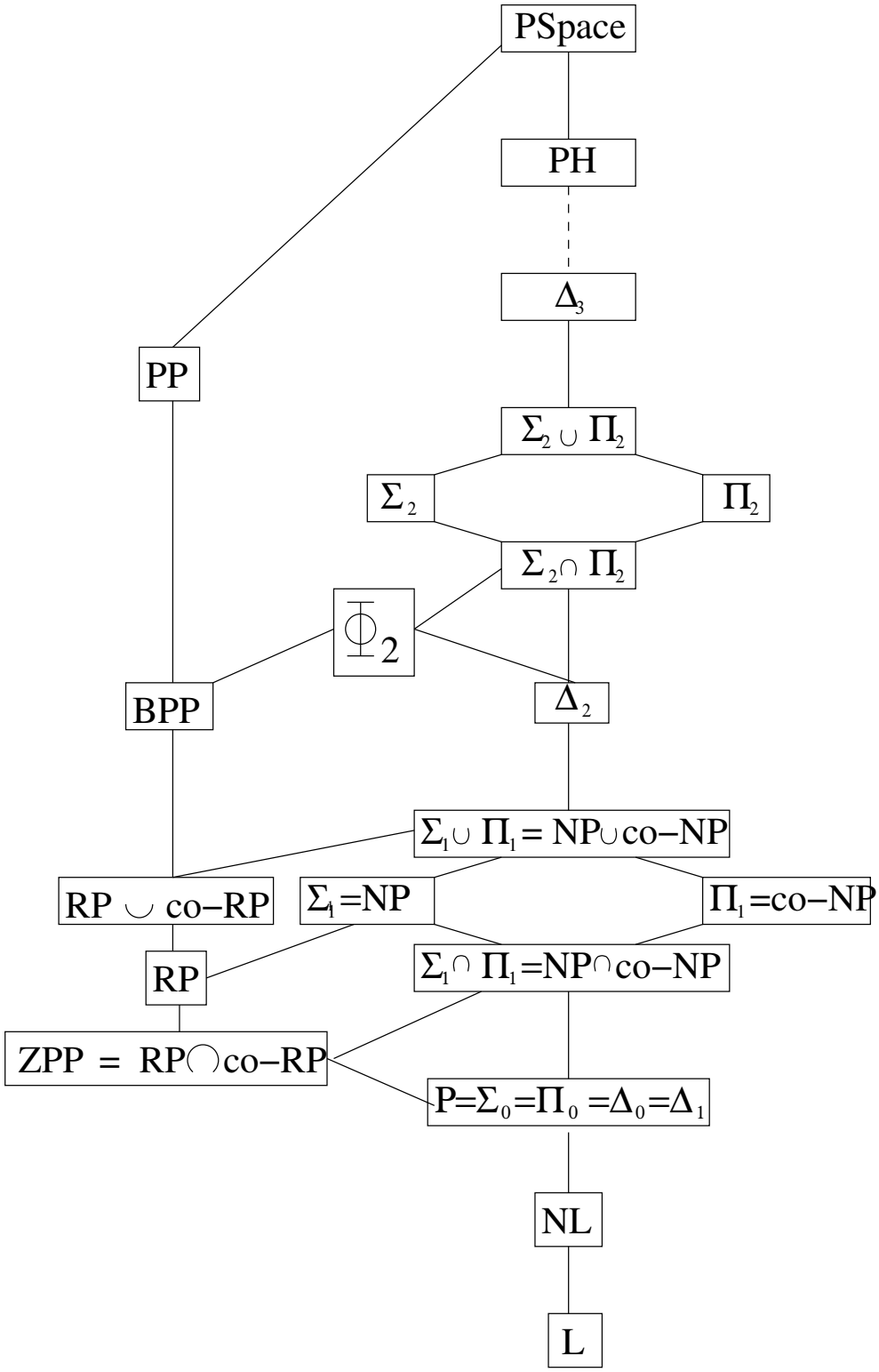


Abbildung 19: Die Landschaft der Komplexitätsklassen

3-SAT: Einschränkung von SAT auf Eingaben, deren Boolesche Klauseln aus jeweils 3 Booleschen Literalen bestehen.

EXACT-ONE 3-SAT: Variante von 3-SAT: es soll entschieden werden, ob eine Belegung existiert, welche jede Klausel mit exakt einem Literal erfüllt.

CLIQUE: Cliquenproblem.

Eingabe: Ein ungerichteter Graph $G = (V, E)$ und eine natürliche Zahl $k \leq |V|$.

Frage: Existiert in G eine Clique der Größe mindestens k, d.h., eine Menge $C \subseteq V$ mit $|C| \geq k$, deren Knoten paarweise in G benachbart sind ?

Zertifikat: eine Menge $C \subseteq V$ der Größe k

Verifikation: Überprüfung, dass alle Knoten in C paarweise benachbart sind

INDEPENDENT SET: Unabhängige Mengen.

Eingabe: Ein ungerichteter Graph $G = (V, E)$ und eine natürliche Zahl $k \leq |V|$.

Frage: Existiert in G eine unabhängige Menge der Größe mindestens k, d.h., eine Menge $U \subseteq V$ mit $|U| \geq k$, deren Knoten paarweise in G nicht benachbart sind ?

Zertifikat: eine Menge $U \subseteq V$ der Größe k

Verifikation: Überprüfung, dass alle Knoten in U paarweise nicht benachbart sind

VERTEX COVER: Überdeckung mit Knoten.

Eingabe: Ein ungerichteter Graph $G = (V, E)$ und eine natürliche Zahl $k \leq |V|$.

Frage: Existiert in G ein „Vertex Cover (Knotenüberdeckungsmenge)“ der Größe höchstens k, d.h., eine Menge $C \subseteq V$ mit $|C| \leq k$, die von jeder Kante aus E mindestens einen Randknoten enthält ?

Zertifikat: eine Menge $C \subseteq V$ der Größe k (oder kleiner)

Verifikation: Überprüfung, dass jede Kante in E mindestens einen Randknoten in C besitzt

HITTING SET: Auffinden eines Repräsentantensystems.

Eingabe: eine Kollektion $M_1, M_2, \ldots, M_m$ endlicher Mengen und eine natürliche Zahl $k \leq m$.

Frage: Gibt es für diese Mengen ein Repräsentantensystem der Größe höchstens k, d.h., eine Menge R mit $|R| \leq k$, die von jeder der Mengen $M_1, M_2, \ldots, M_m$ mindestens ein Element enthält ?

Zertifikat: eine Menge $R \subseteq \cup_{i=1}^{m} M_i$ der Größe k (oder kleiner)

Verifikation: Überprüfung, dass jede Menge M_i durch mindestens ein Element in R repräsentiert ist

SET COVER: Mengenüberdeckung.

Eingabe: eine Kollektion $M_1, M_2, \ldots, M_m$ endlicher Mengen und eine natürliche Zahl $k \leq m$.

Frage: Gibt es eine Auswahl von höchstens k dieser Mengen, deren Vereinigung mit der Vereinigung aller Mengen übereinstimmt, d.h., existiert eine Indexmenge $I \subseteq \{1, \ldots, m\}$ mit $|I| \leq k$ und

$$\bigcup_{i \in I} M_i = \bigcup_{i=1}^{m} M_i \ ?$$

Zertifikat: eine Menge $I \subseteq \{1, \ldots, m\}$ der Größe k (oder kleiner)

Verifikation: Überprüfung, dass die Vereinigung aller Mengen M_i mit $i \in I$ bereits alle Elemente enthält, welche in den Mengen $M_1, \ldots, M_m$ vorkommen

SUBSET SUM: Erzielung einer vorgeschriebenen Teilsumme.

Eingabe: n Zahlen $a_1, \ldots, a_n \in \mathbb{N}$ und eine „Teilsummenzahl“ $S \in \mathbb{N}$.

Frage: Gibt es eine Menge $I \subseteq \{1, \ldots, n\}$, so dass $\sum_{i \in I} a_i = S$?

Zertifikat: eine Menge $I \subseteq \{1, \ldots, n\}$

Verifikation: Überprüfung, dass die Zahlen a_i mit $i \in I$ sich genau zum Wert S aufaddieren

PARTITION: Zerlegung in zwei gleichgroße Teilsummen.

Eingabe: n Zahlen $a_1, \ldots, a_n \in \mathbb{N}$.

Frage: Kann man diese Zahlen in zwei gleichgroße Teilsummen zerlegen, d.h., existiert eine Teilmenge $I \subseteq \{1, \ldots, n\}$, so dass $\sum_{i \in I} a_i = \sum_{j \notin I} a_j$?

Zertifikat: eine Menge $I \subseteq \{1, \ldots, n\}$

Verifikation: Überprüfung, dass die Summe der Zahlen a_i mit $i \in I$ denselben Wert liefer wie die Summe der restlichen Zahlen

KNAPSACK: Rucksackproblem.

Eingabe: n Objekte mit Gewichten $w_1, \ldots, w_n \in \mathbb{N}$ und Nutzen $p_1, \ldots, p_n \in \mathbb{N}$, eine Gewichtsschranke W und eine Nutzenschranke P.

Frage: Kann man einen Rucksack R so packen, dass die Objekte in R einen Gesamtnutzen von mindestens P und ein Gesamtgewicht von höchstens W besitzen, d.h., existiert eine Teilmenge $I \subseteq \{1, \ldots, n\}$, so dass $\sum_{i \in I} p_i \geq P$ und $\sum_{i \in I} w_i \leq W$?

Zertifikat: eine Menge $I \subseteq \{1, \ldots, n\}$

Verifikation: Überprüfung, dass das Gesamtgewicht (bzw. der Gesamtnutzen) aller durch I referenzierten Objekte die Schranke W nicht überschreitet (bzw. die Schranke P nicht unterschreitet)

BP: Bin Packing (Behälterpackungsproblem).

Eingabe: n Objekte der Größen $a_1, \ldots, a_n \in \mathbb{N}$, m Behälter (=bins) der „Bingröße“ $b \in \mathbb{N}$.

Frage: Kann man die n Objekte so in die m Behälter verpacken, dass in jedem Behälter die Größen der in ihm enthaltenen Objekte sich zu höchstens b addieren, d.h., existiert eine Zerlegung von $\{1, \ldots, n\}$ in m disjunkte Teilmengen $I_1, \ldots, I_m$, so dass $\sum_{i \in I_j} a_i \leq b$ für alle $1 \leq j \leq m$ erfüllt ist ?

Zertifikat: m Mengen $I_1, \ldots, I_m \subseteq \{1, \ldots, n\}$

Verifikation: Überprüfung, dass $I_1, \ldots, I_m$ eine Zerlegung von $\{1, \ldots, n\}$ bilden und dass $\sum_{i \in I_j} a_i \leq b$ für alle $1 \leq j \leq m$.

DHP: Directed Hamiltonian Path (Gerichteter Hamiltonscher Pfad).

Eingabe: Ein gerichteter Graph $G = (V, E)$.

Frage: Gibt es in G einen gerichteten Hamiltonschen Pfad, d.h., können wir mit (gerichteten) Kanten aus E einen Pfad formen, der jeden Knoten aus V genau einmal durchläuft ?

Zertifikat: eine Folge $v_1, \ldots, v_n \in V$ der Länge $n = |V|$

Verifikation: Überprüfung, dass $V = \{v_1, \ldots, v_n\}$ und dass $(v_i, v_{i+1}) \in E$ für $i = 1, \ldots, n-1$.

HP: Hamiltonian Path (Hamiltonscher Pfad).
das entsprechende Problem für ungerichtete Graphen

START-ZIEL-DHP Gerichteter Hamiltonscher Pfad mit vorgegebenem Start und Ziel.
Variante von DHP: die Eingabe enthält neben dem gerichteten Graphen $G = (V, E)$ zwei ausgezeichnete Knoten $s, t \in V$ und vom gerichteten Hamiltonschen Pfad wird gefordert, dass er in s beginnt und in t endet.

START-ZIEL-HP Hamiltonscher Pfad mit vorgegebenem Start und Ziel.
die entsprechende Variante von HP

DHC: Directed Hamiltonian Circuit (Gerichteter Hamiltonscher Kreis).

Eingabe: Ein gerichteter Graph $G = (V, E)$.

Frage: Gibt es in G einen gerichteten Hamiltonschen Kreis, d.h., können wir mit (gerichteten) Kanten aus E einen Kreis formen, der (abgesehen von dem identischen Start- und Zielknoten) jeden Knoten aus V genau einmal durchläuft ?

Zertifikat: eine Folge $v_0, \ldots, v_{n-1} \in V$ der Länge $n = |V|$

Verifikation: Überprüfung, dass $V = \{v_0, \ldots, v_{n-1}\}$ und dass $(v_i, v_{i+1 \bmod n}) \in E$ für $i = 0, \ldots, n-1$.

HC: Hamiltonian Circuit (Hamiltonscher Kreis).
das entsprechende Problem für ungerichtete Graphen

TSP: Travelling Salesman Problem (Problem des Handelsreisenden)

Eingabe: Eine Kostenschranke K, n Städte $C_0, \ldots, C_{n-1}$ und eine Distanzmatrix $D = (d_{i,j})_{0 \leq i,j \leq n-1}$, wobei $d_{i,j} \in \mathbb{N}$ die Distanz zwischen C_i und C_j angibt.

Frage: Existiert eine Rundreise durch $C_0, \ldots, C_{n-1}$, deren Gesamtlänge K nicht überschreitet, d.h., existiert eine Permutation σ von $0, \ldots, n-1$, so dass

$$\sum_{i=0}^{n-1} d_{\sigma(i)\sigma(i+1 \bmod n)} \leq K \ ?$$

Zertifikat: eine Permutation σ von $0, \ldots, n-1$

Verifikation: Überprüfung, dass die durch σ gegebene Rundreise die Kostenschranke K nicht überschreitet

Metrisches TSP: Einschränkung von TSP auf Eingaben, deren Distanzmatrix symmetrisch ist und die Dreiecksungleichung erfüllt.