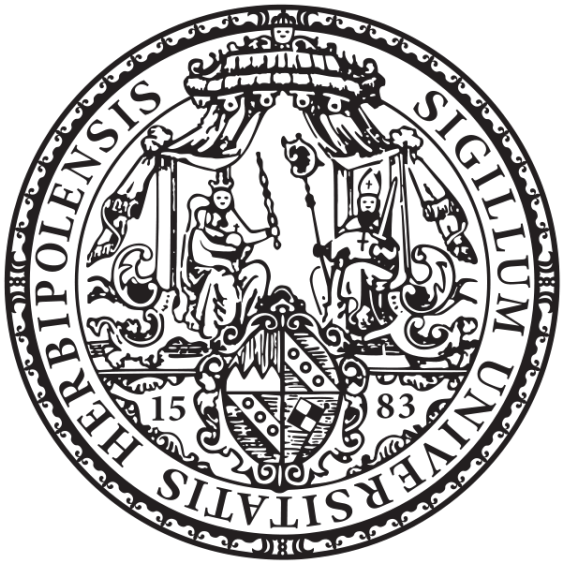
Bachelorarbeit

Separation der relativierten Vermutungen SAT und TFNP

David Dingel

Abgabedatum:	21. November 2022
Betreuer:	Prof. Dr. Christian Glaßer
	Prof. Dr. Jörn Steuding



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

Separation der relativierten Vermutungen SAT und TFNP

Bachelorarbeit

Abstract

Wir untersuchen den Zusammenhang zwischen zwei offenen Vermutungen, SAT und TFNP. Die Vermutung SAT behauptet, dass es kein p-optimales Beweissystem für SAT gebe, während TFNP besagt, dass die Klasse TFNP kein vollständiges Problem habe. Aktuell ist weder bekannt, ob eine der beiden Vermutungen gilt (denn beide würden jeweils $P \neq NP$ implizieren), noch ist bekannt, ob sie äquivalent sind. Pudlák [Pud17] stellte die beiden mit einer Reihe weiterer Vermutungen in Zusammenhang, und fragte nach deren Separation durch Orakel. Verbitskii [Ver91], Glaßer et al. [Gla04], Khaniki [Kha19], Dose und Glaßer [DG19], Dose [Dos20a; Dos20b; Dos21], sowie Egidy und Ehrmanntraut [EEG22] konnten viele der Vermutungen voneinander separieren. Dose [Dos21, zweiter Punkt in Kapitel 3.6] stellte explizit die Frage nach einem Orakel, das SAT und TFNP separiert. Die vorliegende Arbeit beantwortet diese Frage, indem ein Orakel konstruiert wird, bezüglich dessen SAT und ¬TFNP gilt. Bisher war dieses Resultat nicht bekannt. Mit diesem Orakel beantworten wir anschließend vier weitere offene Fragen aus Pudlák's Programm, womit nur noch zwei Fragen ungelöst verbleiben.

Inhaltsverzeichnis

Kapitel 1

Einführung

Die beiden Vermutungen, die in dieser Arbeit untersucht werden, weisen bei genauerer Betrachtung tiefe semantische Ähnlichkeiten auf. Daher liegt die Frage nahe, ob es sich möglicherweise sogar um äquivalente Aussagen handeln könnte. Bisher konnte jedoch nur eine der beiden dazu nötigen Implikationen bewiesen werden. Doch auch wenn es derzeit nicht möglich sein mag, die Frage nach der Äquivalenz zweier komplexitätstheoretischen Aussagen zu beantworten, lässt sich durch sogenannte Orakelseparationen beweisen, dass sie immerhin keine identischen Umformulierungen voneinander sind, sondern sich in gewisser Hinsicht *tatsächlich* unterscheiden. Dieses Phänomen kann so aufgefasst werden, als würden die Aussagen auf bestimmte Modifikationen unterschiedlich reagieren, weil sich hinter einer gleichen Fassade ein unterschiedliches Gerüst verbirgt. Selbst wenn sich die Aussagen schließlich als äquivalent erweisen sollten, würde es sich weiterhin um zwei verschiedene Formulierungen handeln, die nur in diesem einen spezifischen Kontext zum selben Wahrheitsgehalt führten – in einem gewissen anderen Kontext hingegen zu unterschiedlichen. Insbesondere offenbart sich dadurch ein Unterschied zwischen der konkreten Menge an Problemen, aus denen eine Komplexitätsklasse besteht, und der Komplexitätsklasse selbst.

Für nicht mit Orakelseparationen vertraute Leser kann diese Beschreibung gewiss einschüchternd sein. Zwar handelt es sich um ein in der Komplexitätstheorie übliches und mittlerweile Jahrzehnte altes Vorgehen, dennoch vermag das Konzept initial schwer greifbar zu sein. Daher wird sich Abschnitt 1.1 ausführlich mit einer Einführung in die Welt der Orakel befassen, und sowohl den Ablauf als auch den Wert der dadurch erreichbaren Resultate im Detail besprechen.

Das restliche erste Kapitel erklärt die für die späteren Resultate relevanten Vermutungen, sowie die damit verbundenen Komplexitätsklassen. Es soll in erster Linie einen groben Überblick zur Thematik bieten, weshalb darin zugunsten besserer Lesbarkeit bewusst auf technische Details verzichtet wird. Die konkreten mathematischen Definitionen finden sich in Abschnitt 2.1.

Da vor allem die Klasse TFNP für das Verständnis der Resultate dieser Arbeit wichtig ist, folgt in Abschnitt 2.2 dessen genauere Untersuchung, inklusive einer nun mathematisch exakten Begründung für den im ersten Abschnitt heuristisch motivierten Zusammenhang zur Fragestellung $\mathrm{NP} \stackrel{?}{=} \mathrm{coNP}$. Gleichzeitig legen wir dadurch bereits die Grundlage für das zentrale Resultat dieser Arbeit: das in Abschnitt 3.1.1

konstruierte Orakel zur Separation von SAT und TFNP. Unsere Konstruktion wird den engen Zusammenhang von TFNP zu NP $\neq$ coNP ausnutzen, um neben SAT dadurch mit geringerem Aufwand auch $\neg$TFNP zu bewirken. Damit beantworten wir die bislang offene Frage nach einer Separation der beiden Vermutungen.

1.1 Relativierbarkeit

1.1.1 Orakel und relativierbare Beweise

Das Konzept von Orakeln und relativierbaren Beweisen gewann in den siebziger Jahren an Bekanntheit,[1] als ein wegweisendes Resultat für die berühmte P $\stackrel{?}{=}$ NP Frage erreicht wurde: sie kann nicht mit relativierbaren Beweismethoden gelöst werden. Ein relativierbarer Beweis ist sinngemäß ein Beweis, der unabhängig davon ist, welche genauen Informationen einer Maschine in einem einzigen Schritt zur Verfügung stehen. Fast alle derzeit bekannten Resultate aus der Komplexitätstheorie sind relativierbar. Ein Beispiel dafür sind Beweise, welche die Gleichheit zweier Komplexitätsklassen durch Maschinensimulation zeigen. Wir betrachten zur Veranschaulichung den Beweis für die Relation NTIME(f) $\subseteq$ DSPACE(f).[2]

Eine grobe Beweisskizze dazu lautet wie folgt: Sei eine beliebige nichtdeterministische Maschine M gegeben, die auf einer Eingabe n zwar viele verschiedene Rechenwege „gleichzeitig“ durchlaufen mag, jedoch in jedem einzelnen Rechenweg eine durch $f(|n|)$ beschränkte Länge hat. Dann kann M auch mit einem Speicherbedarf von höchstens f nachsimuliert werden, da der selbe Speicher wiederverwendet werden kann, und so alle Rechenwege nacheinander simuliert werden können. Das Verhalten von M kann also exakt repliziert werden, und das sogar deterministisch und mit beschränktem Speicherbedarf. Da nach Definition jede beliebige Sprache aus NTIME(f) von einer solchen Maschine wie M akzeptiert wird, kann jede solche Sprache immer auch mit einem Speicherbedarf von höchstens f entschieden werden, womit die Aussage bewiesen ist.

Interessant ist nun die Erkenntnis, dass der Beweis garnicht darauf eingeht, *was* die Maschine M tatsächlich macht. Jeder Schritt wird einfach ohne zu hinterfragen nachsimuliert. Wie M zu ihren Entscheidungen gekommen ist, und welche besonderen Eigenschaften der von ihr untersuchten Sprache sie dabei ausgenutzt hat, spielt keine Rolle – M hätte dazu buchstäblich ein *Orakel* befragen können, und trotzdem hätte die simulierende Maschine jeden Schritt gehorsam repliziert. Genau dieser Gedanke verbirgt sich hinter dem Konzept relativierbarer Beweise: es bezeichnet diejenigen Beweise, die sogar dann noch anwendbar sind, wenn alle betrachteten Maschinen bei ihren Berechnungen ein beliebiges Orakel befragen dürfen.

Dazu wird ein neuer Begriff eingeführt, der der Orakelturingmaschine. Das ist eine gewöhnliche Turingmaschine, die neben ihren sonstigen Operationen auch eine beliebige Entscheidungsfrage an ein Orakel stellen kann. Diese Fragen können beliebige Strings sein. In jedem Schritt hat die Maschine die Möglichkeit, ihr Orakel B nach dem aktuell auf dem Orakelband stehenden Wort zu befragen. Das Orakel

[1] Wobei erwähnt sei, dass bereits Alan Turing [Tur39] eine Idee dieses Konzeptes beschrieb.

[2] Für $f : \mathbb{N} \to \mathbb{N}$ total mit $f(n) > n$

antwortet `true` oder `false`, wobei es auf die selbe Frage immer konsistent die selbe Antwort liefert. B lässt sich also auch als Menge auffassen, und wenn das Wort w auf dem Orakelband steht, entspricht das Befragen von B der Frage, ob $w \in B$ ist. Eine Orakelturingmaschine kann nun in Abhängigkeit der Antworten unterschiedlich reagieren, weshalb ihr Verhalten vom ihr zur Verfügung gestellten Orakel abhängt. Wir notieren solch eine Maschine als $M^{|\cdot|}$, um zu verdeutlichen, dass die Maschine zunächst nicht an ein bestimmtes Orakel gebunden ist.

Gleichermaßen kann man nun bekannte Komplexitätsklassen, die mit Bezug auf übliche Maschinen definiert sind, so modifizieren, dass sie sich nun stattdessen auf Orakelmaschinen beziehen. Die Klasse P beschreibt beispielsweise alle Sprachen, die in polynomieller Laufzeit von deterministischen Turingmaschinen entschieden werden können. Die Klasse P^B beschreibt dann alle Sprachen, die in polynomieller Laufzeit von deterministischen Orakelturingmaschinen mit Zugriff auf das Orakel B entschieden werden können. P^B beschreibt also Sprachen, die *relativ zum Orakel* B eine polynomielle Komplexität besitzen. Das heißt, falls die in B kodierte Information in jeweils nur einem Rechenschritt erreichbar ist, lässt sich die Sprache mit nur polynomiell vielen Schritten entscheiden.

Eine Orakelseparation von zwei Aussagen oder Klassen bezeichnet dann den Vorgang, ein Orakel B zu konstruieren, relativ zu dem die Eine ungleich der Anderen ist. Zur Separation zweier Klassen $\mathcal{C}$ und $\mathcal{D}$ zeigt man beispielsweise, dass es gewisse Sprachen gibt, die in $\mathcal{C}^B$ sind, jedoch nicht in $\mathcal{D}^B$.

Das Orakel kann beliebig komplexe Informationen liefern, und sogar unentscheidbare Fragen beantworten. Deshalb ist es mit diesem Konstrukt auch erstmals möglich, das Konzept von Willkür handfest umzusetzen. Das möchten wir nun am Beispiel der Resultate von Baker, Gill und Solovay [BGS75] verdeutlichen. Sie konnten darin unter Anderem zeigen, dass es ein Orakel B gibt, relativ zu dem $\mathrm{P}^B \neq \mathrm{NP}^B$. Die entsprechende nichtrelativierte Aussage ist nach wie vor eine der größten offenen Fragen der Mathematik. Versuche, die nichtrelativierte Ungleichheit zu beweisen, scheitern daran, dass es einen geschickten Weg geben könnte, der die erschöpfende Suche unnötig macht.

Betrachten wir das Problem des Handlungsreisenden (kurz TSP): Gegeben eine Landkarte mit Städten, von denen manche durch Straßen gewisser Länge verbunden sind. Der Handlungsreisende möchte jede der Städte besuchen, und will dabei möglichst effizient vorgehen. Seine Fragestellung lautet daher, ob es eine Rundreise gibt, die alle Städte genau ein mal durchläuft, und dabei insgesamt höchstens eine bestimmte Länge hat. Wird dem Handlungsreisenden eine konkrete Route präsentiert, kann dieser mit geringem Aufwand selbst verifizieren, ob es sich dabei um eine korrekte Rundreise handelt. Dieses Problem hat daher die Eigenschaft, dass es im Falle der Existenz einer solchen Rundreise auch kurze Beweise für deren Existenz gibt – nämlich die Rundreise selbst. Falls es also tatsächlich eine gibt, kann man den Handlungsreisenden leicht davon überzeugen.

Weitaus schwieriger wird es jedoch, wenn keine Rundreise existiert – es ist nicht bekannt, ob es auch dafür immer einen kurzen Beweis gibt, oder ob im Zweifelsfall erst dann Gewissheit herrscht, wenn *alle* möglichen Pfade durch Brute Force ausprobiert wurden. Ein in polynomieller Zeit arbeitender Algorithmus hätte dazu nicht die Spur einer Chance, da ihm dafür deutlich zu wenige Rechenschritte zur

Verfügung stehen. Die Frage, ob sich Brute Force vermeiden lässt, ist eine äquivalente Umformulierung der ebenfalls berühmten Frage NP $\stackrel{?}{=}$ coNP.

Um dies nun jedoch für einen Beweis von P $\neq$ NP ausnutzen zu können, müsste gezeigt werden, dass die Rundreisen tatsächlich willkürlich verteilt sind, ein polynomieller Algorithmus also keinen geschickten Trick anwenden kann, um immer an der richtigen Stelle zu suchen. Das ist bisher aber nicht auszuschließen, weshalb die P $\stackrel{?}{=}$ NP Frage nach wie vor offen ist.

Unter Verwendung von Orakeln ist dieser Ansatz jedoch erfolgreich. Der Grund liegt darin, dass die zur Entscheidungsfindung nötigen Argumente nun auch im Orakel liegen können. Darin ist es tatsächlich möglich, Informationen genau so zu platzieren, dass polynomielle Algorithmen niemals alle finden werden.

Während Ursachen für die Existenz bei den ursprünglichen Problemen einem „natürlichen“ Muster folgten (Rundreisen durch Graphen, erfüllende Belegungen für aussagenlogische Formeln, eine Dreifärbung eines Graphen), so können sie im Orakel völlig frei platziert werden, was wir nun an einem Beispiel zeigen. Wir definieren für ein beliebiges Orakel B die Menge G_B sinngemäß wie folgt:

$G_B :=$ „Menge aller Wörter w, für die ein Wort gleicher Länge im Orakel B ist“

Dabei bleibt das Orakel B zunächst unspezifiziert, dieses müssen wir später erst noch konstruieren. Für festes B ließe sich also beweisen, dass ein Wort w in G_B enthalten ist, indem ein Wort b angegeben wird, welches im Orakel B ist und selbe Länge wie w hat. Unter Verwendung des Orakels kann man solch einen Beweis schnell überprüfen, da man nur das Orakel danach zu befragen braucht, ob b darin liegt.

Interessant ist aber vor allem, dass es *keinen* anderen Weg gibt, sich von der Zugehörigkeit von Wörtern zu G_B zu überzeugen, als die von uns behutsam in B platzierten Beweise zu finden. Es gibt keinen „natürlichen“ Grund dafür, dass ein Wort in G_B ist. Stattdessen hängt es einzig und allein davon ab, ob wir uns dazu entscheiden werden, ein Wort der selben Länge im Orakel B zu platzieren – also von unserer *Willkür*. Insbesondere können wir sämtliche „Tricks“ sabotieren, mit denen sich polynomielle Algorithmen von der Nichtexistenz zu überzeugen versuchen könnten: ist ein polynomieller Algorithmus zum Entschluss gekommen, dass kein Beweis existiert, platzieren wir einen an einer der unzähligen Stellen, an denen er nicht nachgeprüft hat. Falls er hingegen von der Existenz eines Beweises überzeugt ist (obwohl wir noch garkeinen hinzugefügt haben), platzieren wir einfach keinen. In beiden Fällen liegt der Algorithmus also falsch.

Zwar hängt G_B nun natürlich vom verwendeten Orakel B ab, jedoch wird G_B immer in NP^B liegen: eine nichtdeterministische Maschine kann bei Eingabe w nach wie vor alle möglichen Beweise (d.h. alle Wörter der selben Länge) „gleichzeitig“ abfragen, und damit immer einen Beweis finden, falls einer existiert. Die polynomielle Maschine hingegen wäre auch hier darauf angewiesen, dass nicht alle Wörter in Frage kämen und auf Zugehörigkeit zum Orakel überprüft werden müssten, also durch ein schlaues Argument nicht überall gesucht werden müsste.

Wenigstens in der relativierten Welt können wir nun aber endlich sicher sein, dass sie diesen Luxus im Allgemeinen nicht hat, denn wir können bei Konstruktion unseres Orakels komplett frei entscheiden, wo wir die Beweise anbringen. Wählt man nun für jede polynomielle Maschine eine Wortlänge, bei der die Beweise genau vor dieser Maschine im Orakel B versteckt werden (beziehungsweise keiner

hinzugefügt wird, wenn sie bereits ein Wort dieser Länge akzeptiert), so wird jede Maschine bei der für sie verwendeten Wortlänge mindestens einen Fehler machen. Damit wäre $G_B \notin \mathrm{P}^B$, obwohl aber $G_B \in \mathrm{NP}^B$, weshalb $\mathrm{P}^B \neq \mathrm{NP}^B$. Wenn zu viele voneinander „unabhängige“ Informationen relevant für die Entscheidungsfindung sind, unterscheiden sich die Klassen also *tatsächlich* in mindestens einem Szenario.

Entsprechend dieser Skizze wird auch das Orakel von Baker, Gill und Solovay [BGS75] konstruiert, wobei jedoch anzumerken ist, dass in unserer Zusammenfassung gewisse relevante technische Details ausgelassen wurden. Die Schwierigkeit dieser Konstruktion, wie auch die der meisten Orakelkonstruktionen, liegt hauptsächlich im Beweis der Wohldefiniertheit, also darin, Widersprüche in der Konstruktion auszuschließen. Problematisch ist dabei vor allem, dass spätere Fragen der Maschine auch von den Antworten auf ihre vorherigen Fragen abhängen können, weshalb das Orakel in der Regel schrittweise konstruiert werden muss. Aber auch darüber hinaus müssen noch weitere entsprechende Vorkehrungen getroffen werden. Auch unsere Konstruktion des Orakels in Abschnitt 3.1.1 wird solche Maßnahmen ergreifen, weshalb wir in diesem einleitenden Kapitel auf eine genauere Ausführung verzichten.

Kehren wir nun stattdessen zurück zu unserem Problem des Handlungsreisenden, und versuchen wir, die eben kennengelernte Orakelseparation auf dieses Beispiel anzuwenden. Dazu ändern wir die Aufgabenstellung geringfügig, indem wir den Straßen zusätzlich auch Namen geben, und manche Straßen von der Nutzung ausschließen. Die Landkarte enthält nun also auch Straßen, die der Handlungsreisende nicht für seine Rundreise nutzen kann. Um festzulegen, ob eine Straße genutzt werden darf oder nicht, verwenden wir die soeben kennengelernte Menge G_B: Eine Straße mit Namen $w \in \Sigma^*$ darf genau dann genutzt werden, wenn $w \in G_B$, d.h. wenn ein $b \in G_B$ mit $|b| = |w|$ existiert. Die neue Fragestellung lautet nun ähnlich wie zuvor – ob eine Rundreise existiert, die nur aus legalen Straßen existiert. Eine solche Rundreise nennen wir „legale Rundreise“.

Auch für diese Abwandlung von TSP gilt weiterhin, dass der Handlungsreisende schnell von der Existenz einer legalen Rundreise überzeugt werden kann (sofern es denn eine gibt). Denn hier muss nach Angabe der Rundreise lediglich bewiesen werden, dass sie legal ist, also ausschließlich erlaubte Straßen verwendet. Und um zu beweisen, dass eine Straße mit Namen w erlaubt ist, genügt schon dasjenige Wort b mit gleicher Länge, das sich im Orakel befindet (denn schließlich ist G_B genau durch solche Beweiswörter definiert). Hängen wir an den ursprünglichen Beweis, also der erfüllenden Rundreise, nun zusätzlich für jede verwendete Straße den Beweis ihrer Nutzbarkeit, kann der Handlungsreisende weiterhin verifizieren, ob es sich um eine legale Rundreise handelt.

Doch kein polynomieller Algorithmus kann nun erkennen, welche Straßen überhaupt erlaubt sind – insbesondere also auch nicht, ob eine legale Rundreise existiert. Zuvor bestand die Schwierigkeit darin, dass ein geschickter Trick existieren könnte, um zu Erkennen, dass keine Rundreise möglich sein kann. Diese Möglichkeit haben wir nun für bestimmte Fälle ausgeschlossen – manchmal liegt der einzige Grund, aus dem keine legale Rundreise existiert, lediglich darin, dass die einzig möglichen Straßen nicht erlaubt sind. Und welche Straßen erlaubt sind unterlag keinem polynomiell erkennbaren Muster, sondern unserer Willkür.

1.1.2 Bedeutung von Orakelkonstruktionen

Da auch wir später in Abschnitt 3.1.1 ein Resultat dieser Natur gewinnen werden, wollen wir nun kurz darauf eingehen, welcher Nutzen aus der Konstruktion von Orakeln gezogen werden kann.

Separation von Konzepten

Orakelseparationen ermöglichen ein objektives, handfestes Argument, dass zwei Aussagen **nicht** nur geringfügig unterschiedliche Formulierungen eines ansonsten identischen Konzeptes sind. Denn durch sie wird gezeigt, dass die beiden Formulierungen unter bestimmten Gegebenheiten tatsächlich zu unterschiedlichen Objekten führen. Wir wollen diese Auffassung nun präziser rechtfertigen.

Dazu möchten wir den alternativen Blickwinkel präsentieren, eine Komplexitätsklasse $\mathcal{C}$ nicht wie üblich als Menge aus Sprachen aufzufassen, sondern als eine *Funktion* $\mathcal{C}(\cdot)$, die jedem Orakel eine Menge aus Sprachen zuordnet:

$$\mathcal{C} : \mathcal{P}(\Sigma^*) \to \mathcal{P}(\mathcal{P}(\Sigma^*)),\ A \mapsto \mathcal{C}(A) := \mathcal{C}^A$$

In Abgrenzung zur ursprünglichen Klasse $\mathcal{C}$ notieren wir die eben definierte Funktion mit $\mathcal{C}(\cdot)$. Die initiale, nichtrelativierte Komplexitätsklasse $\mathcal{C}$ entspricht also $\mathcal{C}(\emptyset)$, denn ein immer mit `false` antwortendes Orakel bringt keinen Mehrwert. Eine mögliche Interpretation für die Funktion $\mathcal{C}(\cdot)$ ist, sie als die „abstrakte Definition“ der Komplexitätsklasse aufzufassen, also als dem ihr zugrunde liegenden Berechnungsmodell. Eine einzelne Auswertung der Funktion auf einem konkreten Argument entspricht dann der zu diesem spezifischen Kontext relativierten Komplexitätsklasse. Stellen wir die beiden Konzepte gegenüber:

- $\mathcal{C}$ charakterisiert die spezifischen Fragen, die in einem ganz spezifischen Kontext mit den gegebenen Mitteln[3] lösbar sind.
- $\mathcal{C}(\cdot)$ charakterisiert für *jeden* Kontext, welche Fragen mit den gegebenen Mitteln lösbar sind.

Über das Konzept zur Lösungsfindung bietet daher $\mathcal{C}(\cdot)$ tiefere Einblicke als $\mathcal{C}$. Der Fokus liegt nun nicht mehr nur auf den spezifischen Fragen, sondern auf dem erlaubten Vorgehen zum Finden von Antworten. Mit dieser Auffassung entspricht für ein Orakel A beispielsweise die ausgewertete Funktion $\mathrm{P}(A)$ denjenigen Fragen, die mit Zugriff auf A in *deterministischer* Polynomialzeit beantwortbar sind; $\mathrm{NP}(A)$ denjenigen, die bei Zugriff auf A in *nichtdeterministischer* Polynomialzeit beantwortbar sind.

Statt wie üblich zwei Komplexitätsklassen $\mathcal{C}$ und $\mathcal{D}$ miteinander zu vergleichen, kann man mithilfe von Orakeln nun $\mathcal{C}(\cdot)$ und $\mathcal{D}(\cdot)$ miteinander vergleichen. Zeigt man deren Ungleichheit, so hat man die beiden Konzepte voneinander separiert – selbst wenn die Gleichheit von $\mathcal{C}$ und $\mathcal{D}$ unbekannt ist. Eine Orakelseparation zeigt dazu die Ungleichheit der beiden Funktionen $\mathcal{C}(\cdot)$ und $\mathcal{D}(\cdot)$, indem ein Orakel A

[3] z.B. der det. Poly-OTM bei P, der nichtdet. Poly-OTM bei NP, dem interaktiven Beweissystem bei IP etc.

konstruiert wird, sodass $\mathcal{C}(A) \neq \mathcal{D}(A)$. Die Frage nach der Gleichheit der Klassen $\mathcal{C}$ und $\mathcal{D}$ entspricht hingegen der Frage, ob die Funktionen auf dem Argument $\emptyset$ den selben Wert annehmen. Zwar ist das für den Vergleich der abstrakten Konzepte nun unerheblich, doch ist diese Frage natürlich im Allgemeinen trotzdem weiterhin von Interesse. Doch darüber lässt sich durch die Orakelseparation jedoch keine Aussage treffen: Neben dem soeben in 1.1.1 konstruierten Orakel B mit $\mathrm{P}^B \neq \mathrm{NP}^B$ existiert auch ein Orakel A mit $\mathrm{P}^A = \mathrm{NP}^A$. Es gilt also $\mathrm{P}(A) = \mathrm{NP}(A)$, aber auch $\mathrm{P}(B) \neq \mathrm{NP}(B)$. Dies zeigt zwar, dass die beiden Funktionsgraphen von $\mathrm{P}(\cdot)$ und $\mathrm{NP}(\cdot)$ weder gleich noch disjunkt sind, doch trotzdem ist die $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ Frage bis heute ungelöst. Tatsächlich konnte sogar gezeigt werden, dass $\mathrm{P}(D) \neq \mathrm{NP}(D)$ mit Wahrscheinlichkeit 1 für ein zufällig aus $\mathcal{P}(\Sigma^*)$ gewähltes Orakel D gilt [BG81]. Doch auch dieses probabilistische Resultat erlaubt keine Rückschlüsse auf die nichtrelativierten Klassen. Die Vermutung, dass aus solch stochastischen Betrachtungen Rückschlüsse gezogen werden könnten, ist als die „Random Oracle Hypothesis“ bekannt, und konnte durch ein Gegenbeispiel widerlegt werden [C94].

Notwendigkeit für nichtrelativierbare Beweise

Orakelkonstruktionen stellen keinen Selbstzweck dar, sondern helfen bei der Suche nach der Lösung der ursprünglichen Frage – ob die Komplexitätsklassen gleich sind. Bleiben wir zunächst beim Beispiel der intensiv untersuchten $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ Frage. Da es sowohl ein Orakel A gibt, relativ zu dem $\mathrm{P}^A = \mathrm{NP}^A$, als auch das eben skizzierte Orakel B, für das $\mathrm{P}^B \neq \mathrm{NP}^B$ gilt, kann für keine der beiden Aussagen ein Beweis existieren, der bezüglich *aller* Orakel gilt. Die $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ Frage kann daher nicht mit relativierbaren Beweismethoden beantwortet werden – weder die Gleichheit, noch die Ungleichheit. Stattdessen ist ein nichtrelativierbarer Beweis nötig.

Nichtrelativierbare Beweismethoden zu finden erweist sich als kreative Herausforderung. Bisher sind nur wenige bekannt, wie beispielsweise die von Shamir für $\mathrm{IP} = \mathrm{PSPACE}$ [Sha92] eingeführte Arithmetisierung.[4] Wir möchten daher kurz beschreiben, was unter einem nichtrelativierbaren Beweis zu verstehen ist.

Wir haben oben bereits erläutert, dass eine Orakelseparation eine tatsächliche Unterscheidung des Berechnungsmodelles (bzw. der durch die Komplexitätsklasse diktierten Einschränkungen) bedeutet. Ein nichtrelativierbarer Beweis der Gleichheit darf sich also nicht nur auf das Berechnungsmodell berufen, sondern muss tiefere Eigenschaften der konkreten Klassen ausnutzen. Im konkreten Fall des Orakels B erkennen wir beispielsweise, dass die Möglichkeit zur nichtdeterministischen Verzweigung prinzipiell tatsächlich ein mächtigeres Werkzeug ist, wenn die für die Entscheidung relevanten Daten willkürlich sind. Um trotzdem die Gleichheit $\mathrm{P} = \mathrm{NP}$ beweisen zu können, müsste man daher insbesondere (wenn auch nur implizit) zeigen, dass sie das *nicht* sind, und deswegen die notwendige Information durch raffiniertes Vorgehen immer auch selbst in Erfahrung gebracht werden könnte.

Doch auch ein Beweis der Ungleichheit darf sich nicht nur auf das Berechnungsmodell beziehen, da es wie im Falle von Orakel A durchaus sein kann, dass eine polynomielle Maschine mit wenig Aufwand alle nötigen Informationen selbst fin-

[4] Leider handelt es sich dabei jedoch um eine „algebraisierende“ Beweismethode, von denen gezeigt werden konnte, dass auch diese keine Antwort auf die $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ Frage liefern können [AW09].

den kann, was vom Beweis (wenn auch nur implizit) ausgeschlossen werden müsste. Vorsichtig formuliert müsste der Beweis also in irgendeiner Weise auf SAT[5] „zugeschnitten“ sein.

Betrachten wir die zuvor eingeführten Funktionen $\mathcal{C}(\cdot)$ und $\mathcal{D}(\cdot)$ zu Komplexitätsklassen $\mathcal{C}$ und $\mathcal{D}$. Während die Frage, ob $\mathcal{C} = \mathcal{D}$, nur der Frage entspricht, ob die Funktionen auf einer einzigen Eingabe den selben Wert annehmen, würde ein relativierbarer Beweis sogar zeigen, dass die Funktionen auf *jedem* Eingabewert gleich (bzw. ungleich) wären. Er müsste also zeigen, dass die Funktionsgraphen gänzlich identisch (bzw. disjunkt) wären. Nicht nur wäre das ein stärkeres Resultat als nötig, sondern ist es im Falle einer Orakelseparation von $\mathcal{C}$ und $\mathcal{D}$ sogar unmöglich. In diesem Fall ist es daher gerechtfertigt, relativierbare Beweise als *zu allgemein* aufzufassen. Sie sind zu weit von der konkreten Fragestellung entfernt und versuchen die Klassen mit zu abstrakten Argumenten zu vergleichen (bzw. zu separieren). Diese Kenntnis hilft bei der Suche nach einem Beweis (wenn auch bisher ohne Erfolg), denn dadurch verstehen wir immerhin, warum bisherige Versuche scheitern mussten. Wir erkennen, dass potenzielle Beweise auf gewisse Weise Bezug zum jeweiligen kombinatorischen Problem nehmen müssen, statt nur distanziert über eine Aufzählung der Maschinen zu diagonalisieren. Allerdings sei hier auf Resultate zu sogenannten „Natural Proofs“ verwiesen, die zeigen, dass unter üblichen kryptographischen Annahmen auch solche Beweise für P $\neq$ NP scheitern, die *zu* spezifisch auf das kombinatorische Problem von SAT zugeschnitten sind [Raz97].

1.2 Untersuchte Klassen

1.2.1 Die Klasse TFNP

Der Name TFNP steht für Total Function NP. TFNP ist eine Klasse sogenannter Suchprobleme – bei einem Suchproblem besteht die Aufgabe bei Eingabe eines Wortes nicht wie bei Entscheidungsproblemen darin, zu prüfen, ob es in einer Menge enthalten ist, sondern ein dazu „passendes“ Wort zu finden. Das spezifische Problem charakterisiert dabei, was mit „passenden“ Wörtern gemeint ist. TFNP ist beispielsweise eine Klasse aus Suchproblemen, bei denen immer die Existenz einer effizient verifizierbaren Lösung garantiert ist. Eine alternative Betrachtungsweise wäre, Suchprobleme aus TFNP als „Urbildsuche für surjektive, effizient berechenbare Funktionen“ zu verstehen.[6] Sind eine solche Funktion f und ein Funktionswert $y \in \Sigma^*$ gegeben, so lautet die Aufgabe, ein nicht all zu großes x zu finden, sodass $f(x) = y$ gilt. Ein typisches Beispiel eines TFNP-Suchproblemes ist das Faktorisieren: ist eine Zahl $y \in \mathbb{N}$ gegeben, suchen wir eine Zerlegung in Primzahlen, deren Produkt genau y ist. *Dass* eine Faktorisierung existiert, steht außer Frage – trotzdem fällt es möglicherweise schwer, sie zu ermitteln.

Es ist nicht sofort klar, warum TFNP einer gesonderten Untersuchung würdig ist.

[5] oder auf ein beliebiges anderes NP-vollständiges Problem

[6] Wobei wir hier aus Gründen der Lesbarkeit ausgelassen haben, dass sogar ein polynomiell beschränktes Urbild existieren muss, und mehrere Funktionswerte erlaubt sein können. Effiziente Berechenbarkeit impliziert bei uns immer die Totalität.

Schließlich ist die Definition sehr ähnlich zur Definition typischer NP-Suchprobleme,[7] sie unterscheidet sich lediglich darin, dass bei TFNP immer eine Lösung existiert. Dies spiegelt sich auch in der Namensgebung Total Function NP wieder. Doch genau dieses subtile Detail hat zur Folge, dass es bisher noch nicht gelang, die Schwierigkeiten der beiden Klassen gleichzusetzen – denn möglicherweise wird die Suche durch die Erfolgsgarantie tatsächlich leichter. Dieses Konzept wollen wir im Folgenden näher ausführen.

Beginnen wir dazu mit einer textuellen Definition von TFNP (die kompakte, mathematisch exakte Formulierung befindet sich in Definition 2.10). Jede TFNP Instanz ist durch eine zweistellige, effizient prüfbare Relation $R \in (\mathbb{P}(\Sigma^* \times \Sigma^*) \cap \mathrm{P})$ charakterisiert, die für jedes y die möglichen Lösungen festlegt. Zu gegebenen x, y lässt sich also in polynomieller Zeit überprüfen, ob x mit y in Relation steht, d.h. ob $(x, y) \in R$. Wir interpretieren $(x, y) \in R$ als „x ist eine Lösung für y“. Außerdem ist garantiert, dass es zu jedem y auch tatsächlich ein nicht viel größeres x gibt, das mit ihm in Relation steht. TFNP Instanzen lassen sich also als ein Tupel (R, p) darstellen, wobei $R \in \mathrm{P}$ der Funktionsgraph und p das Polynom ist, das garantiert, dass eine Lösung dieser Länge existiert. Präziser formuliert muss (R, p) folgendes erfüllen:

$$\forall y \in \Sigma^* \; \exists x \in \Sigma^{\leq p(|y|)} : (x, y) \in R \tag{1.1}$$

Die zum Problem (R, p) assoziierte algorithmische Aufgabenstellung lautet nun, zu gegebenem y ein passendes x zu finden, sodass $(x, y) \in R$. Zu einem Suchproblem $T = (R, p)$ bezeichnen wir das Lösen dieser Aufgabenstellung als das „Lösen des Suchproblemes T“. TFNP ist nun die Menge aller Suchprobleme $(R, p) \in (\mathbb{P}(\Sigma^* \times \Sigma^*) \cap \mathrm{P}) \times \mathbb{N}[X]$, die (1.1) erfüllen.

Zunächst mag es so wirken, als wäre die Existenz eines Urbildes eine unwichtige Zusatzinformation, welche die Fragestellung nicht wesentlich leichter macht. Beispielsweise stellt sich bei der recht unhandlichen Formulierung „Urbildsuche für eine surjektive, effizient berechenbare Funktion“ direkt die Frage, ob man darin auf die Einschränkung der Surjektivität verzichten könnte, und stattdessen nur für solche Werte ein korrektes Verhalten verlangt, für die ein Urbild existiert. Zwar mag dies initial äquivalent erscheinen, doch verbirgt sich darin ein Fallstrick: das gefundene potenzielle Urbild ist effizient verifizierbar. Deswegen könnten unter Verwendung von Algorithmen, die TFNP Suchprobleme lösen, nicht nur surjektive Funktionen invertiert, sondern auch erkannt werden, *ob* eine beliebige effizient berechenbare Funktion ein Urbild für einen Wert y hat. Die Klasse TFNP enthielte also insbesondere Probleme, deren Lösung die Fähigkeit zur Beantwortung NP-vollständiger Entscheidungsprobleme erfordert.[8] Beliebige TFNP Probleme lösen zu können wäre ein Werkzeug, das mindestens so mächtig wäre, wie ein Algorithmus zur Lösung beliebiger NP Probleme. Später werden wir mit Satz 2.13 zeigen, dass die Surjektivität

[7] „Finde eine Rundreise mit Kosten $\leq k$, falls eine solche existiert“ etc.

[8] Denn ohne die Notwendigkeit zur Surjektivität könnte für eine Maschine M, die SAT akzeptiert, die Relation $\{(x, y) \mid x \text{ ist akzeptierender Rechenweg der Maschine } M \text{ auf Eingabe } y\}$ gewählt werden, um eine TFNP Instanz zu erzeugen. Diese wäre dann NP hart unter der Turing Reduktion, weshalb dann nach Satz 2.13 NP = coNP. Abschnitt 2.2 wird sich genauer mit der potenziellen NP-Härte befassen.

in der Definition genau dann weggelassen werden kann, wenn NP = coNP gilt. An dieser Stelle wollen wir jedoch zuvor ein besseres intuitives Verständnis erarbeiten, warum diese Änderung so gravierend ist.

Die Berechnungsvariante NP-vollständiger Probleme ist genau so schwer wie deren Entscheidungsvariante – TFNP-Berechungen könnten hingegen durchaus weniger komplex sein, als die Eintscheidungsvariante NP-vollständiger Probleme. Warum sich die Berechnung von Lösungen für ein TFNP-Problem möglicherweise fundamental von der für Lösungen eines NP-vollständigen Problemes unterscheidet wollen wir im Folgenden veranschaulichen.

Versetzen wir uns dazu zunächst in das Szenario, aus einem Algorithmus für eine beliebige Entscheidungsfrage einen für die Berechnungsfrage zu gewinnen. Die bloße Existenz einer Lösung in einem gegebenen Suchbereich wäre also überprüfbar, und wir möchten damit nun auch eine konkrete Lösung finden. Dazu könnten wir binäre Suche einsetzen, d.h. sukzessive Halbierung des Suchbereichs. Wir haben dieses Vorgehen in Algorithmus 1 grob skizziert.

Algorithm 1 Binäre Suche (Skizze)

```
1: while Suchbereich enthält mehr als ein Wort do
2:     if Erste Hälfte enthält Lösung then
3:         Suchbereich = erste Hälfte
4:     else
5:         Suchbereich = zweite Hälfte
6:     end if
7: end while
```

Dadurch halbiert sich der Suchbereich in jedem Schritt, weshalb er schon nach wenigen Schritten nur noch aus einem einzigen Element besteht. Auf diesem Weg könnte zu gegebenem y also immer auch ein passendes x gefunden werden, sodass $(x, y) \in R$ – also wäre auch jedes TFNP-Suchproblem effizient lösbar. Allerdings leistet die beschriebene Berechnungsvorschrift etwas (möglicherweise) stärkeres, könnte man schließlich nicht nur Urbilder finden, sondern sogar solche Fälle erkennen, in denen gar kein Urbild existiert. Tatsächlich ist diese Möglichkeit dabei sogar essenziell: falls in der ersten Hälfte keine Lösung existiert, muss in der zweiten Hälfte weitergesucht werden. Um zu dieser Erkenntnis zu gelangen, *muss* die Gewissheit bestehen, dass die erste Hälfte keine Lösung enthält. Daher muss es bei der binären Suche möglich sein, einen Beweis für die Nichtexistenz einer Lösung in einem gegebenen Bereich zu erhalten.

Hingegen ist dieser Fall bei TFNP prinzipiell ausgeschlossen – es ist *nie* möglich, dass gar keine Lösung existiert. Insbesondere kann ein potenzieller Suchalgorithmus für ein TFNP Problem möglicherweise auch nicht rekursiv auf die beiden Hälften des Suchbereiches angewandt werden, da eine der beiden Hälften im Allgemeinen keine Lösung enthalten wird. Die Berechnung eines solchen Suchproblemes könnte also durchaus auf gänzlich andere Weise erfolgen, als die oben beschriebene binäre Suche – vielleicht ist dieses Vorgehen dazu ja gar nicht nötig.

Hierin verbirgt sich der Schlüssel zur Sonderstellung der Klasse TFNP: Ein potenzieller Algorithmus, der ein TFNP-Suchproblem löst, muss möglicherweise *nie-*

mals einen Beweis dafür erbringen, dass gar keine Lösung existiert. Der Suchalgorithmus könnte also in der Lage sein, Lösungen zu finden, *ohne* es erkennen zu können, wenn mal keine Lösung existieren sollte. Wie bereits in Abschnitt 1.1.1 angedeutet stellt häufig genau das Erkennen der *Nichtexistenz* von Lösungen die eigentliche Schwierigkeit dar. Denn die Existenz einer Lösung kann zumindest schnell bewiesen werden – spätestens durch das explizite Angeben einer Lösung. Jedoch ist es eine offene Frage, ob es immer auch kurze Beweise gibt, die die Nichtexistenz belegen, oder ob solch eine Gewissheit im Allgemeinen nur durch erschöpfende (und schließlich erfolglose) Suche erlangt werden kann. Es ist also nicht unmittelbar klar, ob das Finden von Lösungen, *wenn* bereits fest steht, dass welche existieren, auch genau so schwer ist, wie erstmal überhaupt auf die Existenz von Lösungen zu prüfen.

Es stellt sich daher die Frage, ob P = NP wirklich notwendig wäre, um alle TFNP Instanzen effizient zu lösen. Darauf ist bisher keine Antwort bekannt. Fenner, Fortnow, Naik und Rogers konnten immerhin zeigen, dass (unter Anderem) $P = NP \cap coNP$ zur effizienten Lösbarkeit beliebiger TFNP Suchprobleme notwendig ist [FFNR]. Eine eng verwandte, doch möglicherweise stärkere Formulierung wäre die ebenfalls offene Frage, ob manche TFNP-Instanzen NP-hart sind.[9] Diese möchten wir nun genauer betrachten.

Versuche, die NP-Härte zu zeigen, scheitern häufig daran, dass nicht klar ist, wie ein TFNP Problem helfen können soll, wenn das NP Problem gar keine Lösung besitzt. Betrachten wir dafür z.B. das kanonische NP-vollständige Problem, zu entscheiden, ob eine gegebene NP-Maschine M eine Eingabe x innerhalb einer gewissen Schrittzahl akzeptiert. Ein Algorithmus zur Lösung eines potenziell NP harten TFNP-Suchproblemes T mag zwar in Abhängigkeit der Maschine M und der Eingabe x einen akzeptierenden Rechenweg von $M(x)$ finden können, *falls* einer existiert. Doch ist nicht unmittelbar klar, wie dessen Ein- und Ausgabe konkret aussehen könnte. Denn im Allgemeinen kann nicht einfach die Maschinennummer von M und das Wort x entgegengenommen werden, denn alle möglichen Eingaben *müssen* eine Lösung haben. Es gibt jedoch in einer normalen Gödelisierung gewiss Maschinen M', die eine Eingabe x' *nicht* akzeptieren, weshalb auch kein akzeptierender Rechenweg existiert, also auch keine Lösung. Damit würde T nicht mehr zu TFNP gehören, denn hier ist die Existenz einer Lösung vorgeschrieben.

Daher lässt sich ein NP-hartes TFNP Problem nicht auf dem Weg wie bei den gewohnten kanonischen vollständigen Problemen umsetzen. Entweder muss die Definition dahingehend geändert werden, dass nicht mehr nur akzeptierende Rechenwege zurückgegeben werden, oder das Eingabeformat müsste geändert werden. Betrachten wir zunächst den Fall, dass die möglichen Lösungen geändert werden, damit auch nichtakzeptierende Rechnungen eine Lösung besitzen. Könnte das TFNP Problem genau unterscheiden, welche Rechnungen nicht akzeptieren, für welche es daher eine „falsche“ Lösung zurückgeben muss, so wäre bereits NP = coNP. Kann es das hingegen nicht unterscheiden, so enthalten auch akzeptierende Rechnungen „falsche“

[9] Wobei man hier zurecht die Frage erheben sollte, entsprechend welcher Reduktion die NP-Härte überhaupt definiert wäre – denn schließlich handelt es sich bei TFNP nicht um Entscheidungs-, sondern um Berechnungsprobleme. Wir fassen die NP-Härte eines TFNP Problems hier im folgenden Sinne auf: jedes NP-Entscheidungsproblem ist effizient lösbar, solange bei der Berechnung eine einzige Lösung des NP-harten TFNP-Suchproblemes berechnet werden darf.

Lösungen. Wie könnte T hier genutzt werden, ohne dass der aufrufende Algorithmus selbst die Arbeit leisten muss, zu erkennen, ob eine Lösung existiert? Gibt es einen Hinweis, den T bei einer nichtakzeptierenden Maschine liefern könnte, sodass wir die Nichtakzeptanz von M schnell erkennen können? Dies ist die Frage, ob es auch kurze Beweise für die Nichtexistenz von Lösungen gibt – also der Frage, ob NP = coNP. Zwar haben wir ihn hier nur heuristisch begründet, doch tatsächlich besteht ein tiefgreifender Zusammenhang, welcher in Abschnitt 2.2 präzise untersucht wird.

Betrachten wir aber zuvor die andere Möglichkeit, das kanonische vollständige Problem zu modifizieren: weiterhin nur akzeptierende Rechenwege zurückzugeben, und stattdessen das Eingabeformat abzuändern, sodass nur akzeptierende Rechnungen eingegeben werden können. Es kann also im Allgemeinen nicht mehr einfach die Rechnung (M, x) übergeben werden, bei der M einer leicht berechenbaren Nummerierung *aller* NP Maschinen entspringt, zumindest nicht, wenn M auf allen möglichen Eingaben x überprüfbar sein soll. Stattdessen müsste z.B. der übergebene Index der Rechnung (M, x) aus einer solchen Nummerierung stammen, in der von vornherein nur akzeptierende Rechnungen aufgelistet werden.[10]

Solch eine Nummerierung zu definieren ist jedoch nicht trivial, und die Nummer einer spezifischen Rechnung effizient zu ermitteln wird sich ebenfalls als schwierig erweisen – denn falls $M(x)$ nicht akzeptiert, gibt es keinen solchen Index, was erkannt werden muss. Entweder ist der berechnete Index nicht mehr verlässlich, oder die Berechnung würde mindestens so schwer sein, wie die Nichtakzeptanz beliebiger NP-Maschinen zu beweisen. Allerdings ist dieser Ansatz noch aus einem weiteren Grund problematisch.

Falls M tatsächlich auf x akzeptiert, so würde ein Index innerhalb solch einer Nummerierung immerhin existieren. Doch sobald solch ein Index berechnet wäre, stünde die Akzeptanz ja bereits fest – der Index der Maschine in einer Nummerierung aller akzeptierenden Maschinen fungiert als ein *Beweis* dafür, dass die Maschine akzeptiert. Die Frage nach der Akzeptanz wäre also schon selbst beantwortet, und das TFNP-Problem T wäre nun unnötig. Der einzige durch T erreichbare Nutzen wäre nur noch der, auch eine konkrete Lösung zu erhalten.

Der letzte im Beispiel behandelte Fall warf also die Frage auf, ob das Berechnen der konkreten Lösung tatsächlich noch einen Mehrwert darstellen würde – denn es ist nicht klar, wie ein Beweis der Existenz einer Lösung ablaufen könnte, ohne dabei zwangsläufig auch eine konkrete Lösung zu beinhalten. Gibt es überhaupt Methoden, die Existenz fundamental anders zu beweisen, als konstruktiv vorzugehen? Dies führt uns zum Begriff p-optimaler Beweissysteme, deren Untersuchung sich mit einer Verallgemeinerung dieser Frage befasst: statt sich auf den konstruktiven Beweis der Existenz via konkreter Lösung zu beschränken, wird hier stattdessen gefragt, ob für eine *beliebige* Beweismethode immer auch ein Argument existiert, das keinerlei Rückschlüsse auf ein Argument der ursprünglichen Methode erlaubt.

[10] Dass die eingegebenen Maschinen M auf jedem x akzeptieren wäre nicht unbedingt notwendig, solange jede mögliche eingegebene Rechung (M, x) akzeptiert – es könnte sich also auch um eine Nummerierung aller akzeptierender Rechnungen handeln, um die Totalität zu wahren.

1.2.2 Optimale Beweissysteme

Um die Definition eines Beweissystemes zu motivieren erinnern wir an eine Formulierung aus dem vorherigen Absatz. Darin behaupteten wir, der Index einer Maschine in einer Nummerierung aller Maschinen mit einer gewissen Eigenschaft X wäre ein *Beweis* dafür, dass die Maschine die Eigenschaft X habe. Dabei ist hier unerheblich, welche Eigenschaft gemeint ist. Es sei f die Funktion, die einer gegebenen Nummer die entsprechende Maschine zuordnet. Es ist also

$$f(\mathbb{N}) = \{\text{Maschinen mit Eigenschaft } X\} =: L_X$$

Die Funktion f gibt also genau die Maschinen mit Eigenschaft X zurück. Erhalten wir für irgendein $x \in \mathbb{N}$ durch Auswertung der Funktion die Maschine $f(x) = y$, so wissen wir daraus, dass y die Eigenschaft X hat, und x ist ein Beweis dafür, der sich durch f verifizieren ließe. Damit das Verifizieren auch tatsächlich möglich ist, fordern wir nun noch, dass f effizient berechenbar sei. Erfüllt es das, fassen wir f als ein Beweissystem für die Menge L_X auf.

Ein Beweissystem ist also eine Funktion $f \in \mathrm{FP}$, die einen Beweis x in polynomieller Zeit zu $y := f(x)$ auswertet, und dadurch die zunächst triviale Aussage „$y \in f(\mathbb{N})$“ beweist. Erst in Kombination mit der Kenntnis, dass f *nur* Objekte mit der Eigenschaft X zurückgibt, also dass $f(\mathbb{N}) \subseteq L_X$, entsteht ein Beweis für „y hat Eigenschaft X“.

Man beachte, dass zwar f in polynomieller Laufzeit im Verhältnis zur Eingabe (dem Beweis) arbeiten muss, der Beweis selbst jedoch trotzdem unverhältnismäßig lang sein könnte. Deswegen ist man besonders an möglichst effektiven Beweissystemen interessiert: solchen, in denen möglichst kurze Beweise existieren, oder aber auch solche, die „universelle“ Argumente verwenden. Damit sind Argumente gemeint, die mit nur geringfügiger Modifikation in *jedem* Beweissystem funktionieren.

Besonders interessant sind dabei Beweissysteme für die Klassen NP und coNP, denn NP ist explizit durch die Existenz leicht verifizierbarer Beweise für eine Aussage charakterisiert, womit coNP durch die Nichtexistenz leicht verifizierbarer Beweise charakterisiert ist. Dadurch lassen sich zwei wesentliche Bausteine der Logik formulieren, der Existenzquantor und der Allquantor. Daher ermöglichen sie es, die Grundbausteine quantifizierter Aussagenlogik komplexitätstheoretisch darzustellen.

Bei der Betrachtung genügt es, sich auf die jeweiligen vollständigen Probleme SAT und TAUT zu beschränken. Während SAT diejenigen aussagenlogischen Formeln beschreibt, die erfüllbar sind, beschreibt TAUT diejenigen Formeln, die immer erfüllt sind. In der Menge aller aussagenlogischen Formeln ist TAUT also sinngemäß komplementär zu SAT – ist eine Formel nicht erfüllbar, ist ihre Negation immer erfüllt, und umgekehrt.

Beide Probleme haben Beweissysteme, auch wenn im Fall von TAUT bisher nur solche bekannt sind, die sich auf erschöpfende Suche berufen, weshalb dessen Beweise im Allgemeinen exponenziell lang im Vergleich zur bewiesenen Formel sind. Im Fall von SAT ist die Suche nach einem effizienten Beweissystem hingegen erfolgreich: Will man die Erfüllbarkeit einer Formel beweisen – also die Existenz einer erfüllenden Variablenbelegung – liegt es nahe, einfach direkt eine solche Variablenbelegung anzugeben. Dadurch lässt sich ein Beweissystem formulieren, das alle erfüllbaren Formeln beweist, und bei dem Beweise jeweils die erfüllende Belegung beinhalten. In-

teressanter ist jedoch die Frage, ob es überhaupt möglich ist, die Erfüllbarkeit zu beweisen, *ohne* dabei über eine konkrete erfüllende Belegung zu argumentieren. Die etwas stärkere Annahme, dass sogar zu jeder beliebigen Beweismethode fundamental andere Argumente möglich seien, verbirgt sich hinter der im Fokus des nächsten Abschnittes stehenden Vermutung SAT.

1.2.3 Die Vermutungen

SAT

Um die eben gestellte Frage nach universellen Beweisargumenten formal sauber stellen zu können, führen wir den Begriff eines p-optimalen Beweissystemes ein. Ein p-optimales Beweissystem f ist eines, das die Kernaussagen aller Beweise aller anderen Beweissysteme enthält. Aus jedem Beweis eines anderen Beweissystemes kann mit geringem Aufwand ein Beweis in f ermittelt werden. Das lässt sich so interpretieren, als dass sie immer ein „universelles“ Argument verwenden, das im Kern auch in *jedem* anderen Beweis zur selben Aussage verwendet wird. Es wäre also nicht möglich, Aussagen auf fundamental anderem Wege zu beweisen. Im Fall des Erfüllbarkeitsproblemes SAT entspräche das der Vermutung, dass zum Beispiel *alle* Beweise für die Erfüllbarkeit auch implizit eine konkrete erfüllende Belegung erkennen lassen. Es ist nicht bekannt, ob das der Fall ist. Die Vermutung, dass es durchaus möglich sei, auch signifikant andere Beweise zu liefern, dass es also kein p-optimales Beweissystem für SAT gebe, ist unter dem Namen SAT bekannt.

TFNP

Für ein besseres Verständnis einer Komplexitätsklasse hilft es oft, dessen vollständige Probleme zu betrachten. Vollständige Probleme sind in gewisser Weise die schwersten Probleme einer Klasse, da sie die Lösung jedes anderen Problemes der Klasse ermöglichen. Formal bedeutet dies, dass sich jedes andere Problem der Klasse auf das vollständige Problem reduzieren lässt. Auch für TFNP lässt sich definieren, wie ein Suchproblem A auf ein anderes Suchproblem B reduziert wird. Die genaue Definition findet sich in 2.11 – an dieser Stelle genügt die Betrachtung, dass A leicht lösbar wäre wenn schon ein effizienter Algorithmus zur Lösung von B zur Verfügung stünde.

Versucht man damit jedoch auch für TFNP ein vollständiges Problem zu finden, stößt man auf Schwierigkeiten. Bis heute ist nicht bekannt, ob TFNP vollständige Probleme besitzt. Die Vermutung, dass es keine gibt, kürzt man mit dem Namen TFNP ab. Wie bei SAT und SAT benutzen wir unterschiedliche Schrift, um zwischen der Klasse TFNP und der Vermutung TFNP zu differenzieren. Betrachten wir nun, warum die Existenz vollständiger Probleme für TFNP als unwahrscheinlich gilt.

In Abschnitt 1.2.1 stellten wir uns nur die Frage nach der NP-Härte. Diese erwies sich als schwierig nachzuweisen, weil TFNP zumindest nicht auf trivialem Wege für erfolglose Suchen eingesetzt werden kann. Die NP-Härte wäre zwar hinreichend für die Existenz eines TFNP-vollständigen Problemes, doch ist sie möglicherweise nicht notwendig. Untersucht man daher stattdessen die Existenz TFNP-vollständiger Probleme, offenbart sich tatsächlich noch eine weitere Herausforderung: ein TFNP-

vollständiges Problem müsste für seine Suche solche Lösungen liefern, die gleichzeitig auch Lösungen für *jede* andere Frage ermöglichen. Akzeptierende Rechenwege einer immer akzeptierenden nichtdeterministischen Maschine bei einer Eingabe y müssten dann immer auch für jede andere immer akzeptierende Maschine in akzeptierende Rechenwege bei Eingabe y übersetzt werden können. Entweder ist die Suche nach akzeptierenden Rechenwegen trivial, oder alle akzeptierenden Pfade stehen in einem nichttrivialen Zusammenhang – es gäbe einen „universellen“ Rechenweg, der implizit bei jeder der Maschinen erfolgreich ist.

Diese Formulierung wurde bewusst mit der Intention gewählt, die Analogie zu p-optimalen Beweissystemen hervorzuheben. Denn auch bei näherer Betrachtung erscheint es, als würden beide Vermutungen die selbe Aussage beschreiben. Bisher war nicht bekannt, ob es sich in jedem relativierten Kontext um die selben Aussagen handelt. Doch tatsächlich handelt es sich um unterschiedlich relativierende Aussagen, was wir in *Kapitel* 3 beweisen. Werfen wir daher einen genaueren Blick auf den Zusammenhang der beiden, auch im Kontext weiterer verwandter Vermutungen.

Zusammenhänge

Pudlák stellte neben SAT und TFNP noch eine Reihe weiterer Vermutungen miteinander in Zusammenhang. Sie alle formal einzuführen würde den Rahmen dieser Arbeit sprengen, doch wollen wir hier trotzdem kurz auf sie eingehen, um eine breitere Perspektive auf die im Fokus dieser Arbeit stehenden Vermutungen zu erlangen. Besonders interessant ist hierbei die Vermutung CON, denn hier handelt es sich um die Vermutung, dass es keine p-optimalen Beweissysteme für TAUT gebe. Sie stellt also in gewisser Weise das Gegenstück zu SAT dar: während es bei SAT um Beweissysteme für Existenzquantor-Aussagen geht, bezieht sich CON auf Allquantor-Aussagen. Wir betrachten die folgenden Vermutungen:

TFNP	Es existiert kein TFNP-vollständiges Problem.
SAT	Es existiert kein p-optimales Beweissystem für SAT.
CON	Es existiert kein p-optimales Beweissystem für TAUT.
$\mathsf{CON}^{\mathsf{N}}$	Es existiert kein optimales Beweissystem für TAUT.[11]
DisjNP	Es gibt kein vollständiges disjunktes NP-Paar.
DisjCoNP	Es gibt kein vollständiges disjunktes coNP-Paar.
UP	Es gibt kein UP-vollständiges Problem.[12]

[11] Ein optimales Beweissystem ist ähnlich wie ein p-optimales Beweissystem, nur muss die übersetzende Funktion nicht effizient berechenbar sein. Eine genaue Definition ist im nächsten Kapitel.

[12] Die Klasse UP beinhaltet diejenigen Sprachen, die von einer NP-Maschine akzeptiert werden, die bei jeder Eingabe auf höchstens einem Rechenweg akzeptiert. Die Frage nach vollständigen Problemen bezieht sich auf $\leq_m^p$-Reduktion.

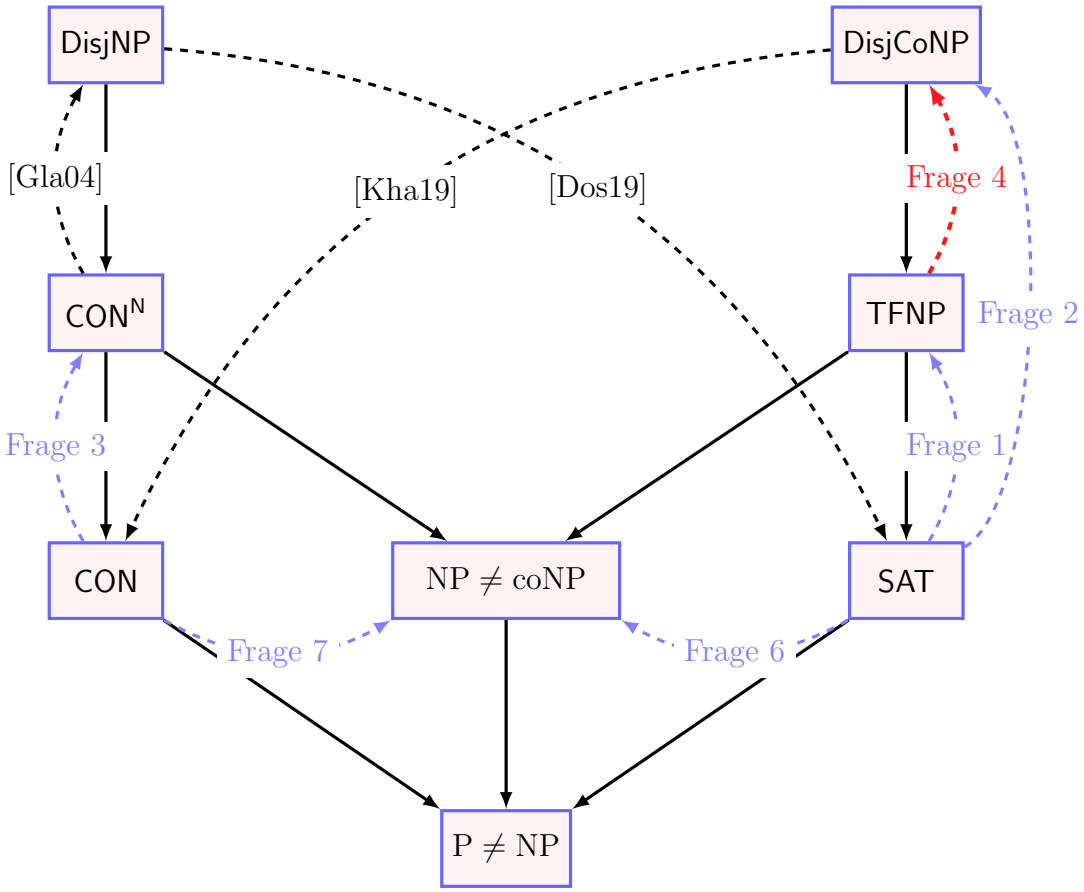


Abbildung 1.1: Beziehungen zwischen den Vermutungen, inklusive der bisher offenen Fragen. Durchgezogene Pfeile bedeuten relativierbare Implikationen. Ein gestrichelter Pfeil von A nach B bedeutet, dass es ein Orakel O gibt, relativ zu dem $A \wedge \neg B$ gilt. Alle blau markierten Pfeile stellen bisher unbekannte Separationen dar, die wir in Abschnitt 4.1 mit einem einzigen Orakel beweisen. Nur der rot markierte Pfeil bleibt eine offene Frage. Implizit aus anderen eingezeichneten Resultaten folgende Separationen wurden hier weggelassen, finden sich aber in Abb. 4.2.

Es konnten bereits einige relativierbare Implikationen zwischen diesen Aussagen gezeigt werden. Für die Beweise sei hier auf Pudlák's Arbeit [Pud17] verwiesen. Abbildung 1.1 gibt einen Überblick der bekannten Implikationen, sowie über bekannte Orakelseparationen.

Frühere Arbeiten, die sich mit den Beziehungen dieser Vermutungen befassen, zeichnen oftmals ein ähnliches Diagramm, an dem sich auch unseres orientiert hat. In den bisherigen Untersuchungen war NP $\neq$ coNP jedoch selten von besonderem Interesse. Für jede der folgenden Implikationen $A \Longrightarrow B$ war es bisher eine offene

Frage, ob ein Orakel mit $A^O \wedge \neg B^O$ existiert:

$$\begin{aligned} \mathsf{SAT} &\implies \mathsf{TFNP} && \text{(Frage 1)} \\ \mathsf{SAT} &\implies \mathsf{DisjCoNP} && \text{(Frage 2)} \\ \mathsf{CON} &\implies \mathsf{CON}^{\mathsf{N}} && \text{(Frage 3)} \\ \mathsf{TFNP} &\implies \mathsf{DisjCoNP} && \text{(Frage 4)} \\ \mathsf{UP} &\implies \mathsf{DisjNP} && \text{(Frage 5)} \end{aligned}$$

Währenddessen war die Relativierbarkeit der folgenden Implikationen zwar ebenfalls unbekannt, wurde jedoch auch wenig beachtet.

$$\begin{aligned} \mathsf{SAT} &\implies \mathrm{NP} \neq \mathrm{coNP} && \text{(Frage 6)} \\ \mathsf{CON} &\implies \mathrm{NP} \neq \mathrm{coNP} && \text{(Frage 7)} \end{aligned}$$

Doch tatsächlich werden wir durch Beantwortung von Frage 6 auch fast alle anderen Fragen lösen. Beispielsweise wird unser Satz 3.12 durch Separation der Vermutungen SAT und NP $\neq$ coNP insbesondere auch SAT und TFNP separieren (also Frage 1 beantworten). Wir schlagen deshalb vor, die Vermutung NP $\neq$ coNP ebenfalls in zukünftige Diagramme zu inkludieren, wodurch eine in Abb. 1.1 illustrierte Symmetrie erkennbar wird. Und tatsächlich lassen sich unsere für die rechte Seite des Diagrammes erhaltenen Resultate buchstäblich auf die linke Seite spiegeln, denn gilt für ein Orakel O $\mathrm{NP}^O = \mathrm{coNP}^O$, so gilt auch $\mathsf{SAT}^O \iff \mathsf{CON}^O$, wodurch dann insbesondere auch Frage 7 beantwortet ist. Insgesamt können wir fünf der sieben genannten offenen Fragen in dieser Arbeit lösen, lediglich Frage 4 und Frage 5 bleiben offen. Abb. 4.2 gibt einen Gesamtüberblick aller nun bekannten Separationen. Die Vermutung UP gehört ebenfalls zu den von Pudlák betrachteten Vermutungen, aus Gründen der Übersichtlichkeit wurde sie jedoch aus den Diagrammen dieser Arbeit weggelassen. Um die Relevanz von NP $\neq$ coNP weiter zu untermauern werden wir im nächsten Abschnitt (nach einer Reihe vorbereitender Definitionen) den Zusammenhang von NP $\neq$ coNP zur Klasse TFNP genauer beleuchten.

Kapitel 2

Vorbereitung

2.1 Definitionen

Grundlagen

Wir bezeichnen mit $\mathbb{N}$ die natürlichen Zahlen *inklusive* der 0, und mit $\mathbb{N}^+$ die positiven natürlichen Zahlen. Es bezeichne π die Cantorsche Paarungsfunktion, und $\mathbb{N}[X]$ bezeichne die Polynome mit natürlichzahligen Koeffizienten. Für eine Menge A bezeichne $\mathbb{P}\big(A\big)$ die Potenzmenge von A. Alle Sprachen und Maschinen dieser Arbeit sind über dem Alphabet $\Sigma = \{0, 1\}$ definiert. Unsere Definition von Polynomialzeitberechenbarkeit setzt immer Totalität voraus. Σ^* bezeichne die Menge aller Wörter endlicher Länge, und für $w \in \Sigma$ sei $|w|$ die Länge von w. Für $k \in \mathbb{N}$ bezeichnen $\Sigma^k := \{x \in \Sigma^* \mid |x| = k\}$ und $\Sigma^{\leq k} := \{x \in \Sigma^* \mid |x| \leq k\}$.
Für $x \in \Sigma^*$ bezeichne $x_1, \ldots, x_{|x|}$ die Zeichen des Wortes x, und sind umgekehrt $x_1, \ldots, x_n$ Zeichen, so bezeichne x das aus Ihnen zusammengesetzte Wort.
Es bezeichne $bin : \mathbb{N} \to \Sigma^*$ eine bijektive Kodierung, sodass sowohl bin als auch bin^{-1} in Polynomialzeit berechenbar sind. Es bezeichne $\leq_L$ bzw $<_L$ die lexikographische Ordnung von Wörtern über Σ^*. Für Wörter $w_1, \ldots, w_n$ bezeichnen wir mit $\langle w_1, \ldots, w_n \rangle$ eine bijektive Kodierung des n-Tupels als einzelnes Wort. $\langle \cdot \rangle$ bezeichnet also für jede Stelligkeit n eine (andere) Bijektion, die jeweils n Eingabewörter in polynomieller Zeit zu einem einzigen Wort kodiert, und auch in polynomieller Zeit invertierbar ist (sofern die Stelligkeit bekannt ist, damit die richtige Bijektion invertiert wird). Dies kann beispielsweise durch wiederholte Verwendung der Cantorschen Paarungsfunktion erfolgen.
Wenn wir n Parameter eines $n + k$-stelligen Ausdruckes entsprechend der n Zeichen eines Wortes $w \in \Sigma^n$ belegen, schreiben wir kurz „$f(w, x_{n+1}, \ldots, x_{n+k})$“ statt „$f(w_1, \ldots, w_n, x_{n+1}, \ldots, x_{n+k})$“ (für beliebige $n \in \mathbb{N}^+, k \in \mathbb{N}$).

Ist M eine beliebige deterministische Maschine mit Ausgabe, so bezeichnen wir mit $M^B(x)$ den Wert der Berechnung von M auf Eingabe x unter Verwendung des Orakels B. Ist M hingegen eine nichtdeterministische Maschine bezeichnen wir damit stattdessen nur deren Entscheidungsverhalten, d.h. in diesem Fall ist $M^B(x) \in \{\texttt{true}, \texttt{false}\}$. Eine Polynomialzeit Orakelturingmaschine werden wir manchmal als „poly-OTM“ abkürzen. $\langle M \rangle \in \Sigma^*$ bezeichne eine Kodierung des Programmcodes der Maschine M, und $L(M^B)$ bezeichnet die von M unter Verwendung des Orakels B akzeptierte Sprache.

Definition 2.1 (Relativierte Klassen) Sei $B \subseteq \Sigma^*$ ein beliebiges Orakel.

$\mathrm{P}^B := \{L \mid \exists \text{ deterministische poly-OTM } M, \text{ sodass } L(M^B) = L\}$

$\mathrm{NP}^B := \{L \mid \exists \text{ nichtdeterministische poly-OTM } M, \text{ sodass } L(M^B) = L\}$

$\mathrm{FP}^B := \{f : \Sigma^* \to \Sigma^* \mid \exists \text{ det. poly-OTM } M \text{ sodass } \forall x \in \Sigma^* : \ M^B(x) = f(x)\}$

Relativierte Aussagenlogik

Wir erweitern nun die Definition von SAT, sodass es auch in relativierten Umgebungen NP-vollständig ist, indem zusätzlich zu den üblichen Konstrukten aussagenlogischer Formeln auch sogenannte Orakelgatter $B(x_1, \ldots, x_n)$ bereit stehen. Diese Gatter repräsentieren die Aussage „$x_1 \cdot \ldots \cdot x_n \in B$“, wobei die x_i sowohl freie als auch gebundene Variablen sein dürfen. Das zu einem Orakel B relativierte Erfüllbarkeitsproblem bezeichnen wir dann als SAT^B.

Definition 2.2 (Aussagenlogische Formeln)

i) $\Upsilon := \{\vee, \wedge, 1, 0, (,), \neg, B, x, _1, _0\}^*$ ist die Menge aller (ggf. syntaktisch inkorrekten) aussagenlogischen Formeln. Freie Variablen können durch „$x_{bin(1)}, x_{bin(2)}, \ldots$“ dargestellt werden. Wir schreiben dafür jedoch „$x_1, x_2, \ldots$“.

ii) $\tau : \Sigma^* \to \Upsilon$ sei eine bijektive Kodierung mit $\tau, \tau^{-1} \in \mathrm{FP}$.

iii) $\mathsf{eval} : \Upsilon \to \{\mathtt{true}, \mathtt{false}\}$ sei die Funktion, die den Wahrheitswert einer Formel zurückgibt. Falls w freie Variablen beinhaltet oder w syntaktisch falsch ist, so ist der Rückgabewert `false`. Für ein Wort $w \in \Sigma^*$ wird die Formel „$B(w_1, \ldots, w_n)$“ entsprechend der Aussage $w \in B$ ausgewertet, während für $w \notin \Sigma^*$ bei der Auswertung alle $w_i \notin \Sigma$ übersprungen werden.

iv) Für $w \in \Upsilon$ bezeichnet $w : \Sigma^* \to \Upsilon$ die Funktion, die freie Variablen in w entsprechend der eingegebenen Bits belegt. Überschüssige Bits werden dabei ignoriert, fehlende mit 0 belegt.

Korollar 2.3 Für jedes $B \subseteq \Sigma^*$ ist $\mathsf{eval} \in \mathrm{FP}^B$.

Definition 2.4 (SAT^B, TAUT^B)
Sei $B \subseteq \Sigma^*$ ein beliebiges Orakel. Dann definieren wir

$$\mathrm{SAT}^B := \left\{w \in \Sigma^* \mid \exists x \in \Sigma^{\leq |w|} : \ \mathsf{eval}(\tau(w)(x)) = \mathtt{true}\right\} \in \mathrm{NP}^B$$

$$\mathrm{TAUT}^B := \left\{w \in \Sigma^* \mid \forall x \in \Sigma^{\leq |w|} : \ \mathsf{eval}(\tau(w)(x)) = \mathtt{true}\right\} \in \mathrm{coNP}^B$$

Korollar 2.5 Sei $B \subseteq \Sigma^*$ beliebig. Dann gilt

i) $\tau^{-1}\big(B(x_1, \ldots, x_n)\big) \in \mathrm{SAT}^B \iff \exists x \in \Sigma^n : x \in B$

ii) SAT^B ist NP^B-vollständig

iii) $\overline{\mathrm{SAT}^B}$ und TAUT^B sind coNP^B-vollständig

Definition 2.6 (Aufzählung der $\mathrm{FP}^{|\cdot|}$ Maschinen)
Sei $(f_i)_{i\in\mathbb{N}}$ eine Aufzählung von über dem Alphabet Σ arbeitenden poly-OTMs mit Ausgabe, sodass für jedes Orakel $B \subseteq \Sigma^*$ gilt

i) $i \mapsto \langle f_i \rangle \ \in \mathrm{FP}$

ii) $\langle f_i \rangle \mapsto i \ \in \mathrm{FP}$

iii) $\forall f : \Sigma^* \to \Sigma^* \in \mathrm{FP}^B \ \exists i \in \mathbb{N} \ \forall x \in \Sigma^* : \ f(x) = f_i^B(x)$, und

iv) $\forall i \in \mathbb{N} \forall x \in \Sigma^*$: die Berechnung von $f_i^B(x)$ genügt der Zeitschranke $|x|^i + i$. Wir notieren das Polynom $|x|^i + i$ kurz als $t_i(x)$.

Wir werden aus Gründen der Übersichtlichkeit gegebenenfalls eine Aufzählung verwenden, in der $f_i : \mathbb{N} \to \mathbb{N}$ statt $f_i : \Sigma^* \to \Sigma^*$. In diesen Fällen ist damit streng genommen $bin^{-1} \circ f_i \circ bin$ gemeint.

Definition 2.7 (Aufzählung der $\mathrm{NP}^{|\cdot|}$ Maschinen)
Sei $(M_i)_{i\in\mathbb{N}}$ eine Aufzählung von nichtdeterministischen poly-OTMs über dem Alphabet Σ, sodass für jedes Orakel $B \subseteq \Sigma^*$ gilt

i) $i \mapsto \langle M_i \rangle \ \in \mathrm{FP}$

ii) $\langle M_i \rangle \mapsto i \ \in \mathrm{FP}$

iii) $\forall L \in \mathrm{NP}^B \ \exists i \in \mathbb{N} \ L(M_i^B) = L$, und

iv) $\forall i \in \mathbb{N} \forall x \in \Sigma^*$: die Berechnung von $M_i^B(x)$ genügt der Zeitschranke $|x|^i + i$.

Definition 2.8 (Beweissystem)
Für ein Orakel $B \subseteq \Sigma^*$ ist ein Beweissystem für eine Menge $L \subseteq \Sigma^*$ eine Funktion $f \in \mathrm{FP}^B$ mit $f(\Sigma^*) = L$. Wir schreiben abgekürzt „f ist ein L-Beweissystem“.

Definition 2.9 (p-optimales Beweissystem)
Sei $B \subseteq \Sigma^*$ ein beliebiges Orakel. Ein p-optimales Beweissystem für eine Menge $L \subseteq \Sigma^*$ ist ein L-Beweissystem g, sodass für jedes andere L-Beweissystem h ein $f \in \mathrm{FP}^B$ existiert, sodass $g \circ f = h$. In diesem Fall wird h von g „p-simuliert“.

Definition 2.10 (TFNP^B) Sei $B \subseteq \Sigma^*$ ein beliebiges Orakel.

$$\mathrm{TFNP}^B := \left\{ (R,p) \in \left(\mathbb{P}\left(\Sigma^* \times \Sigma^*\right) \cap \mathrm{P}^B\right) \times \mathbb{N}[\mathrm{X}] \;\middle|\; \forall y \in \Sigma^* \ \exists x \in \Sigma^{\leq p(|y|)} : (x,y) \in R \right\}$$

Definition 2.11 (Reduktion für TFNP^B)
Sei $B \subseteq \Sigma^*$ ein beliebiges Orakel, und seien (R,p) und $(T,q) \in \mathrm{TFNP}^B$.
Dann ist $(T,q) \leq_{\mathrm{TFNP}} (R,p) \ :\Longleftrightarrow \exists f,g \in \mathrm{FP}^B$, sodass

$$\forall y \in \Sigma^* : \forall x \in \Sigma^{\leq p(|y|)} : \big(x, f(y)\big) \in R \Longrightarrow \big(g(x,y), y\big) \in T$$

Bemerkung In der Definition von $(T,q) \leq_{\mathrm{TFNP}} (R,p)$ dient f als Vorbereitung der an R gestellten Frage. Besitzt R zur Frage $f(y)$ eine Lösung x, berechnet g mit dieser Kenntnis eine Lösung für T zur Frage y, d.h. T lässt sich durch R lösen.

Definition 2.12 (TFNP^B-Vollständigkeit) Sei $B \subseteq \Sigma^*$ ein beliebiges Orakel.
Sei $(R,p) \in (\mathbb{P}\big(\Sigma^* \times \Sigma^*\big) \cap \mathrm{P}^B) \times \mathbb{N}[X]$. Dann ist (R,p) TFNP^B-vollständig $:\Longleftrightarrow$

$$(R,p) \in \mathrm{TFNP}^B \quad \wedge \quad \forall (T,q) \in \mathrm{TFNP}^B : (T,q) \leq_{\mathrm{TFNP}} (R,p)$$

2.2 Untersuchungen zu TFNP

Zusammenhang mit NP = coNP

Im Einführungsabschnitt zu TFNP stellten wir die Frage, ob P = NP notwendig wäre, um alle TFNP Instanzen effizient lösen zu können. Tatsächlich ist sogar die (möglicherweise) schärfere Frage gerechtfertigt, ob dazu überhaupt NP = coNP notwendig wäre. Da ein TFNP Problem immer Lösungen besitzt, sind "falsche" Eingaben von vornherein ausgeschlossen. Würde ein TFNP Problem also z.B. akzeptierende Rechenwege von NP Maschinen berechnen, wie im Beispiel aus dem vorherigen Abschnitt, so dürfte das Eingabeformat gar keine Repräsentation für nichtakzeptierenden Maschinen *haben*. Um dieses TFNP Problem "nutzen" zu können (d.h., um die gewünschte Eingabe zu tätigen), müsste also zwangsläufig bereits solch eine Repräsentation bekannt sein. Anders formuliert: könnte ein Algorithmus für beliebige Maschinen die entsprechende Repräsentation im Eingabeformat für T ermitteln, so könnte er insbesondere erkennen, ob eine Lösung existiert oder nicht.

Dieser Gedanke lässt sich mit wenigen Schritten zur folgenden Aussage weiterführen:

Satz 2.13 Existiert ein NP-hartes TFNP Problem, so gilt NP = coNP.

Beweis. Es sei $T = (R, p)$ ein NP-hartes TFNP Problem, d.h. $\forall L \in \text{NP}$ existiert ein Algorithmus, der L in Polynomialzeit entscheidet, und dabei für jede Eingabe höchstens eine[1] Lösung von T erfragen darf. Dem Algorithmus wird also für ein polynomiell langes $y \in \Sigma^*$ seiner Wahl ein $x \in \Sigma^{\leq p(|y|)}$ mit $(x, y) \in R$ gestellt.

Sei $\overline{A} \in \text{coNP}$, also $A \in \text{NP}$. Dann existiert nach Annahme ein Algorithmus r, der A in deterministischer Polynomialzeit entscheidet, unter Verwendung einer Frage an T. Wir zeigen nun, dass $\overline{A} \in \text{NP}$, indem wir einen nichtdeterministischen Polynomialzeitalgorithmus beschreiben, der $\overline{A}$ entscheidet. Dieser arbeitet bei Eingabe $n \in \Sigma^*$ wie folgt:

Simuliere r auf Eingabe n bis zu dem Schritt, an dem die Frage y an T gestellt worden wäre. D.h. es wird nach einem $x \in \Sigma^{\leq p(|y|)}$ gefragt, sodass $(x, y) \in R$. Da y polynomiell in n ist, kann unser Algorithmus hier stattdessen selbst durch nichtdeterministische Verzweigung nach einem solchen x suchen. Und da $R \in \text{P}$, kann r anschließend auch das gewählte x auf $(x, y) \overset{?}{\in} R$ überprüfen. Falls ein Rechenweg ein falsches x gewählt hat, so stoppe er ablehnend. Falls er ein korrektes x ermittelt hat, so führe er anschließend die Simulation von r mit dieser Antwort fort. Da immer ein x existiert, wird mindestens ein Rechenweg bis zu diesem Punkt kommen. Da r deterministisch ist, kann jeder einzelne solche Rechenweg nun entsprechend des Ergebnisses von r erkennen, ob $n \in A$. Falls $n \in A$, so lehne ab. Falls $n \notin A$, so akzeptiere.

Der beschriebene Algorithmus hat genau dann einen akzeptierenden Rechenweg, wenn $n \notin A$, also $n \in \overline{A}$. Damit entscheidet er $\overline{A}$, und da A beliebig war, ist $\text{coNP} \subseteq \text{NP}$.

[1] Tatsächlich ist es für diesen Beweis nicht relevant, ob der Algorithmus auch mehr Fragen stellen darf.

Insbesondere ist dann NP $\subseteq$ coNP: Sei $N \in$ NP, dann ist $\overline{N} \in$ coNP. Da coNP $\subseteq$ NP, folgt $\overline{N} \in$ NP, und somit $N \in$ coNP. Da N beliebig war, ist also auch NP $\subseteq$ coNP, und insgesamt NP = coNP. □

Wir haben hier die Folgen der Existenz von NP-harten TFNP Problemen untersucht. Nun möchten wir uns der Frage widmen, inwiefern diese mit der Existenz von TFNP-vollständigen Problemen zusammenhängt, um ein besseres Verständnis der Vermutung TFNP zu entwickeln. Dazu zeigen wir zunächst, dass NP = coNP hinreichend für die Existenz eines TFNP-vollständigen Problemes wäre.

Der Grund liegt darin, dass die von uns konstruierte TFNP-Instanz es nun schon selbst erkennen kann, wenn die gegebene Maschine überhaupt keinen akzeptierenden Rechenweg hat – und entsprechende Maßnahmen ergreifen kann, um trotzdem auch für diese Eingabe Lösungen zu besitzen. In diesem Fall kann sie nämlich problemlos akzeptieren, denn durch die Reduktionsfunktionen werden nur solche Eingaben getätigt, die auch tatsächlich akzeptierende Rechenwege haben. Dadurch kann ein kanonisches vollständiges Problem konstruiert werden, dessen Totalität tatsächlich sichergestellt ist, und mit dem wir den folgenden Satz erhalten. Wir gehen davon aus, dass dieses naheliegende Resultat bereits bekannt ist, konnten jedoch keine Quelle dazu finden.

Satz 2.14 NP = coNP $\implies \neg$TFNP

Aus Gründen der Lesbarkeit definieren wir einen Rechenweg einer nichtdeterministischen Maschine auf einer Eingabe im folgenden Beweis so, dass nur die nichtdeterministischen Verzweigungen protokolliert werden, nicht die danach folgenden deterministischen Berechnungen. Der Rechenweg protokolliert also genau die Bits des nichtdeterministisch erzeugten Wortes.

Beweis. Sei NP = coNP. Wir müssen zeigen, dass es ein TFNP-vollständiges Problem gibt. Betrachten wir dazu folgende NP Maschine T, die auf einer Eingabe $y = \langle i, x, w\rangle$ wie folgt arbeitet: Falls $|w| < |x|^i + i$, so akzeptiere. Ansonsten spalte sie sich nichtdeterministisch in zwei verschiedene Rechenwege auf: der erste Weg (d.h. entsprechende Rechenwege beginnen mit 0) dient dazu, anzunehmen, dass Maschine i die Eingabe x auf keinem Rechenweg der Länge $\leq |x|^i + i$ akzeptiert. Hier wird die Maschine sich also aufspalten und die möglichen polynomiell langen Beweise dafür erraten – solche existieren, weil NP = coNP.[2] Findet dabei ein Rechenweg einen Beweis, dass $x \notin L(M_i)$, so wird er akzeptieren, ansonsten nicht.

Die andere Hälfte (d.h. mit 1 beginnende Rechenwege) dient dazu, anzunehmen, dass die Maschine i die Eingabe x tatsächlich auf einem Rechenweg der Länge $\leq |x|^i + i$ akzeptieren wird. Hierzu rät sie nichtdeterministisch den Rechenweg, auf dem M_i akzeptieren könnte, simuliert M_i, und akzeptiert genau dann, wenn der simulierte Rechenweg akzeptiert hätte. Die Maschine T akzeptiert also genau dann auf einem Rechenweg der ersten Hälfte, wenn M_i die Eingabe x nicht akzeptiert.

[2] und daher das Komplement des kanonischen vollständigen NP Problemes $\{(i, x, w) \mid M_i(x)$ akzeptiert nach $\leq |w|^i + i$ Schritten$\}$ ebenfalls in NP liegt, dieses also genutzt werden kann, um die Nichtakzeptanz von M_i bei Eingabe x auf mindestens einem Rechenweg zu erkennen.

Falls M_i die Eingabe x auf einem Rechenweg akzeptiert, so wird dieser Rechenweg in der zweiten Hälfte gefunden, und T akzeptiert. Falls die Maschine nicht akzeptiert, so wird in der ersten Hälfte ein Beweis gefunden, der das bezeugt. Somit wird T jede Eingabe y in polynomieller Zeit akzeptieren – denn entweder, die simulierte Maschine M_i akzeptiert auf einem Rechenweg, oder nicht. Daher ist die Relation

$$R := \{(r, y) \mid T \text{ akzeptiert } y \text{ auf Rechenweg } r\}$$

total, das heißt $\forall y \in \Sigma^* \; \exists x \in \Sigma^{\leq p(|y|)} : \; (x, y) \in R$, wobei p ein Polynom sei, das die Länge der Rechenwege von T beschränkt. Somit handelt es sich bei $\Omega := (R, p)$ um eine TFNP Instanz.

Wir zeigen nun, dass Ω tatsächlich ein vollständiges TFNP Problem ist. Sei dazu eine beliebige TFNP Instanz (S, q) gegeben, die o.B.d.A. auf dem Alphabet $\Sigma = \{0, 1\}$ definiert ist. Wir müssen zeigen, dass es Funktionen $f, g \in \mathrm{FP}^B$ gibt, sodass für jedes $y \in \Sigma^*$ und alle $x \in \Sigma^{\leq p(|f(y)|)}$ gilt

$$\big(x, f(y)\big) \in R \Longrightarrow \big(g(x, y), y\big) \in S \tag{2.1}$$

Sei j die Maschinennummer der nichtdeterministischen Maschine, die bei Eingabe x nichtdeterministisch alle möglichen $s \in \Sigma^{\leq q(|x|)}$ konstruiert, und anschließend in deterministischer Polynomialzeit prüft, ob $(s, x) \in S$. Akzeptierende Rechenwege der Maschine M_j auf Eingabe x kodieren also auf eindeutige Weise ein erfüllendes $s \in \Sigma^{\leq q(|x|)}$, welches $(s, x) \in S$ erfüllt. Die Reduktionen f und g definieren wir dann wie folgt:

$$\begin{aligned} f &: \Sigma^* \to \Sigma^*, && y \mapsto f(y) \quad := \langle j, y, 0^{|y|^j + j} \rangle \\ g &: \Sigma^* \times \Sigma^* \to \Sigma^*, && (x, y) \mapsto g(x, y) := x[1:] \end{aligned}$$

Dabei bezeichnet für $w \in \{0, 1\}^*$ der Ausdruck $w[1:]$ denjenigen Substring, der durch das Entfernen des führenden Bits von w entsteht.

Sei y in Σ^* beliebig, und es gelte nun $(x, f(y)) \in R$ für ein $x \in \Sigma^{\leq p(|y|)}$. Nach Definition von R ist dann x ein erfüllender Rechenweg der Maschine T bei Eingabe $f(y) = \langle j, y, 0^{|y|^j + j} \rangle$.

Ist x ein akzeptierender Rechenweg von T bei Eingabe $\langle j, y, 0^{|y|^j + j} \rangle$, sodass das erste Bit von x den Wert 1 hat, so hat die Maschine T auf dem restlichen Rechenweg nach der ersten Aufspaltung einen akzeptierenden Rechenweg von $M_j(y)$ gefunden, also ein erfüllendes Wort s mit $(s, y) \in S$. In diesem Fall stellt $g(x, y) = x[1:] = s$ also die erfüllende Belegung dar, d.h. es gilt $(g(x, y), y) = (s, y) \in S$. Es verbleibt zu zeigen, dass x tatsächlich mit 1 beginnt.

Nach Wahl von M_j akzeptiert sie y genau dann, wenn ein $z \in \Sigma^{\leq q(|y|)}$ existiert, sodass $(z, y) \in S$. Da (S, q) eine TFNP Instanz war, gilt immer die Totalität, weshalb M_j jede Eingabe akzeptiert, insbesondere auch y. Würde x mit 0 beginnen, so hätte nach Definition von x die Maschine T in der ersten Hälfte des Verzweigungsbaumes akzeptiert. Nach Konstruktion von T hätte T also einen Beweis dafür gefunden, welcher belegt, dass M_j die Eingabe y nicht akzeptiert. Ein solcher Beweis existiert jedoch nicht, weshalb x mit 1 beginnen muss. Damit folgt nun $(g(x, y), y) \in S$, und (2.1) ist bewiesen. Damit ist gezeigt, dass Ω TFNP-vollständig, und die Vermutung TFNP damit falsch wäre. □

Das in Satz 2.14 konstruierte TFNP-vollständige Problem Ω nimmt jede beliebige NP-Maschinennummer i auf jeder Eingabe x entgegen. Im Falle, dass M_i die Eingabe x akzeptiert, sind alle möglichen zurückgelieferten Lösungen leicht verifizierbare Beweise für die Akzeptanz. Falls M_i die Eingabe x hingegen ablehnt, kann die zurückgelieferte Lösung keinen Beweis für die Akzeptanz darstellen – denn ein solcher existiert ja garnicht. Da nun aber trotzdem eine Lösung zurückgegeben wird, muss diese falsch sein. Anhand dieser kann also leicht verifiziert werden, dass M_i die Eingabe x nicht akzeptiert. Wir fassen das wie folgt zusammen:

Satz 2.15 NP = coNP $\implies \Omega$ ist ein NP-hartes TFNP Problem

Beweis. Wegen 2.14 ist das dort konstruierte Problem $\Omega = (R, p)$ eine TFNP Instanz. Wir definieren $\Omega(y)$ als die möglichen Lösungen von Ω für y:

$$\Omega(y) := \left\{ x \in \Sigma^{\leq p(|y|)} \;\middle|\; (x, y) \in R \right\}$$

Das entspricht also den akzeptierenden Rechenwegen der dort konstruierten NP-Maschine T bei Eingabe y. Die Zeichenketten in $\Omega(y)$, jeweils ohne das führende Zeichen, repräsentieren also den "verwertbaren" Teil der Lösungen: Falls T akzeptierende Rechenwege der betrachteten Maschine gefunden hat, entspricht $\Omega(y)$ genau den Rechenwegen mit jeweils einer führenden 1. Falls keine akzeptierenden Rechenwege existieren, sind beliebige mit 0 beginnende Strings in $\Omega(y)$ enthalten. In allen Fällen ist $\Omega(y) \neq \emptyset$, und entweder alle enthaltenen Strings beginnen mit 0, oder alle beginnen mit 1.

Sei nun eine beliebige NP-Maschinennummer i und eine Eingabe n gegeben. Mit $y := \langle i, n, 0^{|n|^i+i} \rangle$ gilt dann

$$M_i \text{ akzeptiert } n \iff \forall x \in \Omega(y) : x[0] == 1 \tag{2.2}$$

$$M_i \text{ akzeptiert } n \text{ nicht} \iff \forall x \in \Omega(y) : x[0] == 0 \tag{2.3}$$

Die Äquivalenz (2.2) folgt, weil T in diesem Falle nur beim Nachsimulieren akzeptierender Rechenwege von $M_i(n)$ akzeptiert. Äquivalenz (2.3) folgt, weil dann M_i bei Eingabe n garkeine akzeptierenden Rechenwege hat – jede Lösung also mit 0 beginnen muss.

Zu gegebenem i existiert eine Maschine f, die in deterministischer Polynomialzeit die entsprechende Eingabe $y = (i, n, 0^{|n|^i+i})$ erzeugt, und die Frage an das Problem Ω stellt. Durch Auswerten des führenden Zeichens erhält f die Gewissheit, ob die Maschine M_i die Eingabe n akzeptiert, oder nicht. Akzeptiert sie entsprechend, entscheidet sie $L(M_i)$. Damit ist Ω ein NP-hartes TFNP Problem. □

Zusammenfassend erhalten wir eine erfreuliche Erkenntnis.

Korollar 2.16 Die folgenden Aussagen sind äquivalent:

i) Ω ist NP-hartes TFNP Problem

ii) Es gibt NP-harte TFNP Probleme

iii) NP = coNP

Beweis. „$i) \Longrightarrow ii)$“ Klar.
„$ii) \Longrightarrow iii)$“ folgt aus Satz 2.13.
„$iii) \Longrightarrow i)$“ folgt aus Satz 2.15. □

Die Frage nach der NP-härte von TFNP[3] ist also völlig äquivalent zur Frage, ob NP = coNP. Und im Falle der Existenz NP-harter TFNP Probleme kennen wir sogar schon eines davon explizit.[4] Kombinieren wir die Ergebnisse zur NP-Härte mit denen zur TFNP-Vollständigkeit, ergibt sich nun das folgende Bild:

$$\text{Es gibt NP-harte TFNP Probleme} \overset{2.16}{\Longleftrightarrow} \text{NP} = \text{coNP} \overset{2.14}{\Longrightarrow} \neg\text{TFNP}$$

Die Frage, ob die Existenz eines TFNP-vollständigen Problemes auch die eines NP-harten TFNP Problemes impliziert, ist also tatsächlich gänzlich äquivalent zur Frage, ob auch die Rückrichtung von Satz 2.14 gilt – also der eingangs gestellten Frage, ob ein TFNP-vollständiges Problem immer auch selbst Beweise für die Nichtexistenz von Lösungen erbringen können muss. Diese Rückrichtung von Satz 2.14 kann nicht mit relativierbaren Beweismethoden gezeigt werden [FR03].

Kombiniert man diese Ergebnisse mit der Tatsache, dass das effiziente Lösen von TFNP Instanzen P = NP ∩ coNP impliziert [FFNR], erhalten wir Tabelle 2.1.

	TFNP $\subseteq$ P	TFNP $\not\subseteq$ P
SAT $\leq_T^p$ TFNP	P = NP (= coNP)	P $\neq$ NP = coNP
SAT $\not\leq_T^p$ TFNP	P = NP ∩ coNP $\neq$ NP	P $\neq$ NP $\neq$ coNP

Tabelle 2.1: Implikationen der für TFNP möglichen Eigenschaften

[3] Aus Gründen der Lesbarkeit sprechen wir hier und in Tabelle 2.1 nur von der „NP-Härte von TFNP“, statt von der Existenz mindestens eines TFNP-Problemes, das NP-hart ist.

[4] Wobei hier anzumerken ist, dass die Definition von Ω von der Maschine T abhängt, deren exakte Arbeitsweise eine Methode inkludiert, ein coNP-vollständiges Problem zu lösen. Daher können wir Ω nicht konkret angeben, sondern wissen nur, wie man es erhalten kann, wenn NP = coNP.

Kapitel 3

Orakelkonstruktion

In diesem Abschnitt werden wir ein Orakel Φ konstruieren, bezüglich dem SAT^Φ keine p-optimalen Beweissysteme hat, jedoch $\mathrm{NP}^\Phi = \mathrm{coNP}^\Phi$.

3.1 Vorbereitung

3.1.1 Hilfsmittel

Im gesamten restlichen Kapitel sei M eine nichtdeterministische Polynomialzeit-Orakelturingmaschine, die bei Verwendung jedes beliebigen Orakels B die Menge SAT^B akzeptiert, und dabei für ein $n \in \mathbb{N}^+$ bei beliebiger Eingabe $x \in \Sigma^*$ die Laufzeit $t(x) := |x|^n + n$ benötigt.

Hilfsmittel für SAT

Um zu erreichen, dass SAT^Φ keine p-optimalen Beweissysteme hat, müssen wir erreichen, dass für jedes potenzielle Beweissystem f von SAT^Φ jeweils mindestens ein anderes Beweissystem g für SAT^Φ nicht von f p-simuliert wird. Die Kontraposition des folgenden Lemma 3.1 ermöglicht uns ein technisch weniger kompliziertes Vorgehen: es genügt, zu gegebenem f jeweils nur ein Beweissystem für eine beliebige Teilmenge von SAT^Φ zu konstruieren, welches nicht von f p-simuliert wird.

Lemma 3.1 Sei $B \subseteq \Sigma^*$ ein beliebiges Orakel, und sei $f \in \mathrm{FP}^B$ ein p-optimales Beweissystem für $f(\Sigma^*)$. Dann gilt auch für jedes $g \in \mathrm{FP}^B$ mit $g(\Sigma^*) \subseteq f(\Sigma^*)$:

$$\exists h \in \mathrm{FP}^B \; \forall x \in \Sigma^* : \; f(h(x)) = g(x)$$

Beweis. Betrachte die Funktion

$$g' : \Sigma^* \to \Sigma^*, \; x \mapsto g'(x) := \begin{cases} f(\varepsilon) & \text{falls } x = \varepsilon \\ g(x[:-1]) & \text{falls } x[-1] = 0 \\ f(x[:-1]) & \text{falls } x[-1] = 1 \end{cases}$$

Damit ist $g'(\Sigma^*) = f(\Sigma^*)$, und deswegen wird g' nach Annahme durch f p-simuliert. Es existiert also eine Funktion h' sodass $\forall x \in \Sigma^* : f(h'(x)) = g'(x)$. Wähle nun die Funktion h definiert durch $h(x) := h'(x \cdot 0)$. Damit gilt für alle $x \in \Sigma^*$:

$$g(x) = g'(x \cdot 0) = f(h'(x \cdot 0)) = f(h(x)) \qquad \square$$

Bemerkung
Wir sagen daher auch im Fall $g(\Sigma^*) \subsetneq f(\Sigma^*)$, dass g von f p-simuliert wird.

Definition 3.2 (h_x, H_i)
Für $w \in \Sigma^*$ definieren wir die Formel

$$h_w := \tau^{-1}\big(B(w_1, \ldots, w_{|w|}, 0, x_1, \ldots, x_{t(w)})\big),$$

wobei die x_i entsprechend Definition 2.2 Punkt i) jeweils freie Variablen darstellen. Für $i \in \mathbb{N}$ definieren wir nun die Funktion

$$g_i : \Sigma^* \to \Sigma^*, \quad x =: \langle bin(i'), bin(j), b\rangle \;\mapsto\; g_i(x) := h_{\langle bin(i), bin(j), b\rangle},$$

sowie die von g_i bewiesenen Formeln $H_i := g_i(\Sigma^*)$. Die erste Komponente, i', wird von g_i also ignoriert, und stattdessen immer i verwendet.

Korollar 3.3 Für beliebiges $B \subseteq \Sigma^*$, alle $i \in \mathbb{N}$ und alle $x \in \Sigma^*$ gilt

i) $g_i \in \mathrm{FP}^B$

ii) $h_x \in \mathrm{SAT}^B \iff \exists w \in \Sigma^{t(x)} :\; x \cdot 0 \cdot w \in B$

iii) $g_i(\Sigma^*) \subseteq \mathrm{SAT}^B \iff \forall j \in \mathbb{N} \forall b \in \Sigma^* \; \exists w \in \Sigma^{t(x)} :\; \langle bin(i), bin(j), b\rangle \cdot 0 \cdot w \in B$

Beweis. Aussage i) ist klar, ii) folgt direkt aus den Definitionen $\big($2.2iii$\big)$ und 2.4$\big)$, und iii) ist äquivalent zu ii). □

Da es sich bei den H_i um aussagenlogische Formeln handelt, deren Wahrheitswert jeweils vom Orakel abhängt, können wir durch geschickte Platzierung gewisser Wörter beeinflussen, ob sie erfüllbar sind oder nicht. Wir werden also bei der Konstruktion von Φ kontrollieren, ob $g_i(\Sigma^*) = H_i \subseteq \mathrm{SAT}^\Phi$. Denn falls dies gilt, müssen wir dank Lemma 3.1 nur noch erreichen, dass g_i nicht von f_i p-simuliert wird – spätestens dann wäre f_i kein p-optimales Beweissystem mehr.

Hilfsmittel für NP = coNP

Definition 3.4 Für beliebiges $B \subseteq \Sigma^*$ definieren wir die Menge

$$Z^B := \Big\{ x \in \Sigma^* \;\Big|\; \exists w \in \Sigma^{t(x)} :\; x \cdot 1 \cdot w \in B \Big\}$$

Korollar 3.5 Für beliebiges $B \subseteq \Sigma^*$ ist $Z^B \in \mathrm{NP}^B$.

Wir werden das Orakel Φ so konstruieren, dass schließlich $Z^\Phi = \overline{\mathrm{SAT}^\Phi}$, also coNP^Φ hart ist. Damit folgt dann $\mathrm{NP}^\Phi = \mathrm{coNP}^\Phi$.

3.1.2 Ziele

Definition 3.6 (Angezielte Eigenschaften von Φ)
Das von uns konstruierte Orakel Φ soll folgende Eigenschaften haben.

I $\forall x \in \Sigma^* : \left(x \notin \mathrm{SAT}^\Phi \iff x \in Z^\Phi\right)$

II $\forall i \in \mathbb{N}$ gilt mindestens eine der beiden Aussagen

II.a $\exists x \in \Sigma^* : f_i(x) \notin \mathrm{SAT}^\Phi$ (d.h. f_i ist kein Beweissystem für SAT^Φ)

II.b $g_i(\Sigma^*) \subseteq f_i(\Sigma^*)$ und
$\forall j \in \mathbb{N}\ \exists x \in \Sigma^* : f_i(f_j(x)) \neq g_i(x)$ (d.h. g_i wird nicht von f_i p-simuliert)

Korollar 3.7
Sei $\Phi \subseteq \Sigma^*$ beliebig.

i) Gilt Aussage **I** für Φ, ist $\mathrm{NP}^\Phi = \mathrm{coNP}^\Phi$.

ii) Gilt Aussage **II** für Φ, existiert kein p-optimales Beweissystem für SAT^Φ.

Beweis. Aussage i) folgt mit Korollar 3.5, weil $\overline{\mathrm{SAT}^\Phi}$ $\leq_m^p$-vollständig für coNP^Φ ist. Aussage ii) folgt direkt aus der Definition eines p-optimalen Beweissystemes (Definition 2.9) in Kombination mit der Einsicht, dass alle Beweissysteme in der Aufzählung $(f_i)_{(i\in\mathbb{N})}$ aufgeführt werden. □

Wir werden **I** erreichen, indem wir das Prädikat aus Z^Φ genau dann wahr werden lassen, wenn die Maschine M *nicht* akzeptiert. Konkret werden wir dazu für $x \in \Sigma^*$ genau dann ein Wort $x \cdot b \cdot w$ hinzufügen, wenn $\mathrm{M}^\Phi(x) = \texttt{false}$. Die Wahl $w \in \Sigma^{t(x)}$ hat dabei zur Folge, dass das hinzugefügte Wort bei der Berechnung $\mathrm{M}^\Phi(x)$ nicht erfragt werden kann, denn $\mathrm{M}^B(x)$ hat eine Laufzeit von maximal $t(x)$ Schritten.

Aussage **II** erfordert hingegen etwas mehr Aufwand. Wir diagonalisieren hier über alle Paare an Maschinen f_i, f_j, und erzwingen jeweils, dass **II.a** oder **II.b** gilt. Falls $f_i \circ f_j$ eine Formel $h \in H_i$ zurückgibt, ohne dass wir bereits h erfüllbar gemacht haben, so gilt bereits **II.a**. Falls nicht, so müssen wir die von g_i bei Eingabe x bewiesene Aussage erfüllbar machen, und dafür sorgen, dass $f_i(f_j(x)) \neq g_i(x)$.

3.1.3 Stufen

Zunächst wählen wir gewisse Stufen m, die sich der Behandlung von SAT widmen. Sie müssen hinreichend weit voneinander gewählt werden – unter Anderem, um einander nicht zu beeinflussen, jedoch würde das noch nicht genügen. Die zusätzlichen Anforderungen sind jedoch größtenteils technischer Natur, und es ist nicht schwer, diese zu bezwecken. Daher gestaltet es sich als sinnvoller, die Bedingungen zwar direkt zu Beginn zu formulieren, aber erst an späterer Stelle zu motivieren. Dazu wird in denjenigen Beweisschritten, die sie erfordern, klar darauf hingewiesen. Bis dahin soll es genügen, zu wissen, dass diese Stufen „hinreichend großen“ Abstand zueinander haben.

Wir wählen m als eine Funktion $m : \mathbb{N} \times \mathbb{N} \to \Sigma^*, (i,j) \mapsto m(i,j)$, die folgende Anforderungen erfüllt:

M1 $m(i,j) = \langle bin(i), bin(j), b\rangle$ für ein $b \in \Sigma^*$

M2 $\forall x \geq_L m(i,j) : \; 2^{|x|} > (|m(i,j)|^j + j)^i + i$

M3 $m(i,j) >_L m(i',j') \implies f_{i'}(f_{j'}(m(i',j')))$ läuft in Zeit $< t(m(i,j))$

Da wir die Berechenbarkeit von m im Folgenden nicht benötigen, werden wir hier auch nicht darauf eingehen. Wir benötigen nur die Totalität – und ein geeigneter totaler Kandidat für m existiert gewiss, da jeweils nur b groß genug gewählt werden muss, sodass beide Anforderungen M2 und M3 erfüllt sind. Wir nennen die Werte der Funktion m im Folgenden „SAT-Stufen“, da nur auf Ihnen zu SAT^Φ (d.h. die Vermutung SAT bezogen auf das relativierte Problem SAT^Φ) beigetragen wird.

3.2 Konstruktion

Wir konstruieren das Orakel sequenziell für alle $x \in \Sigma^*$. Für das jeweilige x seien $i,j \in \mathbb{N}, b \in \Sigma^*$ so definiert, dass $\langle bin(i), bin(j), b\rangle = x$. Falls zuvor bereits eine SAT-Stufe erreicht wurde, bezeichnen wir die zuletzt besuchte mit $x' = \langle bin(i'), bin(j'), b'\rangle$. Für jedes $x \in \Sigma^*$ werden dann die folgenden Schritte durchgeführt.

Schritt 1 Es sei $w \in \Sigma^{t(x)}$ so gewählt, dass sowohl die Berechnung von $f_i(f_j(x))$ als auch die von $f_{i'}(f_{j'}(x'))$ (falls vorherige SAT-Stufe existent) keine der Fragen $x0w$ oder $x1w$ ans Orakel stellt.

Schritt 2 Falls M bei Eingabe x ablehnt, fügen wir das Wort $x1w$ zu Φ hinzu (für Aussage **I**).

Schritt 3 Falls $m(i,j) \neq x$ oder $f_i(f_j(x)) \neq h_x$, fügen wir das Wort $x0w$ zum Orakel hinzu (für Aussage **II**).

3.3 Korrektheit

Satz 3.8 Schritt 1 ist für jedes x möglich. Falls für ein x eine Prämisse aus Schritt 2 oder Schritt 3 galt, so wird sie nie durch hinzugefügte Wörter falsch.

Beweis. Schritt 1 ist möglich, weil wegen M2 und $t(x) > |x|$ hinreichend viele Kandidaten für $w \in \Sigma^{t(x)}$ zur Verfügung stehen, damit $f_i(f_j(x))$ gewisse Fragen nicht stellt. M3 stellt dabei sicher, dass höchstens eine SAT-Stufe berücksichtigt werden muss, da die Fragen ansonsten zu lang sein würden. Deswegen wird für kein x, das eine SAT-Stufe $x = \langle bin(i), bin(j), b\rangle$ ist, *jemals* ein Wort hinzugefügt, nach welchem das Orakel bei der Berechnung von $f_i(f_j(x))$ befragt wird.

Die Prämisse aus Schritt 2 wird nicht revidiert, weil die Berechnung von M bei Eingabe x nur $t(x)$ Schritte tätigen darf, also die Frage $x1w$ nicht stellen kann. Es werden also von nun an keine Wörter mehr zum Orakel hinzugefügt, die die Berechnung $\mathrm{M}^\Phi(x)$ beeinflussen.

Betrachten wir nun die Prämisse aus Schritt 3. Galt für ein x $m(i,j) \neq x$, so wird dies immer der Fall bleiben. Gilt hingegen $f_i(f_j(x)) \neq h_x$, wird dies nie durch hinzugefügte Wörter geändert, weil nach Wahl von w weder die Berechnung von $f_i(f_j(x))$, noch die von $f_{i'}(f_{j'}(x'))$ die Frage $x0w$ oder $x1w$ stellen. Andere Wörter werden bei dieser Stufe noch nicht hinzugefügt. Das auf diesem x erreichte Fehlverhalten von $f_i(f_j(x))$ wird nicht durch zukünftige Orakelwörter beeinflusst, denn auch in zukünftigen Stufen wird immer ein Wort hinzugefügt, das von der $f_i(f_j(x))$ nicht befragt wird: bis zur nächsten SAT-Stufe wird das Wort w sogar dediziert so gewählt, und ab der nächsten SAT-Stufe werden die hinzugefügten Wörter wegen M3 zu lang sein. □

Lemma 3.9 Bezüglich des Orakels Φ gilt **I**.

Beweis. Für alle $x \in \Sigma^*$ gilt wegen Schritt 2

$$x \notin \mathrm{SAT}^\Phi \Longrightarrow \exists w \in \Sigma^{t(x)} : \ x1w \in \Phi$$

„$\Leftarrow$“ gilt, weil solche Wörter in keinem anderen Fall hinzugefügt werden. □

Lemma 3.10 Für beliebiges $x \in \Sigma^*$ gilt

$$f_i(\Sigma^*) \subseteq \mathrm{SAT}^\Phi \Longrightarrow \forall j \in \mathbb{N}, b \in \Sigma^* : \ h_{\langle bin(i),bin(j),b\rangle} \in \mathrm{SAT}^\Phi$$

Beweis. Für beliebiges $x \in \Sigma^*$ mit $x = \langle bin(i), bin(j), b\rangle$ wird das Wort $x0w$ genau dann *nicht* zum Orakel hinzugefügt, wenn $m(i,j) = x$ und $f_i(f_j(x)) = h_x$. Es gilt also

$$h_x \notin \mathrm{SAT}^\Phi \Longleftrightarrow m(i,j) = x \wedge f_i(f_j(x)) = h_x \Longrightarrow f_i(\Sigma^*) \not\subseteq \mathrm{SAT}^\Phi$$

Ist f_i ein Beweissystem für SAT^Φ, muss $h_{\langle bin(i),bin(j),b\rangle}$ für jedes $j \in \mathbb{N}$ und $b \in \Sigma^*$ erfüllbar sein. □

Satz 3.11 Bezüglich des Orakels Φ gilt **II**.

Beweis. Sei $i \in \mathbb{N}$ beliebig. Gilt **II.a** sind wir fertig, es gelte nun also $f_i(\Sigma^*) \subseteq \mathrm{SAT}^\Phi$. Dann muss nach Lemma 3.10 für alle $j \in \mathbb{N}, b \in \Sigma^*$ mit $x := \langle bin(i), bin(j), b\rangle$ die Formel h_x in SAT^Φ sein. Die Formel h_x ist nach Korollar 3.3 Punkt ii) genau dann in SAT^Φ, wenn wir ein Wort $x0w$ mit $w \in \Sigma^{t(x)}$ hinzugefügt haben. Wir haben also für jedes x, das für gewisse $\alpha, \beta \in \Sigma^*$ von der Form $x = \langle bin(i), \alpha, \beta\rangle$ ist, in Schritt 3 das Wort $x0w$ hinzugefügt. Außerdem ist $g_i(\Sigma^*) = H_i \subseteq \mathrm{SAT}^\Phi = f(\Sigma^*)$ (ebenfalls nach Korollar 3.3, Punkt iii)).

Es bleibt zu zeigen, dass $\forall j \in \mathbb{N}\ \exists x \in \Sigma^* : f_i(f_j(x)) \neq g_i(x)$. Sei also $j \in \mathbb{N}$ beliebig. Weil m eine totale Funktion ist gibt es (genau) ein $x = m(i,j)$, das nach Wahl der SAT-Stufen (M1) für gewisses $\beta \in \Sigma^*$ von der Form $x = \langle bin(i), bin(j), \beta\rangle$ ist, und für das nach obiger Überlegung ein Wort $x0w$ hinzugefügt worden sein muss. Deshalb muss wegen $m(i,j) = x$ in Schritt 3 der Fall $f_i(f_j(x)) \neq h_x$ eingetreten sein, welcher nach Satz 3.8 auch nach Beendigung der Konstruktion bestehen bleibt. Damit ist

$$f_i(f_j(x)) \neq h_x \overset{Def.\ g_i}{=} g_i(x),$$

womit **II.b** für i gilt.

□

Satz 3.12 Es existiert ein Orakel O, bezüglich dessen keine p-optimalen Beweissysteme für SAT^O existieren, aber $\mathrm{NP}^O = \mathrm{coNP}^O$.

Beweis. Folgt mit $O := \Phi$ aus den Resultaten 3.9, 3.11, und 3.7. □

Kapitel 4

Folgerungen

4.1 Ergebnisse

Das Orakel zeigt zunächst nur eine Separation von SAT und der in diesem Zusammenhang bisher wenig beachteten Vermutung NP $\neq$ coNP. Doch aufgrund der relativierbaren Implikationen lassen sich dadurch nun drei von vier offenen Fragen aus Abb. 1.1 beantworten.

4.1.1 Anwendung auf SAT

Korollar 4.1 Es existiert ein Orakel O mit SAT^O und $\neg\mathrm{TFNP}^O$.

Beweis. Wähle $O = \Phi$. $\mathsf{TFNP} \Longrightarrow \mathrm{NP} \neq \mathrm{coNP}$ ist relativierbar, weswegen aus $\mathrm{NP}^O = \mathrm{coNP}^O$ und SAT^O die Aussage folgt. □

Und wegen der Relativierbarkeit von $\mathsf{DisjCoNP} \Longrightarrow \mathrm{TFNP}$ erhalten wir nun auch direkt die Antwort auf die Frage, ob $\mathsf{SAT} \Longrightarrow \mathsf{DisjCoNP}$ relativierbar bewiesen werden kann.

Korollar 4.2 Es existiert ein Orakel O mit SAT^O und $\neg\mathsf{DisjCoNP}^O$.

Ordnen wir nun unsere Resultate ins Gesamtbild der bereits bekannten Beziehungen ein. Für die Vermutungen SAT, TFNP, NP $\neq$ coNP und P $\neq$ NP ergibt sich ein bezüglich relativierbarer Beweismethoden vollständiges Gesamtbild: Jede der möglichen Implikationen ist nun entweder relativierbar bewiesen, oder es ist ein Orakel bekannt, bezüglich dessen die Implikation nicht gilt. Abbildung 4.1 veranschaulicht den Zusammenhang der Vermutungen. Die darin zutrage kommenden Orakelseparationen werden wir nun im restlichen Abschnitt erläutern.

Zum einen ist ein Orakel[FR03] bekannt, bezüglich dessen NP $\neq$ coNP gilt, aber $\neg\mathsf{SAT}$.[1] Dieses Orakel zeigt also insbesondere auch, dass die Implikation $\mathrm{NP} \neq \mathrm{coNP} \Longrightarrow \mathsf{TFNP}$ nicht mit relativierbaren Beweismethoden gezeigt werden kann (denn $\mathsf{TFNP} \Longrightarrow \mathsf{SAT}$ ist relativierbar). Weiterhin zeigt dieses Orakel implizit auch die Nichtrelativierbarkeit der Implikation $\mathrm{P} \neq \mathrm{NP} \Longrightarrow \mathsf{SAT}$ (denn $\mathrm{NP} \neq \mathrm{coNP} \Longrightarrow \mathrm{P} \neq \mathrm{NP}$ ist relativierbar).

[1] Aufgrund eines Resultates von Köbler und Messner [KM00], dass $\neg\mathsf{SAT}$ aus der vom Orakel erfüllten Eigenschaft $\mathrm{TFNP} \subseteq \mathrm{FP}$ folgt.

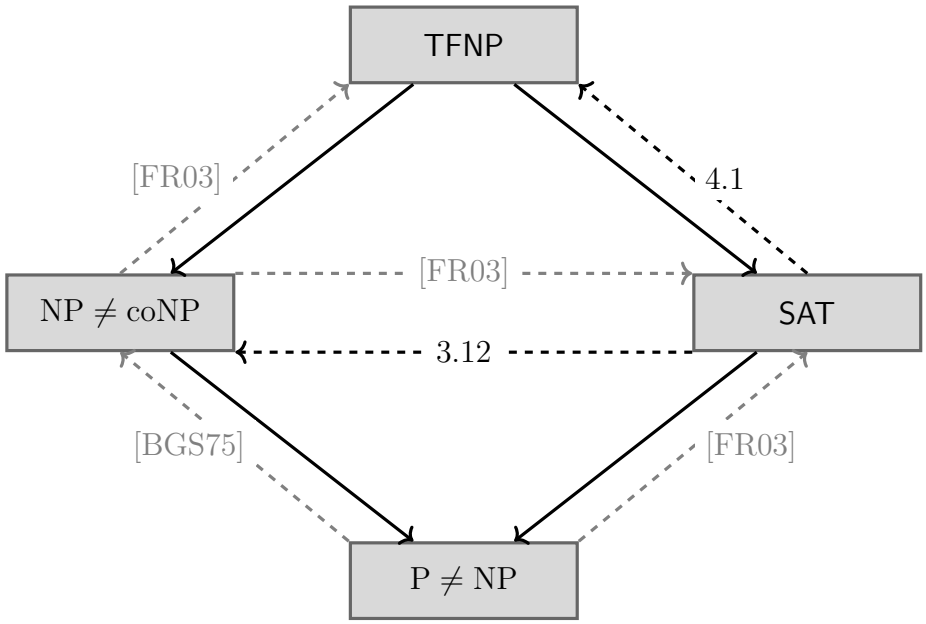


Abbildung 4.1: Zusammenhang von TFNP, SAT, NP ≠ coNP und P ≠ NP. Durchgehende Pfeile bedeuten relativierbare Implikationen. Ein gestrichelter Pfeil von A nach B bedeutet, dass es ein Orakel O gibt, relativ zu dem $A \wedge \neg B$ gilt. Schwarze gestrichelte Pfeile sind neue Resultate. Analog kann ein Diagramm mit CON und $\mathsf{CON}^{\mathsf{N}}$ anstelle von SAT und TFNP gezeichnet werden.

Auf der anderen Seite zeigt unser Orakel aus Satz 3.12, dass die Implikation SAT $\Longrightarrow$ NP $\neq$ coNP nicht mit relativierbaren Beweismethoden gezeigt werden kann, und somit insbesondere auch nicht SAT $\Longrightarrow$ TFNP. Und zuletzt existiert auch ein Orakel mit P $\neq$ NP = coNP [BGS75]. Insgesamt erhalten wir die komplette Unabhängigkeit der relativierten Vermutungen SAT und NP $\neq$ coNP.

4.1.2 Anwendung auf TAUT

Tatsächlich zeigt dieses Orakel ebenfalls die Nichtrelativierbarkeit von CON $\Longrightarrow$ NP $\neq$ coNP, denn bezüglich Φ gilt CON. Das liegt daran, dass durch NP = coNP p-optimale Beweissysteme für eine NP vollständige Menge gleichzeitig auch p-optimale Beweissysteme einer coNP vollständigen Menge sind. Daher zeigt das Orakel Φ analog zu den Aussagen über SAT, TFNP und DisjCoNP auch Aussagen über CON, $\mathsf{CON}^{\mathsf{N}}$ und DisjNP.

Korollar 4.3 Es existiert ein Orakel O, bezüglich dessen keine p-optimalen Beweissysteme für TAUT^O existieren, aber $\mathrm{NP}^O = \mathrm{coNP}^O$.

Beweis. Wähle $O := \Phi$. Wegen 3.9 gilt $\mathrm{NP}^O = \mathrm{coNP}^O$, weswegen TAUT als coNP^O vollständige Menge auch NP^O vollständig wäre. Insbesondere ist dann $\mathrm{SAT}^O \leq_m^p \mathrm{TAUT}^O$. Angenommen, es gäbe ein p-optimales Beweissystem für TAUT^O. Ein Resultat von Köbler, Messner und Torán [KMT03, Lemma 2.1] zeigt, dass dann auch jedes darauf reduzierbare Problem ein p-optimales Beweissystem hätten, also insbesondere SAT^O. Dies Widerspricht 3.11, weswegen kein p-optimales Beweissystem für TAUT^O existiert. □

Korollar 4.4 Es existiert ein Orakel O mit SAT^O und CON^O, aber $\neg(\mathsf{CON}^\mathsf{N})^O$.

Beweis. Wähle $O := \Phi$. Aus $\mathrm{NP}^\Phi = \mathrm{coNP}^O$ und der relativierbaren Implikation $\mathsf{CON}^\mathsf{N} \Longrightarrow \mathrm{NP} \neq \mathrm{coNP}$ folgt dann $\neg(\mathsf{CON}^\mathsf{N})^O$. □

4.2 Offene Fragen

Zeichnen wir in Abb. 1.1 auch Pfeile für implizite bewiesene Nichtrelativierbarkeiten ein, erhalten wir Abb. 4.2.

Insgesamt fehlen nurnoch zwei von Pudlák geforderte Orakel:

1. Gibt es ein Orakel O mit $\mathrm{TFNP}^O \wedge \neg\mathsf{DisjCoNP}^O$?
2. Gibt es ein Orakel O mit $\mathrm{UP}^O \wedge \neg\mathsf{DisjNP}^O$?

Wir möchten anmerken, dass bezüglich unseres Orakels $\neg\mathsf{UP}$ gilt, und es daher nicht zur Lösung der zweiten Frage beitragen kann. Auf einen Beweis verzichten wir an dieser Stelle, geben jedoch eine grobe Begründung, warum ein UP^Φ-vollständiges Problem existiert: Wir haben nur genau ein Wort im Orakel platziert, das die Unerfüllbarkeit einer Formel bezeugt. Es lässt sich also eine Maschine konstruieren, die einen Rechenweg einer anderen Maschine simuliert, und die nur auf genau einem Rechenweg erkennen würde, wenn es sich um den kleinsten akzeptierenden Rechenweg handelt. Damit lässt sich also eine Maschine konstruieren, die eine NP-vollständige Sprache akzeptiert, und dabei auf höchstens einem Rechenweg akzeptiert.

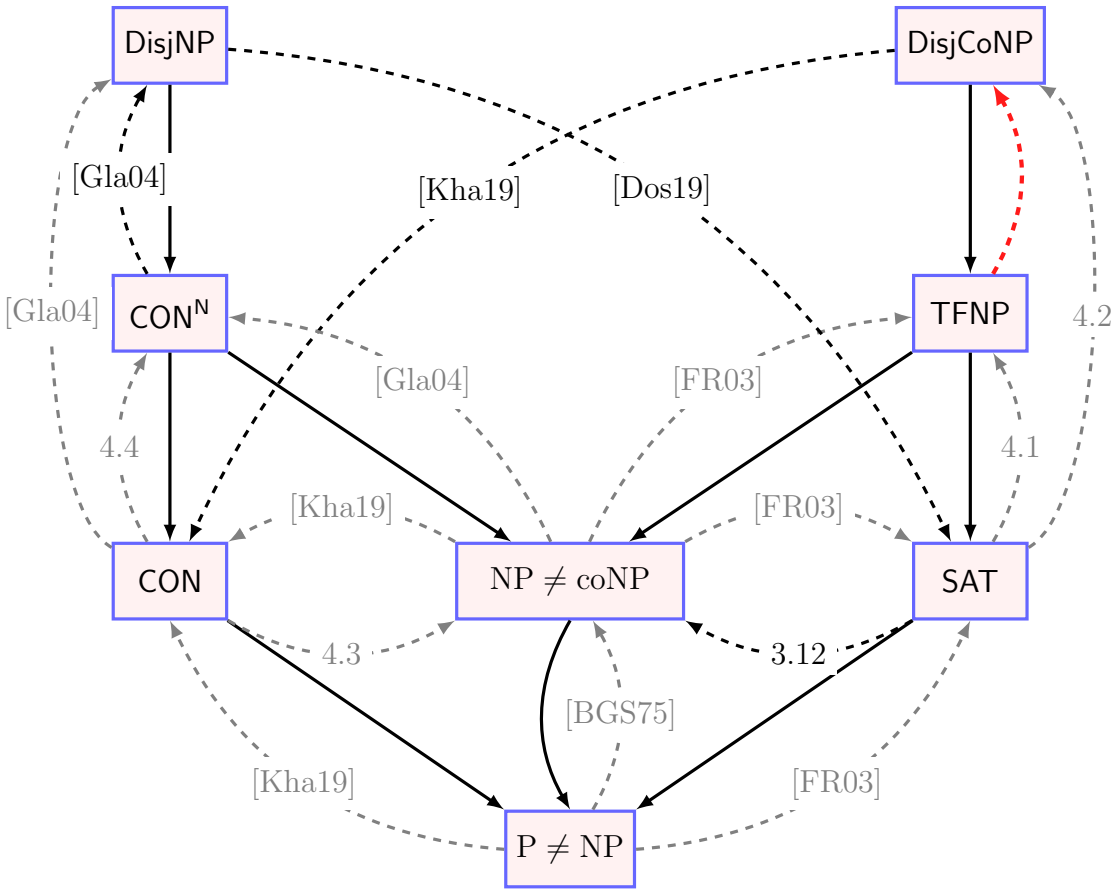


Abbildung 4.2: Bekannte Orakelseparationen. Durchgehende Pfeile bedeuten relativierbare Implikationen. Ein gestrichelter Pfeil von A nach B bedeutet, dass es ein Orakel O gibt, relativ zu dem $A \wedge \neg B$ gilt. Nur die Relativierbarkeit der rot markierten Implikation ist unbekannt. Ein grauer Pfeil folgt aus einem anderen eingezeichneten Resultat. Alle eigenen Resultate waren zuvor unbekannt, und folgen aus 3.12 mit dem selben Orakel. Die implizit aus den Separationen von Dose [Dos19] und Khaniki [Kha19] folgenden Separationen zwischen $\{\mathsf{DisjNP}, \mathsf{CON}^{\mathsf{N}}, \mathsf{CON}\}$ und $\{\mathsf{DisjCoNP}, \mathsf{TFNP}, \mathsf{SAT}\}$ wurden zur Übersicht weggelassen.

Literatur

[AW09] Scott Aaronson und Avi Wigderson. „Algebrization: A New Barrier in Complexity Theory“. In: *ACM Trans. Comput. Theory* 1.1 (2009). ISSN: 1942-3454. DOI: 10.1145/1490270.1490272. URL: https://doi.org/10.1145/1490270.1490272 (besucht am 22. 08. 2022).

[BG81] Charles H. Bennett und John Gill. „Relative to a Random Oracle A, $P^A \neq NP^A \neq co\text{-}NP^A$ with Probability 1“. en. In: *SIAM Journal on Computing* 10.1 (Feb. 1981), S. 96–113. ISSN: 0097-5397, 1095-7111. DOI: 10.1137/0210008. URL: http://epubs.siam.org/doi/10.1137/0210008 (besucht am 06. 02. 2021).

[BGS75] Theodore Baker, John Gill und Robert Solovay. „Relativizations of the $\mathcal{P} =? \mathcal{NP}$ Question“. In: *SIAM Journal on Computing* 4.4 (Dez. 1975), S. 431–442. ISSN: 0097-5397. DOI: 10.1137/0204037. URL: https://epubs.siam.org/doi/10.1137/0204037 (besucht am 21. 08. 2022).

[C94] Richard Chang u. a. „The random oracle hypothesis is false“. en. In: *Journal of Computer and System Sciences* 49.1 (Aug. 1994), S. 24–39. ISSN: 0022-0000. DOI: 10.1016/S0022-0000(05)80084-4. URL: https://www.sciencedirect.com/science/article/pii/S0022000005800844 (besucht am 12. 02. 2021).

[DG19] Titus Dose und Christian Glaßer. „NP-completeness, proof systems, and disjoint NP-pairs“. In: (2020).

[Dos19] Titus Dose. „P-Optimal Proof Systems for Each Set in NP but no Complete Disjoint NP-pairs Relative to an Oracle“. In: (Apr. 2019).

[Dos20a] Titus Dose. „An oracle separating conjectures about incompleteness in the finite domain“. In: *Theoretical Computer Science* 809 (2020), S. 466–481. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2020.01.003. URL: https://www.sciencedirect.com/science/article/pii/S0304397520300116.

[Dos20b] Titus Dose. „Further oracles separating conjectures about incompleteness in the finite domain“. In: *Theoretical Computer Science* 847 (2020), S. 76–94. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2020.09.040. URL: https://www.sciencedirect.com/science/article/pii/S030439752030551X.

[Dos21] Titus Dose. „Balance Problems for Integer Circuits and Separations of Relativized Conjectures on Incompleteness in Promise Classes“. doctoralthesis. Universität Würzburg, 2021. DOI: 10.25972/OPUS-22220.

[EEG22] Anton Ehrmanntraut, Fabian Egidy und Christian Glaßer. *Oracle with $P = NP \cap coNP$, but no Many-One Completeness in UP, DisjNP, and DisjCoNP.* 2022. DOI: 10.48550/ARXIV.2203.11079. URL: https://arxiv.org/abs/2203.11079 (besucht am 22. 08. 2022).

[FFNR] S. Fenner u. a. „Inverting onto functions“. In: 186.1 (Okt. 2003), S. 90–103. DOI: 10.1016/S0890-5401(03)00119-6. URL: https://dx.doi.org/10.1016/S0890-5401(03)00119-6.

[FR03] Lance Fortnow und John Rogers. „Separability and One-way Functions“. In: Bd. 11. Sep. 2003. ISBN: 978-3-540-58325-7. DOI: 10.1007/3-540-58325-4_204.

[Gla04] Christian Glasser u. a. „Disjoint NP-Pairs“. English. In: *SIAM Journal on Computing* 33.6 (2004), S. 1369–48. DOI: 10.1137/S0097539703425848. URL: https://www.proquest.com/scholarly-journals/disjoint-np-pairs/docview/918793976/se-2 (besucht am 22. 08. 2022).

[Kha19] Erfan Khaniki. *New relations and separations of conjectures about incompleteness in the finite domain*. 2019. DOI: 10.48550/ARXIV.1904.01362. URL: https://arxiv.org/abs/1904.01362 (besucht am 22. 08. 2022).

[KM00] Johannes Köbler und Jochen Messner. „Is the Standard Proof System for SAT P-Optimal?“ In: *FST TCS 2000: Foundations of Software Technology and Theoretical Computer Science*. Hrsg. von Sanjiv Kapoor und Sanjiva Prasad. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, S. 361–372.

[KMT03] Johannes Köbler, Jochen Messner und Jacobo Torán. „Optimal proof systems imply complete sets for promise classes“. In: *Information and Computation* 184.1 (2003), S. 71–92. ISSN: 0890-5401. DOI: https://doi.org/10.1016/S0890-5401(03)00058-0. URL: https://www.sciencedirect.com/science/article/pii/S0890540103000580.

[Pud17] Pavel Pudlak. „Incompleteness in the finite domain“. In: *arXiv:1601.01487 [math]* (Mai 2017). arXiv: 1601.01487. URL: http://arxiv.org/abs/1601.01487 (besucht am 24. 03. 2021).

[Raz97] Alexander A Razborov und Steven Rudich. „Natural Proofs“. In: *Journal of Computer and System Sciences* 55.1 (Aug. 1997), S. 24–35. DOI: 10.1006/jcss.1997.1494.

[Sha92] Adi Shamir. „IP = PSPACE“. In: *J. ACM* 39.4 (1992), 869–877. ISSN: 0004-5411. DOI: 10.1145/146585.146609. URL: https://doi.org/10.1145/146585.146609 (besucht am 22. 08. 2022).

[Tur39] A. M. Turing. „Systems of Logic Based on Ordinals†“. In: *Proceedings of the London Mathematical Society* s2-45.1 (1939), S. 161–228. DOI: https://doi.org/10.1112/plms/s2-45.1.161. eprint: https://londmathsoc.onlinelibrary.wiley.com/doi/pdf/10.1112/plms/s2-45.1.161. URL: https://londmathsoc.onlinelibrary.wiley.com/doi/abs/10.1112/plms/s2-45.1.161.

[Ver91] O. V. Verbitskii. „Optimal algorithms for coNP-sets and the EXP $\stackrel{?}{=}$ NEXP problem“. In: *Mathematical notes of the Academy of Sciences of the USSR* 50.2 (Aug. 1991), S. 796–801. ISSN: 1573-8876. DOI: 10.1007/BF01157564. URL: https://doi.org/10.1007/BF01157564 (besucht am 22. 08. 2022).

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel verwendet und die Arbeit keiner anderen Prüfungsbehörde unter Erlangung eines akademischen Grades vorgelegt habe.

Würzburg, den 22. November 2022

.....
David Dingel