Westfälische
Wilhelms-Universität
Münster

Fachbereich Mathematik und Informatik

Bachelorarbeit

Komplexitätsklassen und Hierarchiesätze

Autor:
Maurice Krause

Betreuende Dozentin:
Dr. Franziska Jahnke

Münster 2016

Inhaltsverzeichnis

1 Einleitung

In der heutigen Zeit haben Computer einen immer größeren Stellenwert. In diesem Zusammenhang stellt sich die Frage, welche Probleme ein Computer lösen kann und wie effizient dieser dabei ist. Während sich die Berechenbarkeitstheorie mit der grundlegenden Frage beschäftigt, was mit einem konkreten formalen Rechnermodell gelöst werden kann, widmet sich die Komplexitätstheorie der Effizienz dieser Berechnungen. Hierbei können ganz verschiedene Ressourcen beschränkt werden, in dieser Bachelorarbeit werden jedoch nur die Rechenzeit und der Speicherplatz behandelt. Anhand dieser Ressourcen teilt die Komplexitätstheorie die gegebenen Probleme in verschiedene Klassen auf. Die wohlbekanntesten Komplexitätsklassen sind $\mathcal{P}$ und $\mathcal{NP}$ und das damit verbundene Millennium-Problem. Man weiß zwar, dass alle Probleme aus $\mathcal{P}$ auch in $\mathcal{NP}$ liegen, aber nicht, ob es darüber hinaus noch Probleme gibt, die zwar in $\mathcal{NP}$, aber nicht in $\mathcal{P}$ liegen. Neben diesen beiden Komplexitätsklassen gibt es noch viele weitere, von denen einige in dieser Arbeit vorgestellt werden. Des Weiteren wird untersucht, in welchen Beziehungen diese zueinander stehen. Dabei werden wir feststellen, dass zwar eine grundlegende Anordnung der Komplexitätsklassen bekannt ist, in vielen Fällen aber nicht welche dieser Inklusionen echt sind.

Um diese Ziele zu erreichen, definieren wir in Kapitel 2 zunächst, was eine Turingmaschine ist. Wir betrachten sowohl die deterministische als auch die nichtdeterministische Variante, welche sich dadurch unterscheiden, dass im nichtdeterministischen Fall die Folgekonfigurationen nicht mehr eindeutig festgelegt sind. Im Anschluss führen wir die Komplexitätsklassen ein, die wir in dieser Arbeit behandeln werden. Wir beginnen mit den Zeitkomplexitätsklassen $\mathcal{P}$ und $\mathcal{NP}$, welche Sprachen enthalten, die in polynomieller Zeit von einer deterministischen beziehungsweise nichtdeterministischen Turingmaschine entschieden werden können. Danach folgen die Klassen $\mathcal{EXPTIME}$ und $\mathcal{NEXPTIME}$, welche Sprachen enthalten, die statt in polynomieller Zeit in exponentieller Zeit entschieden werden können. Daraufhin definieren wir die Platzkomplexitätsklassen $\mathcal{L}$ und $\mathcal{NL}$, die Probleme mit logarithmischem Platzbedarf enthalten, die Klassen $\mathcal{PSPACE}$ und $\mathcal{NPSPACE}$, welche auf polynomiellem Platz entscheidbare Probleme enthalten, sowie $\mathcal{EXPSPACE}$ und $\mathcal{NEXPSPACE}$, die Probleme enthalten, die auf exponentiellem Platz entschieden werden können. Parallel dazu werden wir einige Beispiele in die verschiedenen Komplexitätsklassen einordnen und das Kapitel damit abschließen, dass wir Schwere und Vollständigkeit einiger Komplexitätsklassen definieren.

In Kapitel 3 behandeln wir Aussagen, die uns bereits helfen einige Inklusionen festzustellen. Der Satz von Savitch liefert uns in Kapitel 4, dass die deterministischen Komplexitätsklassen $\mathcal{PSPACE}$ und $\mathcal{EXPSPACE}$ mit ihren nichtdeterministischen Äquivalenten

$\mathcal{NPSPACE}$ und $\mathcal{NEXPSPACE}$ übereinstimmen. Kapitel 5 klärt die Frage, ob sowohl deterministische als auch nichtdeterministische Turingmaschinen, die auf logarithmischem Platz arbeiten, auch in polynomieller Zeit arbeiten. Um die beiden echten Inklusionen $\mathcal{NP} \subsetneq \mathcal{PSPACE} \subsetneq \mathcal{EXPSPACE}$ zu beweisen, behandeln wir in Kapitel 6 den Platzhierarchiesatz.

Schlussendlich besprechen wir in Kapitel 7 noch das Spiel Tic–Tac–Toe. Die Standardversion, bei der zwei Personen gegeneinander auf einem 3×3 Spielfeld spielen und versuchen zuerst drei eigene Symbole in einer Reihe zu platzieren, erweitern wir zu einem $k \times k$ Spielfeld auf dem zum Sieg fünf eigene Symbole in einer Reihe benötigt werden. Nachdem wir zwei $\mathcal{PSPACE}$–vollständige Probleme vorgestellt haben, beweisen wir zunächst, dass das verallgemeinerte Tic–Tac–Toe in $\mathcal{PSPACE}$ liegt und anschließend mittels einer polyzeit Reduktion, dass es ebenfalls in $\mathcal{PSPACEC}$ liegt. Zur Verdeutlichung der einzelnen Schritte begleitet ein Beispiel dieses Kapitel.

Diese Arbeit basiert auf [Sip12]. Zum Verständnis ist es vorteilhaft die Grundlagen der Theoretischen Informatik, die beispielsweise in [Jah16] zu finden sind, bereits verstanden zu haben.

2 Grundlagen und Definitionen

2.1 Turingmaschinen

Zu Beginn wollen wir definieren, was genau eine Turingmaschine ist. Sie wurde 1936 von Alan Turing eingeführt und liefert ein starkes Rechnermodell zur Modellierung der Arbeitsweise eines Computers. Hauptbestandteil bildet das Schaltwerk in Verbindung mit der endlichen Kontrolleinheit, wodurch der Kopf auf einem unendlich langen Band gesteuert wird. Schrittweise können so die einzelnen Bandzellen angesprochen werden. Zunächst wird das aktuelle Zeichen gelesen, dann mit einem neuen Zeichen überschrieben und anschließend wird der Kopf eine Stelle nach links oder rechts bewegt. Zu erwähnen wäre noch, dass das neue Zeichen selbstverständlich mit dem alten übereinstimmen kann. Sowohl altes als auch neues Zeichen können das Blank–Symbol sein, welches eine leere Bandzelle repräsentiert. Ist die Berechnung der Turingmaschine unter einer Eingabe abgeschlossen, so hält sie an. Als Ergebnis `akzeptiert` oder `verwirft` sie. Liefert sie kein Ergebnis, da sie niemals anhält, so sagen wir, dass die Turingmaschine zykelt.

Spricht man vom Bandinhalt der Turingmaschine, so versteht man darunter jenes Wort, welches sich aktuell auf dem Band befindet, beginnend bei der ersten beschriebenen Bandzelle und endend bei der letzten beschriebenen Bandzelle. Blank–Symbole innerhalb dieses Wortes gehören, entgegen solchen außerhalb, ebenfalls zum Wort.

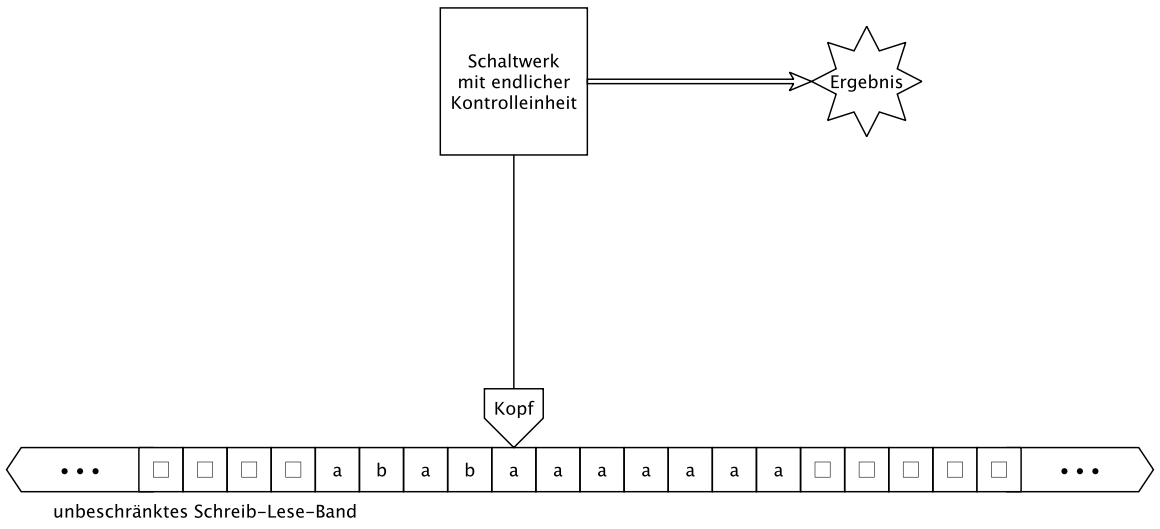

Abbildung 2.1: Turingmaschine mit Bandinhalt `ababaaaaaaa`

Unter der Konfiguration einer Turingmaschine verstehen wir Bandinhalt, Kopfposition und Zustand der gegenwärtigen Situation. Betrachtet man die Turingmaschine aus Abbildung 2.1 und bezeichnet den aktuellen Zustand mit q, dann ist `abab`q`aaaaaaaa` die entsprechende Konfiguration. Der Zustand wird also vor dem Zeichen, auf das der Kopf zeigt, notiert.

Definition 2.1. Eine deterministische Turingmaschine, kurz TM, ist ein 7–Tupel

$$M \coloneqq (Q, \Sigma, \Gamma, \delta, q_0, q_A, q_V)$$

mit folgenden Eigenschaften

- für die Zustandsmenge Q gilt $|Q| < \infty$
- für das Eingabealphabet Σ gilt $|\Sigma| < \infty$
- für das Arbeitsalphabet Γ gilt $|\Gamma| < \infty$
- es gilt $\Sigma \subsetneq \Gamma$
- für das Blank–Symbol $\square$ gilt $\square \in \Gamma \setminus \Sigma$
- $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ ist die Übergangsfunktion
- $q_0 \in Q$ ist der Startzustand
- $q_A \in Q$ ist der akzeptierende Zustand
- $q_V \in Q$ ist der verwerfende Zustand
- es gilt $q_A \neq q_V$

Die oben beschriebene Turingmaschine wird auch als deterministische Turingmaschine bezeichnet, da durch ihre Übergangsfunktion für jede Konfiguration eine eindeutige Folgekonfiguration vorgegeben ist. Modifiziert man die Übergangsfunktion, so erhält man eine nichtdeterministische Turingmaschine. Hierbei ist für eine konkrete Konfiguration nur eine Menge von Folgekonfigurationen vorgegeben. Zwar ist das Konzept der nichtdeterministischen Turingmaschine nicht praktisch realisierbar, es wird aber dennoch betrachtet.

Definition 2.2. Eine nichtdeterministische Turingmaschine, kurz NTM, ist ein 7–Tupel

$$N \coloneqq (Q, \Sigma, \Gamma, \delta, q_0, q_A, q_V)$$

mit $\delta : Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$ als Übergangsfunktion und den anderen Eigenschaften analog zur deterministischen Turingmaschine.

Definition 2.3. Sei N eine nichtdeterministische Turingmaschine mit Eingabe w. Ein Baum heißt *Berechnungsbaum von* $N(w)$, falls gilt:

1. Die Wurzel ist mit der Startkonfiguration von $N(w)$ beschriftet.
2. Die Blätter sind mit den akzeptierenden und verwerfenden Konfigurationen von $N(w)$ beschriftet.
3. Die verbleibenden Knoten sind mit den verbleibenden Konfigurationen von $N(w)$ beschriftet.
4. Für jeden Knoten mit Beschriftung C sind die Kinder mit Folgekonfigurationen von C beschriftet.

Berechnungsbäume müssen im Allgemeinen nicht endlich sein.

Definition 2.4. Im Berechnungsbaum von $N(w)$ heißt jeder Pfad von der Wurzel zu einem Blatt *Lauf von* $N(w)$.

Satz 2.5 ([Sip12, Theorem 3.16])**.** Jede nichtdeterministische Turingmaschinen kann durch eine deterministische Turingmaschinen simuliert werden.

Definition 2.6. Eine Turingmaschine heißt *Entscheider*, falls sie unabhängig von ihrer Eingabe nie zykelt, also immer ein Ergebnis liefert.

2.2 Bekannte Klassen

Ziel der nächsten beiden Abschnitte 2.3 und 2.4 ist es, verschiedene Komplexitätsklassen und Sprachen einzuführen und zu beweisen, in welchen Klassen diese Sprachen liegen. Zuvor werden wir einige Grundlagen wiederholen.

Wir wissen, dass deterministische endliche Automaten reguläre Sprachen und Kellerautomaten kontextfreie Sprachen erkennen können. Turingmaschinen können darüber hinaus weitere Sprachen erkennen.

Definition 2.7. Eine Turingmaschine M *erkennt* die Sprache, die genau die Wörter enthält, auf denen M akzeptierend hält.

Definition 2.8. Sei L eine Sprache. Die Frage, ob für ein Wort w gilt, dass $w \in L$ ist, bezeichnen wir als das zugehörige Sprachproblem.

Ob eine Sprache erkennbar, aufzählbar oder entscheidbar heißt, ist bedingt durch die Art der Turingmaschine, die die Sprache erkennt.

Definition 2.9. Eine Sprache L heißt *erkennbar* genau dann, wenn eine Turingmaschine existiert, die L erkennt.

Definition 2.10. Sei $L = \bigcup_i w_i$ eine Sprache. Eine Turingmaschine heißt *Aufzähler für* L, falls gilt:

1. Die Turingmaschine besitzt ein separates Ausgabeband, welches ausschließlich beschreibbar ist.
2. Nach jedem Zugriff auf das Ausgabeband bewegt sich der dazugehörige Kopf einen Schritt nach rechts und bleibt sonst stehen.
3. Die Turingmaschine schreibt $w_1\#w_2\#w_3\#\cdots$ auf das Ausgabeband, wobei $\# \notin L$ ist. Dabei darf die Turingmaschine Wörter auch mehrfach aufschreiben.

Definition 2.11. Eine Sprache L heißt *aufzählbar* genau dann, wenn ein Aufzähler für L existiert. Die Menge der aufzählbaren Sprachen wird mit $\mathbb{A}$ notiert.

Proposition 2.12 ([Sip12, Theorem 3.21])**.** Sei L eine Sprache, dann gilt

$$L \text{ ist aufzählbar} \iff L \text{ ist erkennbar.}$$

Definition 2.13. Eine Sprache L heißt *entscheidbar* genau dann, wenn ein Entscheider existiert, der L erkennt. Die Menge der entscheidbaren Sprachen wird mit $\mathbb{E}$ notiert.

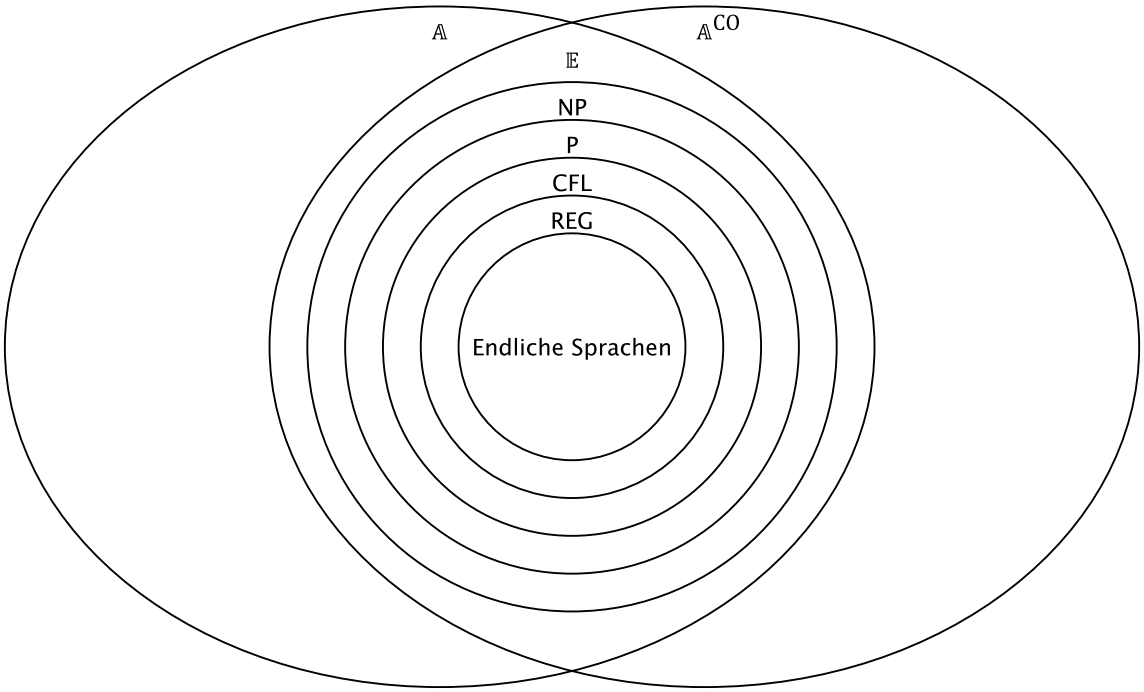

Abbildung 2.2: Bereits bekannte Klassen

In Abbildung 2.2 sind die bereits bekannten Klassen im Überblick dargestellt. Um die, in den beiden folgenden Abschnitten 2.3 und 2.4 zu erschließenden, Komplexitätsklassen besser einordnen zu können, geben die Abbildungen 2.3 und 2.4 eine Vorschau darüber, welche Inklusionen gelten. In Abbildung 2.3 steht die Unterscheidung zwischen echten und unechten Inklusionen im Vordergrund. Sie wird in Kapitel 8 um die entsprechenden Verweise ergänzt, an denen die Beweise gefunden werden können. Abbildung 2.4 stellt die Inklusionen erneut dar. Welche der Inklusionen echt sind, ist hier an den Linienfarben und –arten zu erkennen. In Kapitel 8 wird diese Abbildung um die Komplexitätsklassen $\mathcal{NLC}$, $\mathcal{NPC}$ und $\mathcal{PSPACEC}$ sowie um Beispiele von Problemen ergänzt.

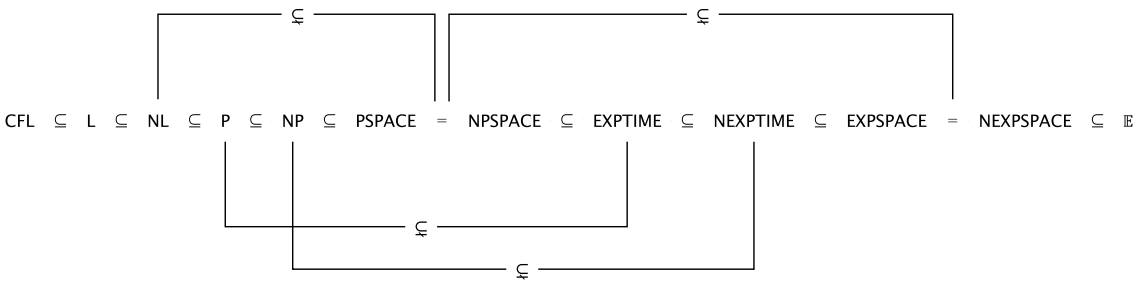

Abbildung 2.3: Inklusionen der Komplexitätsklassen

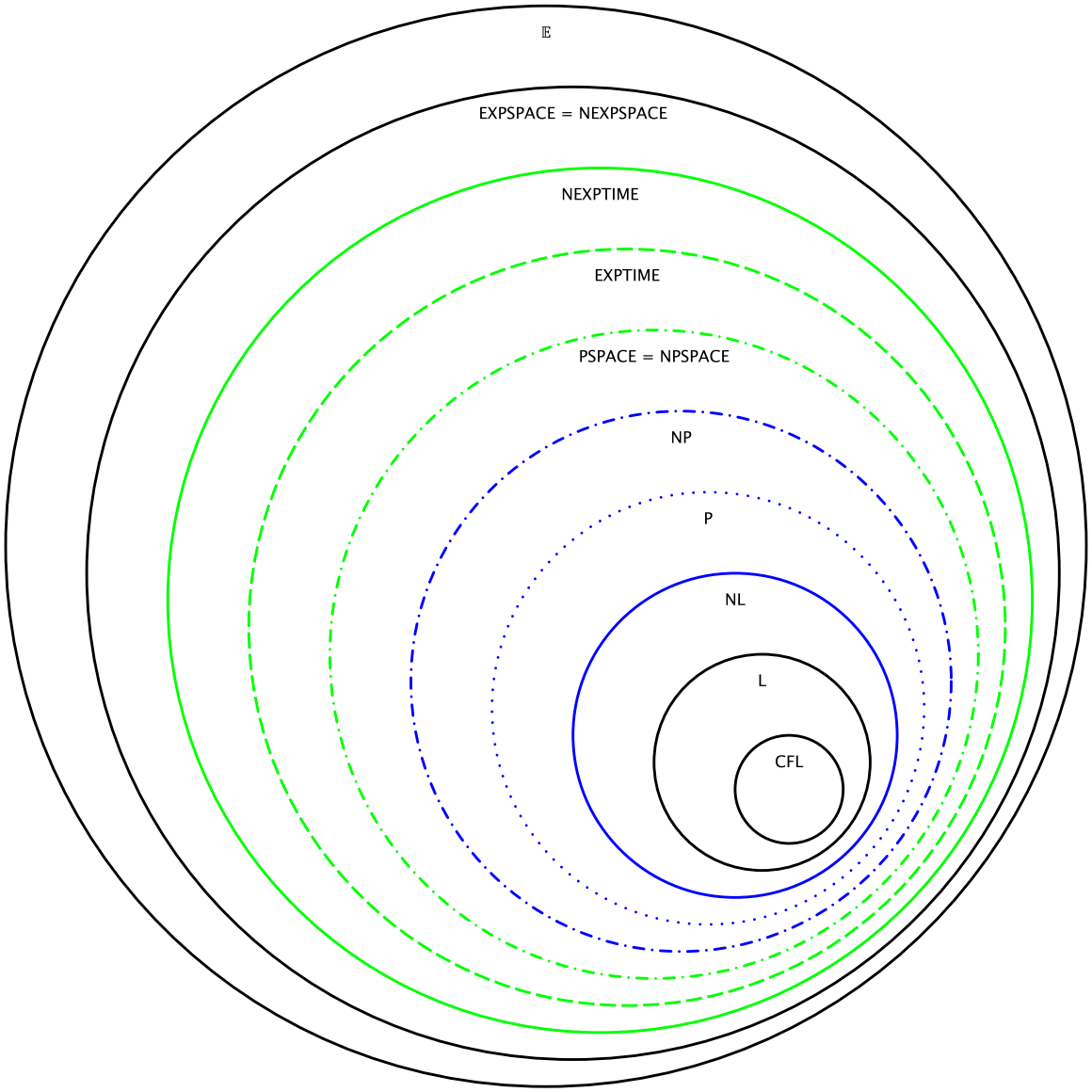

Abbildung 2.4: Komplexitätsklassen

Aufgrund echter Inklusionen müssen mindestens folgende Linien existieren:

- Eine der blauen Linien (——) zwischen $\mathcal{NL}$ und $\mathcal{PSPACE}$
- Eine der grünen Linien (——) zwischen $\mathcal{PSPACE}$ und $\mathcal{EXPSPACE}$
- Eine der gepunkteten Linien (····· oder -·-·-·-) zwischen $\mathcal{P}$ und $\mathcal{EXPTIME}$
- Eine der gestrichelten Linien (– – – – – oder -·-·-·-) zwischen $\mathcal{NP}$ und $\mathcal{NEXPTIME}$

2.3 Zeitkomplexitätsklassen

Zur Klassifizierung von Problemen kann man verschiedene Ressourcen beschränken. Zunächst wollen wir einer Turingmaschine nur noch begrenzte Zeit zur Verfügung stellen und herausfinden, welche Probleme dann noch gelöst werden können.

Um über konkrete Zeitkomplexitätsklassen sprechen zu können, müssen wir zunächst definieren, was wir unter der Laufzeit einer deterministischen bzw. nichtdeterministischen Turingmaschine verstehen.

Definition 2.14. Sei M eine deterministische Turingmaschine. Dann gibt

$$T_M : \Sigma^* \longrightarrow \mathbb{N}, w \longmapsto T_M(w)$$

die Anzahl der Schritte von M bei Eingabe w an.

Definition 2.15. Sei M eine deterministische Turingmaschine. Dann gibt

$$t_M : \mathbb{N} \longrightarrow \mathbb{N}, n \longmapsto \max_{w \in \Sigma^n} T_M(w)$$

die *Laufzeit* von M an.

Definition 2.16. Sei $f : \mathbb{N} \longrightarrow \mathbb{R}^+$, dann bezeichnet

$$\mathcal{TIME}(f(n)) \coloneqq \{L \mid \exists \,\mathrm{TM}\, M \text{ mit } t_M(n) \in \mathcal{O}(f(n)) \text{ und } M \text{ entscheidet } L\}$$

die Zeitkomplexitätsklasse von $f(n)$.

Neben dem Namen $\mathcal{TIME}(f(n))$ gibt es auch die Bezeichnung $\mathcal{DTIME}(f(n))$.

Definition 2.17. Sei N eine nichtdeterministische Turingmaschine. Dann gibt

$$T_N^{ndet} : \Sigma^* \longrightarrow \mathbb{N}, w \longmapsto T_N^{ndet}(w)$$

die Höhe des Berechnungsbaumes von N bei Eingabe w an.

Definition 2.18. Sei N eine nichtdeterministische Turingmaschine. Dann gibt

$$t_N^{ndet} : \mathbb{N} \longrightarrow \mathbb{N}, n \longmapsto \max_{w \in \Sigma^n} T_N^{ndet}(w)$$

die *Laufzeit* von N an.

Definition 2.19. Sei $f : \mathbb{N} \longrightarrow \mathbb{R}^+$, dann bezeichnet

$$\mathcal{NTIME}(f(n)) \coloneqq \left\{L \mid \exists \,\mathrm{NTM}\, N \text{ mit } t_N^{ndet}(n) \in \mathcal{O}(f(n)) \text{ und } N \text{ entscheidet } L\right\}$$

die nichtdeterministische Zeitkomplexitätsklasse von $f(n)$.

Nachfolgend konkretisieren wir die Klassen $\mathcal{TIME}(f(n))$ und $\mathcal{NTIME}(f(n))$, erhalten somit die ersten der in den Abbildungen 2.4 und 2.3 dargestellten Komplexitätsklassen und ordnen ihnen einige Sprachen zu.

Definition 2.20. Die Zeitkomplexitätsklasse $\mathcal{P}$ umfasst alle Probleme, die in polynomieller Zeit von einer deterministischen Turingmaschine entschieden werden können, also

$$\mathcal{P} := \bigcup_{k \in \mathbb{N}} \mathcal{TIME}\left(n^k\right).$$

Neben dem Namen $\mathcal{P}$ gibt es auch die Bezeichnung $\mathcal{PTIME}$. Probleme, die in $\mathcal{P}$ liegen, gelten gemeinhin als effizient lösbar.

Beispiel 2.21. Das positiv–singuläre Erfüllbarkeitsproblem der Aussagenlogik, kurz HORNSAT, nach dem US–amerikanischen Mathematiker *Alfred Horn* und dem Englischen *satisfiability*, liegt in der Zeitkomplexitätsklasse $\mathcal{P}$. Es behandelt die Frage, ob eine Formel, die in konjunktiver Normalform vorliegt und bei der jede Klausel höchstens ein positives Literal enthält, erfüllbar ist.

Notation 2.22. Im folgenden symbolisiert 0 den Wahrheitswert `falsch` und 1 den Wahrheitswert `wahr`.

Vorüberlegung 2.23. Eine Klausel, die höchstens ein positives Literal enthält, nennen wir im Folgenden Horn–Klausel. Da die Reihenfolge der Literale innerhalb einer Horn–Klausel ohne Probleme verändert werden kann, betrachten wir drei Typen von Horn–Klauseln.

Typ I x_1

Typ II $x_1 \vee \neg x_2 \vee \ldots \vee \neg x_k \longleftrightarrow x_1 \vee \neg(x_2 \wedge \ldots \wedge x_k) \longleftrightarrow [x_2 \wedge \ldots \wedge x_k \longrightarrow x_1]$

Typ III $\neg x_1 \vee \ldots \vee \neg x_k \longleftrightarrow \neg(x_2 \wedge \ldots \wedge x_k) \longleftrightarrow [x_2 \wedge \ldots \wedge x_k \longrightarrow 0]$

Horn–Klauseln vom Typ I bestehen ausschließlich aus einem positiven Literal, Horn–Klauseln vom Typ II bestehen aus einem positiven Literal und mindestens einem negativen Literal und Horn–Klauseln vom Typ III bestehen aus keinem positiven Literal und mindestens einem negativen Literal.

Beweis des Beispiels 2.21. Sei $\Phi = \Psi_1 \wedge \ldots \wedge \Psi_m$ eine Formel mit Horn–Klauseln Ψ_j in den Variablen x_i.

Die Idee der Vorgehensweise ist, dass alle Variablen zunächst mit 0 belegt sind und dann solche, die zwingend wahr sein müssen mit 1 belegt werden. Zunächst müssen alle Variablen in Horn–Klauseln vom Typ I wahr sein. Anschließend wird wiederholend überprüft, ob eine Implikation mit wahrem Antezedens entstanden ist. Falls dies der Fall ist und es sich dabei um eine Horn–Klausel vom Typ III handelt, muss verworfen

werden, da die Implikation einen Widerspruch liefert. Sollte es sich stattdessen um eine Horn–Klausel vom Typ II handeln, folgt, dass auch der Sukzedens wahr sein muss.

Die deterministische Turingmaschine M, die HORNSAT in polynomieller Zeit entscheidet, arbeitet wie folgt:

Modulbeschreibung $M(\langle\Phi\rangle)$

1. Falls Φ eine Horn–Klausel $\Psi = x_1$ von Typ I enthält, ersetze alle Vorkommen von x_1 in Φ durch 1
2. Falls in Schritt 1 mindestens ein Literal in Φ ersetzt wurde, dann gehe wieder zu Schritt 1
3. Falls Φ eine Horn–Klausel $\Psi = \neg x_1 \vee \ldots \vee \neg x_k$ von Typ III enthält, in der alle Literale durch 1 ersetzt wurden, dann VERWERFE
4. Falls Φ eine Horn–Klausel $\Psi = x_1 \vee \neg x_2 \vee \ldots \vee \neg x_k$ von Typ II enthält, in der alle Literale bis auf x_1 durch 1 ersetzt wurden, dann ersetze alle Vorkommen von x_1 in Φ durch 1 und gehe zu Schritt 3
5. AKZEPTIERE

Die einzelnen Schritte können offensichtlich in polynomieller Zeit abgearbeitet werden. Da in den Schritten 1 und 4 jeweils mindestens ein Literal durch 1 ersetzt worden sein muss, erkennt die Turingmaschine HORNSAT in polynomieller Zeit.

□

Definition 2.24. Die Zeitkomplexitätsklasse $\mathcal{NP}$ umfasst alle Probleme, die in polynomieller Zeit von einer nichtdeterministischen Turingmaschine entschieden werden können, also

$$\mathcal{NP} := \bigcup_{k\in\mathbb{N}} \mathcal{NTIME}\left(n^k\right).$$

Neben dem Namen $\mathcal{NP}$ gibt es auch die Bezeichnung $\mathcal{NPTIME}$. Im Jahr 2000 listete das Clay Mathematics Institute sieben Millennium–Probleme auf und lobte für die Lösung der Probleme jeweils ein Preisgeld in Höhe von einer Millionen US–Dollar aus. Auch die Frage, ob $\mathcal{P} = \mathcal{NP}$ ist, gehört zu diesen Problemen und ist bislang ungelöst. Bisher konnte lediglich eines der Probleme gelöst werden.

Bemerkung 2.25. In [Sip12, Definition 7.19] wird eine alternative Definition von $\mathcal{NP}$ genannt, welche nach [Sip12, Theorem 7.20] äquivalent zu der hier angegebenen ist.

Beispiel 2.26 ([Sip12, Theorem 7.37])**.** Das Erfüllbarkeitsproblem der Aussagenlogik, kurz SAT, nach dem Englischen *satisfiability*, liegt in der Zeitkomplexitätsklasse $\mathcal{NP}$. Es behandelt die Frage, ob eine aussagenlogische Formel erfüllbar ist.

Beispiel 2.27 ([HMU11, Satz 10.13])**.** Das konjunkt–normale Erfüllbarkeitsproblem der Aussagenlogik, kurz CNF–SAT, nach dem Englischen *conjunctive normal form satisfiability*, liegt in der Zeitkomplexitätsklasse $\mathcal{NP}$. Es behandelt die Frage, ob eine Formel, die in konjunktiver Normalform vorliegt, erfüllbar ist.

Beispiel 2.28 ([Sip12, Theorem 7.42])**.** Das ternäre Erfüllbarkeitsproblem der Aussagenlogik, kurz 3–SAT, nach dem Englischen *satisfiability*, liegt in der Zeitkomplexitätsklasse $\mathcal{NP}$. Es behandelt die Frage, ob eine Formel, die in konjunktiver Normalform vorliegt und bei der jede Klausel aus maximal drei Literalen besteht, erfüllbar ist.

Definition 2.29. Die Zeitkomplexitätsklasse $\mathcal{EXPTIME}$ umfasst alle Probleme, die in exponentieller Zeit von einer deterministischen Turingmaschine entschieden werden können, also

$$\mathcal{EXPTIME} := \bigcup_{k\in\mathbb{N}} \mathcal{TIME}\left(2^{n^k}\right).$$

Neben dem Namen $\mathcal{EXPTIME}$ gibt es auch die Bezeichnung $\mathcal{EXP}$.

Definition 2.30. Die Zeitkomplexitätsklasse $\mathcal{NEXPTIME}$ umfasst alle Probleme, die in exponentieller Zeit von einer nichtdeterministischen Turingmaschine entschieden werden können, also

$$\mathcal{NEXPTIME} := \bigcup_{k\in\mathbb{N}} \mathcal{NTIME}\left(2^{n^k}\right).$$

Neben dem Namen $\mathcal{NEXPTIME}$ gibt es auch die Bezeichnung $\mathcal{NEXP}$.

2.4 Platzkomplexitätsklassen

Analog zum vorherigen Abschnitt 2.3 beschränken wir nun die Ressource Platz.

Um über konkrete Platzkomplexitätsklassen sprechen zu können, müssen wir zunächst definieren, was wir unter dem Speicherplatzbedarf einer deterministischen bzw. nichtdeterministischen Turingmaschine verstehen.

Definition 2.31. Sei M eine deterministische Turingmaschine. Dann gibt

$$S_M : \Sigma^* \longrightarrow \mathbb{N}, w \longmapsto S_M(w)$$

die Anzahl der von M bei Eingabe w beschriebenen Speicherplätze an.

Die Länge des Eingabewortes w wird bei der Anzahl der von M beschriebenen Speicherplätze nicht berücksichtigt. Also kann insbesondere $S_M(w) < |w|$ gelten.

Definition 2.32. Sei M eine deterministische Turingmaschine. Dann gibt

$$s_M : \mathbb{N} \longrightarrow \mathbb{N}, n \longmapsto \max_{w \in \Sigma^n} S_M(w)$$

den *Speicherplatzbedarf* von M an.

Definition 2.33. Sei $f : \mathbb{N} \longrightarrow \mathbb{R}^+$. Dann bezeichnet

$$\mathcal{SPACE}(f(n)) \coloneqq \{L \mid \exists\, \text{TM}\, M \text{ mit } s_M(n) \in \mathcal{O}(f(n)) \text{ und } M \text{ entscheidet } L\}$$

die Platzkomplexitätsklasse von $f(n)$.

Neben dem Namen $\mathcal{SPACE}(f(n))$ gibt es auch die Bezeichnung $\mathcal{DSPACE}(f(n))$.

Definition 2.34. Sei N eine nichtdeterministische Turingmaschine. Dann gibt

$$S_N^{ndet} : \Sigma^* \longrightarrow \mathbb{N}, w \longmapsto S_N^{ndet}(w)$$

die maximale Anzahl über alle Läufe im Berechnungsbaum der von N bei Eingabe w beschriebenen Speicherplätze an.

Auch hier wird die Länge des Eingabewortes w bei der Anzahl der von N beschriebenen Speicherplätze nicht berücksichtigt. Also kann insbesondere $S_N^{ndet}(w) < |w|$ gelten.

Definition 2.35. Sei N eine nichtdeterministische Turingmaschine. Dann gibt

$$s_N^{ndet} : \mathbb{N} \longrightarrow \mathbb{N}, n \longmapsto \max_{w \in \Sigma^n} S_N^{ndet}(w)$$

den *Speicherplatzbedarf* von N an.

Definition 2.36. Sei $f : \mathbb{N} \longrightarrow \mathbb{R}^+$, dann bezeichnet

$$\mathcal{NSPACE}(f(n)) \coloneqq \left\{L \mid \exists\, \text{NTM}\, N \text{ mit } s_N^{ndet}(n) \in \mathcal{O}(f(n)) \text{ und } N \text{ entscheidet } L\right\}$$

die nichtdeterministische Platzkomplexitätsklasse von $f(n)$.

Nachfolgend konkretisieren wir nun die Klassen $\mathcal{SPACE}(f(n))$ und $\mathcal{NSPACE}(f(n))$, erhalten somit die noch fehlenden der in den Abbildungen 2.4 und 2.3 dargestellten Komplexitätsklassen und ordnen auch ihnen einige Sprachen zu.

Definition 2.37. Die Platzkomplexitätsklasse $\mathcal{L}$ umfasst alle Probleme, die auf logarithmischem Platz von einer deterministischen Turingmaschine entschieden werden können, also

$$\mathcal{L} \coloneqq \mathcal{SPACE}(\log n).$$

Neben dem Namen $\mathcal{L}$ gibt es auch die Bezeichnung $\mathcal{LOGSPACE}$.

Beispiel 2.38 ([Rei04])**.** Das Erreichbarkeitsproblem in ungerichteten Graphen kurz USTCON, nach dem Englischen *undirected–s–t–Connectivity*, liegt in der Platzkomplexitätsklasse $\mathcal{L}$. Es behandelt die Frage, ob es in einem gegebenen ungerichteten Graphen einen Weg vom Knoten s zum Knoten t gibt.

Für den hier zitierten Beweis erhielt Omer Reingold 2005 den Grace Murray Hopper Award der Association for Computing Machinery. Der mit 35 000 US–Dollar dotierte Preis ist seit 1971 ausgeschrieben und wird an Computerexperten verliehen, die bis zum Alter von 35 Jahren einen bedeutenden Beitrag erbringen.

Beispiel 2.39. Das unäre Erfüllbarkeitsproblem der Aussagenlogik, kurz 1–SAT, nach dem Englischen *satisfiability*, liegt in der Platzkomplexitätsklasse $\mathcal{L}$. Es behandelt die Frage, ob eine Formel, die in konjunktiver Normalform vorliegt und bei der jede Klausel aus einem Literal besteht, erfüllbar ist.

Beweis. Die deterministische Turingmaschine, die 1–SAT entscheidet, speichert die einzelnen Literale jeweils nacheinander auf dem Band und prüft, ob eins der folgenden Literale die Negation des aktuellen Literals ist, falls ja verwirft die Turingmaschine. Hat die Turingmaschine auch das vorletzte Literal mit dem letzten Literal verglichen und nicht verworfen, so akzeptiert diese. Die angegebene Turingmaschine benötigt nur Platz um genau zwei Literale zu speichern, arbeitet also auf logarithmischem Platz.

□

Definition 2.40. Die Platzkomplexitätsklasse $\mathcal{NL}$ umfasst alle Probleme, die auf logarithmischem Platz von einer nichtdeterministischen Turingmaschine entschieden werden können, also

$$\mathcal{NL} := \mathcal{NSPACE}(\log n).$$

Neben dem Namen $\mathcal{NL}$ gibt es auch die Bezeichnung $\mathcal{NLOGSPACE}$.

Beispiel 2.41. Das Erreichbarkeitsproblem in gerichteten Graphen kurz STCON, nach dem Englischen *s–t–Connectivity*, liegt in der Platzkomplexitätsklasse $\mathcal{NL}$. Es behandelt die Frage, ob es in einem gegebenen gerichteten Graphen einen Weg vom Knoten s zum Knoten t gibt.

Beweis. Sei m die Anzahl der Knoten in dem gegebenen Graphen. Die nichtdeterministische Turingmaschine, die STCON entscheidet, speichert auf dem Band statt des gesamten Weges ausschließlich den aktuellen Knoten des gegebenen Graphen und zusätzlich einen Schrittzähler. Sollte der Knoten t gespeichert werden, akzeptiert die Turingmaschine; sollte der Schrittzähler den Wert m überschreiten, verwirft die Turingmaschine. Jeder Lauf benötigt somit nur logarithmischen Platz und es folgt STCON $\in \mathcal{NL}$.

□

Beispiel 2.42 ([Pap94, Corollary zu Theorem 9.1])**.** Das binäre Erfüllbarkeitsproblem der Aussagenlogik, kurz 2–SAT, nach dem Englischen *satisfiability*, liegt in der Platzkomplexitätsklasse $\mathcal{NL}$. Es behandelt die Frage, ob eine Formel, die in konjunktiver Normalform vorliegt und bei der jede Klausel aus maximal zwei Literalen besteht, erfüllbar ist.

Definition 2.43. Die Platzkomplexitätsklasse $\mathcal{PSPACE}$ umfasst alle Probleme, die auf polynomiellem Platz von einer deterministischen Turingmaschine entschieden werden können, also

$$\mathcal{PSPACE} := \bigcup_{k\in\mathbb{N}} \mathcal{SPACE}\left(n^k\right).$$

Beispiel 2.44. Das Wahrheitsproblem, kurz TQBF, nach dem Englischen *true–quantified–boolean–formula*, liegt in der Platzkomplexitätsklasse $\mathcal{PSPACE}$. Es behandelt die Frage, ob eine mit Quantoren versehene Formel ohne freie Variablen erfüllbar ist.

Bemerkung 2.45. Wir betrachten hier ausschließlich Formeln in Pränex–Normalform.

Notation 2.46. Sei Φ eine Formel, dann bezeichne $\Phi|_{x=b}$ die Formel, die entsteht nachdem man alle x in Φ durch b ersetzt hat.

Beweis des Beispiels 2.44. Die deterministische Turingmaschine M, die TQBF auf polynomiellem Platz entscheidet, arbeitet wie folgt:

Modulbeschreibung $M(\langle\Phi\rangle)$

---

1. Falls Φ von der Form $\exists x\,\Psi$ ist, rufe nacheinander $M(\langle\Psi|_{x=0}\rangle)$ und $M(\langle\Psi|_{x=1}\rangle)$ auf. Falls beide Aufrufe verwerfen, dann VERWERFE, sonst AKZEPTIERE.

2. Falls Φ von der Form $\forall x\,\Psi$ ist, rufe nacheinander $M(\langle\Psi|_{x=0}\rangle)$ und $M(\langle\Psi|_{x=1}\rangle)$ auf. Falls beide Aufrufe akzeptieren, dann AKZEPTIERE, sonst VERWERFE.

3. Werte Φ aus. Falls das Ergebnis der Auswertung 1 ist, dann AKZEPTIERE, sonst VERWERFE.

Offensichtlich wird TQBF von M entschieden. Sei m die Anzahl der Variablen in Φ. Da die Rekursionstiefe höchstens m beträgt und wir auf jeder Rekursionsebene lediglich den Wert einer Variablen speichern müssen, arbeitet M auf $\mathcal{O}(m)$, also insbesondere auf polynomiellem, Platz.

□

Definition 2.47. Die Platzkomplexitätsklasse $\mathcal{NPSPACE}$ umfasst alle Probleme, die auf polynomiellem Platz von einer nichtdeterministischen Turingmaschine entschieden werden können, also

$$\mathcal{NPSPACE} := \bigcup_{k\in\mathbb{N}} \mathcal{NSPACE}\left(n^k\right).$$

Wir werden später in Korollar 4.2 unter Verwendung des Satzes von Savitch feststellen, dass die Gleichheit $\mathcal{PSPACE} = \mathcal{NPSPACE}$ gilt.

Definition 2.48. Die Platzkomplexitätsklasse $\mathcal{EXPSPACE}$ umfasst alle Probleme, die auf exponentiellem Platz von einer deterministischen Turingmaschine entschieden werden können, also

$$\mathcal{EXPSPACE} := \bigcup_{k \in \mathbb{N}} \mathcal{SPACE}\left(2^{n^k}\right).$$

Definition 2.49. Die Platzkomplexitätsklasse $\mathcal{NEXPSPACE}$ umfasst alle Probleme, die auf exponentiellem Platz von einer nichtdeterministischen Turingmaschine entschieden werden können, also

$$\mathcal{NEXPSPACE} := \bigcup_{k \in \mathbb{N}} \mathcal{NSPACE}\left(2^{n^k}\right).$$

Wir werden später in Korollar 4.3 unter Verwendung des Satzes von Savitch feststellen, dass die Gleichheit $\mathcal{EXPSPACE} = \mathcal{NEXPSPACE}$ gilt.

2.5 Reduktionen, Schwere und Vollständigkeit

Definition 2.50. Eine Funktion $f : X \longrightarrow Y$ heißt *berechenbar* genau dann, wenn es eine Turingmaschine M gibt und gilt

1. $\forall x \in \mathrm{Def}(f) \subseteq X : M(\langle x \rangle)$akzeptiert und auf dem Ausgabeband steht$\langle f(x) \rangle$,
2. $\forall x \in X \setminus \mathrm{Def}(f) : M(\langle x \rangle)$verwirft oder zykelt.

Definition 2.51. Eine Funktion $f : X \longrightarrow Y$ heißt *total*, falls gilt

$$\forall x \in X \, \exists! y \in Y : f(x) = y.$$

Definition 2.52. Eine Funktion $f : X \longrightarrow Y$ heißt *polyzeit berechenbar* genau dann, wenn sie von einer Turingmaschine auf polynomiellem Platz berechnet wird.

Definition 2.53. Seien $L \subseteq \Sigma^*$ und $L' \subseteq \Sigma'^*$ Sprachen. Eine Funktion f heißt *polyzeit Reduktion von L auf L'* genau dann, wenn

1. f ist polyzeit berechenbar und total,
2. $\forall w \in \Sigma^* : w \in L \iff f(w) \in L'$.

Definition 2.54. Seien $L \subseteq \Sigma^*$ und $L' \subseteq \Sigma'^*$ Sprachen. L ist *polyzeit reduzierbar* auf L' genau dann, wenn es eine polyzeit Reduktion von L auf L' gibt. Man schreibt dann

$$L \preceq_p L'.$$

Definition 2.55. Eine Sprache L heißt $\mathcal{NP}$*–schwer* genau dann, wenn alle Sprachen aus $\mathcal{NP}$ polyzeit reduzierbar auf L sind, also

$$L \text{ ist } \mathcal{NP}\text{–schwer} \iff \forall L' \in \mathcal{NP} : L' \preceq_p L.$$

Definition 2.56. Eine Sprache L heißt $\mathcal{NP}$*–vollständig* genau dann, wenn sie selbst aus $\mathcal{NP}$ kommt und zusätzlich $\mathcal{NP}$–schwer ist, also

$$L \text{ ist } \mathcal{NP}\text{–vollständig} \iff L \in \mathcal{NP} \land L \text{ ist } \mathcal{NP}\text{–schwer}.$$

Definition 2.57. Die Menge der $\mathcal{NP}$–vollständigen Sprachen wird mit $\mathcal{NPC}$ bezeichnet, also

$$\mathcal{NPC} \coloneqq \{L \mid L \text{ ist } \mathcal{NP}\text{–vollständig}\}.$$

Beispiel 2.58 ([Sip12, Theorem 7.37])**.** Es gilt

$$\text{SAT} \in \mathcal{NPC}.$$

Beispiel 2.59 ([HMU11, Satz 10.13])**.** Es gilt

$$\text{CNF–SAT} \in \mathcal{NPC}.$$

Beispiel 2.60 ([Sip12, Theorem 7.42])**.** Es gilt

$$\text{3–SAT} \in \mathcal{NPC}.$$

Definition 2.61. Eine Sprache L heißt $\mathcal{PSPACE}$*–schwer* genau dann, wenn alle Sprachen aus $\mathcal{PSPACE}$ polyzeit reduzierbar auf L sind, also

$$L \text{ ist } \mathcal{PSPACE}\text{–schwer} \iff \forall L' \in \mathcal{PSPACE} : L' \preceq_p L.$$

Definition 2.62. Eine Sprache L heißt $\mathcal{PSPACE}$*–vollständig* genau dann, wenn sie selbst aus $\mathcal{PSPACE}$ kommt und zusätzlich $\mathcal{PSPACE}$–schwer ist, also

$$L \text{ ist } \mathcal{PSPACE}\text{–vollständig} \iff L \in \mathcal{PSPACE} \land L \text{ ist } \mathcal{PSPACE}\text{–schwer}.$$

Definition 2.63. Die Menge der $\mathcal{PSPACE}$–vollständigen Sprachen wird mit $\mathcal{PSPACEC}$ bezeichnet, also

$$\mathcal{PSPACEC} \coloneqq \{L \mid L \text{ ist } \mathcal{PSPACE}\text{–vollständig}\}.$$

Beispiel 2.64 ([Sip12, Theorem 8.9])**.** Es gilt

$$\text{TQBF} \in \mathcal{PSPACEC}.$$

Zwei weitere Beispiele für $\mathcal{PSPACE}$–vollständige Probleme sind PB3GG und GTTT, welche in den Definitionen 7.10 und 7.13 vorgestellt werden. Die Beweise finden sich anschließend in den Proposition 7.12 und Satz 7.19.

Definition 2.65. Eine Turingmaschine heißt *Logplatz–Transduktor*, falls sie aus

1. einem ausschließlich lesbaren Eingabeband,
2. einem les- und beschreibbaren Arbeitsband mit Bandinhalt stets in $\mathcal{O}(\log n)$ Platz,
3. einem ausschließlich beschreibbaren Ausgabeband besteht.

Definition 2.66. Eine Funktion $f : X \longrightarrow Y$ heißt *logplatz berechenbar* genau dann, wenn sie von einem Logplatz–Transduktor berechnet wird.

Definition 2.67. Seien $L \subseteq \Sigma^*$ und $L' \subseteq \Sigma'^*$ Sprachen. Eine Funktion f heißt *logplatz Reduktion von L auf L'* genau dann, wenn

1. f ist logplatz berechenbar und total,
2. $\forall w \in \Sigma^* : w \in L \iff f(w) \in L'$.

Definition 2.68. Seien $L \subseteq \Sigma^*$ und $L' \subseteq \Sigma'^*$ Sprachen. L ist *logplatz reduzierbar* auf L' genau dann, wenn es eine logplatz Reduktion von L auf L' gibt. Man schreibt dann

$$L \preceq_l L'.$$

Definition 2.69. Eine Sprache L heißt $\mathcal{NL}$*–schwer* genau dann, wenn alle Sprachen aus $\mathcal{NL}$ logplatz reduzierbar auf L sind, also

$$L \text{ ist } \mathcal{NL}\text{–schwer} \iff \forall L' \in \mathcal{NL} : L' \preceq_l L.$$

Definition 2.70. Eine Sprache L heißt $\mathcal{NL}$*–vollständig* genau dann, wenn sie selbst aus $\mathcal{NL}$ kommt und zusätzlich $\mathcal{NL}$–schwer ist, also

$$L \text{ ist } \mathcal{NL}\text{–vollständig} \iff L \in \mathcal{NL} \wedge L \text{ ist } \mathcal{NL}\text{–schwer}.$$

Definition 2.71. Die Menge der $\mathcal{NL}$–vollständigen Sprachen wird mit $\mathcal{NLC}$ bezeichnet, also

$$\mathcal{NLC} \coloneqq \{L \mid L \text{ ist } \mathcal{NL}\text{–vollständig}\}.$$

Beispiel 2.72 ([Pap94, Theorem 16.3])**.** Es gilt

$$2\text{–SAT} \in \mathcal{NLC}.$$

Ein weiteres Beispiel eines $\mathcal{NL}$–vollständigen Problems ist STCON. Der Beweis dazu findet sich in Proposition 5.1.

3 Erste Beziehungen

Wir wollen nun einige elementare Beziehungen bezüglich der zuvor eingeführten Komplexitätsklassen klären. Unterteilt in drei Abschnitte werden wir verschiedene Ergebnisse erzielen, welche uns diesem Ziel näher bringen.

3.1 Beziehungen der allgemeinen Komplexitätsklassen

Betrachten wir die allgemeinen Komplexitätsklassen $\mathcal{TIME}(f(n))$, $\mathcal{NTIME}(f(n))$, $\mathcal{SPACE}(f(n))$, $\mathcal{NSPACE}(f(n))$, so erhalten wir die folgenden vier Aussagen.

Proposition 3.1. Es gilt

$$\mathcal{TIME}(f(n)) \subseteq \mathcal{NTIME}(f(n)).$$

Beweis. Sei $L \in \mathcal{TIME}(f(n))$, dann existiert eine deterministische Turingmaschine, die L in $\mathcal{O}(f(n))$ Zeit entscheidet. Da jede deterministische Turingmaschine insbesondere eine nichtdeterministische Turingmaschine ist, folgt ohne weitere Bemühungen, dass $L \in \mathcal{NTIME}(f(n))$ ist. Also gilt $\mathcal{TIME}(f(n)) \subseteq \mathcal{NTIME}(f(n))$.

□

Proposition 3.2. Es gilt

$$\mathcal{SPACE}(f(n)) \subseteq \mathcal{NSPACE}(f(n)).$$

Beweis. Sei $L \in \mathcal{SPACE}(f(n))$, dann existiert eine deterministische Turingmaschine, die L auf $\mathcal{O}(f(n))$ Platz entscheidet. Da jede deterministische Turingmaschine insbesondere eine nichtdeterministische Turingmaschine ist, folgt ohne weitere Bemühungen, dass $L \in \mathcal{NSPACE}(f(n))$ ist. Also gilt $\mathcal{SPACE}(f(n)) \subseteq \mathcal{NSPACE}(f(n))$.

□

Proposition 3.3. Es gilt

$$\mathcal{TIME}(f(n)) \subseteq \mathcal{SPACE}(f(n)).$$

Beweis. Sei $L \in \mathcal{TIME}(f(n))$, dann existiert eine deterministische Turingmaschine, die L in $\mathcal{O}(f(n))$ Zeit entscheidet. Da jede deterministische Turingmaschine, die in $\mathcal{O}(f(n))$ Zeit arbeitet, nicht mehr als $\mathcal{O}(f(n))$ Platz beschreiben kann, gilt also, dass $L \in \mathcal{SPACE}(f(n))$ ist. Also folgt $\mathcal{TIME}(f(n)) \subseteq \mathcal{SPACE}(f(n))$.

□

Proposition 3.4. Es gilt

$$\mathcal{NTIME}(f(n)) \subseteq \mathcal{NSPACE}(f(n)).$$

Beweis. Sei $L \in \mathcal{NTIME}(f(n))$, dann existiert eine nichtdeterministische Turingmaschine, die L in $\mathcal{O}(f(n))$ Zeit entscheidet. Da jede nichtdeterministische Turingmaschine, die in $\mathcal{O}(f(n))$ Zeit arbeitet, nicht mehr als $\mathcal{O}(f(n))$ Platz beschreiben kann, gilt also, dass $L \in \mathcal{NSPACE}(f(n))$ ist. Also folgt $\mathcal{NTIME}(f(n)) \subseteq \mathcal{NSPACE}(f(n))$.

□

3.2 Einordnung der Komplexitätsklassen

Als nächstes beschäftigen wir uns mit der Einordnung der kennengelernten Komplexitätsklassen in die uns bereits bekannten Klassen. Dazu betrachten wir die Beziehung zwischen $\mathcal{EXPSPACE}$ und $\mathbb{E}$, sowie die zwischen $\mathcal{L}$ und CFL.

Proposition 3.5. Es gilt

$$\mathcal{EXPSPACE} \subseteq \mathbb{E}.$$

Beweis. Sei $L \in \mathcal{EXPSPACE}$, dann existiert eine deterministische Turingmaschine, die L auf exponentiellem Platz entscheidet und es folgt sofort, dass $L \in \mathbb{E}$ ist. Also gilt $\mathcal{EXPSPACE} \subseteq \mathbb{E}$.

□

Der folgende Satz liefert eine bedeutende Aussage, sein Beweis würde jedoch den Rahmen dieser Arbeit überschreiten.

Satz 3.6 ([HU69, Theorem 11.3])**.** Es gilt

$$\text{CFL} \subseteq \mathcal{L}.$$

3.3 Beziehungen der vollständigen Klassen

Zuletzt ordnen wir noch die Klassen $\mathcal{NPC}$ und $\mathcal{NLC}$ ein und verdeutlichen uns ihre Bedeutungen für die offenen Fragen, ob $\mathcal{P} = \mathcal{NP}$ oder $\mathcal{L} = \mathcal{NL}$ gilt.

Lemma 3.7. Es gilt

$$\mathcal{NPC} \subseteq \mathcal{NP}.$$

Beweis. Sei $L \in \mathcal{NPC}$, per Definition 2.57 folgt sofort, dass $L \in \mathcal{NP}$ ist. Also gilt $\mathcal{NPC} \subseteq \mathcal{NP}$.

□

Satz 3.8 ([Sip12, Theorem 7.31])**.** Seien L und L' Sprachen. Dann gilt

$$L' \preceq_p L \wedge L \in \mathcal{P} \Longrightarrow L' \in \mathcal{P}.$$

Proposition 3.9 ([Sip12, Theorem 7.36])**.** Seien L und L' Sprachen. Dann gilt

$$L \in \mathcal{NPC} \wedge L \preceq_p L' \wedge L' \in \mathcal{NP} \Longrightarrow L' \in \mathcal{NPC}.$$

Satz 3.10. Sei L eine Sprache. Dann gilt

$$L \in \mathcal{NPC} \ \wedge \ L \in \mathcal{P} \Longrightarrow \mathcal{P} = \mathcal{NP}.$$

Beweis. Sei $L' \in \mathcal{NP}$. Da $L \in \mathcal{NPC}$ ist, gilt nach Definition 2.57 jetzt $L' \preceq_p L$. Da nun zusätzlich $L \in \mathcal{P}$ ist, folgt aus Satz 3.8, dass $L' \in \mathcal{P}$ ist. Also gilt $\mathcal{NP} \subseteq \mathcal{P}$ und zusammen mit $\mathcal{P} \subseteq \mathcal{NP}$ aus Proposition 3.1 folgt somit $\mathcal{P} = \mathcal{NP}$.

□

Lemma 3.11. Es gilt

$$\mathcal{PSPACEC} \subseteq \mathcal{PSPACE}.$$

Beweis. Sei $L \in \mathcal{PSPACEC}$, per Definition 2.63 folgt sofort, dass $L \in \mathcal{PSPACE}$ ist. Also gilt $\mathcal{PSPACEC} \subseteq \mathcal{PSPACE}$.

□

Satz 3.12. Seien L und L' Sprachen. Dann gilt

$$L' \preceq_p L \wedge L \in \mathcal{PSPACE} \Longrightarrow L' \in \mathcal{PSPACE}.$$

Beweis. Die Turingmaschine, die L' in polynomieller Zeit auf L reduziert, arbeitet nach Proposition 3.3 auf polynomiellem Platz. Da $L \in \mathcal{PSPACE}$ ist, existiert eine Turingmaschine, die L auf polynomiellem Platz entscheidet. Eine Turingmaschine, die die beiden zuvor genannten Turingmaschinen hintereinander ausführt, arbeitet offensichtlich ebenfalls auf polynomiellem Platz. Also gilt $L' \in \mathcal{PSPACE}$.

□

Proposition 3.13 ([GJ79, Lemma 2.2])**.** Seien L und L' Sprachen. Dann gilt

$$L \in \mathcal{PSPACEC} \wedge L \preceq_p L' \wedge L' \in \mathcal{PSPACE} \Longrightarrow L' \in \mathcal{PSPACEC}.$$

Satz 3.14. Sei L eine Sprache. Dann gilt

$$L \in \mathcal{PSPACEC} \ \wedge \ L \in \mathcal{P} \Longrightarrow \mathcal{P} = \mathcal{PSPACE}.$$

Beweis. Sei $L' \in \mathcal{PSPACE}$. Da $L \in \mathcal{PSPACEC}$ ist, gilt nach Definition 2.63 jetzt $L' \preceq_p L$. Da nun zusätzlich $L \in \mathcal{P}$ ist, folgt aus Satz 3.8, dass $L' \in \mathcal{P}$ ist. Also gilt $\mathcal{PSPACE} \subseteq \mathcal{P}$ und zusammen mit $\mathcal{P} \subseteq \mathcal{PSPACE}$ aus Proposition 3.3 folgt somit $\mathcal{P} = \mathcal{PSPACE}$.

□

Mit Satz 3.14 wird klar, warum wir in Definition 2.61 nicht erlaubt haben, dass die Sprachen aus $\mathcal{PSPACE}$ auf polynomiellem Platz auf eine Sprache, für die $\mathcal{PSPACE}$–Schwere gezeigt werden soll, reduzierbar sind.

Lemma 3.15. Es gilt

$$\mathcal{NLC} \subseteq \mathcal{NL}.$$

Beweis. Sei $L \in \mathcal{NLC}$, per Definition 2.71 folgt sofort, dass $L \in \mathcal{NL}$ ist. Also gilt $\mathcal{NLC} \subseteq \mathcal{NL}$.

□

Satz 3.16 ([Sip12, Theorem 8.23])**.** Seien L und L' Sprachen. Dann gilt

$$L \preceq_l L' \wedge L' \in \mathcal{L} \Longrightarrow L \in \mathcal{L}.$$

Proposition 3.17 ([Pap94, Proposition 8.2])**.** Seien L und L' Sprachen. Dann gilt

$$L \in \mathcal{NLC} \wedge L \preceq_l L' \wedge L' \in \mathcal{NL} \Longrightarrow L' \in \mathcal{NLC}.$$

Satz 3.18. Sei L eine Sprache. Dann gilt

$$L \in \mathcal{NLC} \ \wedge \ L \in \mathcal{L} \Longrightarrow \mathcal{L} = \mathcal{NL}.$$

Beweis. Sei $L' \in \mathcal{NL}$. Da $L \in \mathcal{NLC}$ ist, gilt nach Definition 2.71 jetzt $L' \preceq_l L$. Da nun zusätzlich $L \in \mathcal{L}$ ist, folgt aus Satz 3.16, dass $L' \in \mathcal{L}$ ist. Also gilt $\mathcal{NL} \subseteq \mathcal{L}$ und zusammen mit $\mathcal{L} \subseteq \mathcal{NL}$ aus Proposition 3.2 folgt somit $\mathcal{L} = \mathcal{NL}$.

□

4 Spart Nichtdeterminismus Platz?

In diesem Kapitel wollen wir uns mit der Frage beschäftigen, ob eine nichtdeterministische Turingmaschine weniger Speicherplatz benötigt als eine deterministische und falls ja, wie groß diese Speicherplatzersparnis ist.

Dazu betrachten wir zunächst den Algorithmus canYield, der die Frage beantwortet, ob eine nichtdeterministische Turingmaschine in einer beschränken Anzahl an Schritten von einer ersten gegebenen Konfiguration zu einer zweiten gelangen kann. Daraufhin führen wir den Satz von Savitch ein und verwenden im Beweis ebendiesen Algorithmus. Danach werden wir in der Lage sein, sowohl die Gleichheit von $\mathcal{PSPACE}$ und $\mathcal{NPSPACE}$, als auch von $\mathcal{EXPSPACE}$ und $\mathcal{NEXPSPACE}$ zu zeigen.

4.1 Beschränktes Erreichbarkeitsproblem

Wie bereits angekündigt interessiert uns zunächst, ob eine nichtdeterministische Turingmaschine N mit Eingabe w von Konfiguration c_1 ausgehend in maximal 2^t Schritten die Konfiguration c_2 erreichen kann.

Algorithmus $\text{canYield}_{(N,w)}(c_1, c_2, 2^t)$

```
if t = 0 then
  if c_1 = c_2 or N mit Eingabe w erreicht c_2 von c_1 in einem Schritt then
    AKZEPTIERE
  else
    VERWERFE
  end if
else
  for all Konfigurationen c_m von N mit Eingabe w do
    if canYield_(N,w)(c_1, c_m, 2^(t-1)) and canYield_(N,w)(c_m, c_2, 2^(t-1)) akzeptieren then
      AKZEPTIERE
    end if
  end for
  VERWERFE
end if
```

4.2 Satz von Savitch

Satz 4.1 (Savitch)**.** Sei $f : \mathbb{N} \longrightarrow \mathbb{R}^+$, mit $f(n) \geq n$. Dann gilt:

$$\mathcal{NSPACE}(f(n)) \subseteq \mathcal{SPACE}\big(f^2(n)\big)$$

Beweis. Sei N eine nichtdeterministische Turingmaschine, welche eine Sprache L auf $f(n)$ Platz entscheidet und sei c_{start} die Startkonfiguration von N bei Eingaben w. Modifiziere nun N, indem folgende Schritte ausgeführt werden:

1. Markiere die aktuelle Bandzelle
2. Simuliere $N(w)$
3. Falls $N(w)$ akzeptiert, dann
   i) Lösche das Band bis auf die Markierung aus Schritt 1
   ii) Bewege den Kopf zur markierten Bandzelle
   iii) Lösche die Markierung

Bezeichne die Konfiguration, in der sich N jetzt befindet, mit c_{accept}. Und definiere anschließend die deterministische Turingmaschine M wie folgt:

Modulbeschreibung $M(w)$

1. Setze $i \coloneqq 0$

2. Berechne $\text{canYield}_{(N,w)}(c_{start}, c_{accept}, 2^i)$

3. Falls die Berechnung in Schritt 2 akzeptiert, dann AKZEPTIERE

4. Berechne $\text{canYield}_{(N,w)}(c_{start}, c_*, 2^i)$ für jede Konfiguration c_* mit $|\langle c_* \rangle| = i + 1$

5. Falls eine Berechnung in Schritt 4 akzeptiert, dann setze $i \coloneqq i + 1$ und gehe zu Schritt 2

6. VERWERFE

Die deterministische Turingmaschine M simuliert die nichtdeterministische Turingmaschine N korrekt. Die Turingmaschine M verwirft spätestens, sobald $i = f(n)$ ist, da die Turingmaschine N, die auf $f(n)$ Platz arbeitet, offensichtlich keine Konfiguration c mit $|\langle c \rangle| = f(n) + 1$ erreichen kann. Akzeptiert M, so passiert ebenfalls spätestens im Fall $i = f(n)$. Das heißt, die Rekursionstiefe beträgt höchstens $f(n) + 1$, wobei für eine einzelne Rekursionsebene gilt, dass sie in $\mathcal{O}(f(n))$ liegt. Somit folgt insgesamt, dass der von M benötigte Platz in $\mathcal{O}((f(n) + 1) * f(n)) = \mathcal{O}(f^2(n))$ liegt.

□

Betrachtet man den Satz von Savitch für Polynome, so ergibt sich folgendes Korollar.

Korollar 4.2. Es gilt

$$\mathcal{PSPACE} = \mathcal{NPSPACE}.$$

Beweis. Nach Proposition 3.2 gilt, dass $\mathcal{PSPACE} \subseteq \mathcal{NPSPACE}$ ist. Aus dem Satz von Savitch 4.1 folgt außerdem $\mathcal{NPSPACE} \subseteq \mathcal{PSPACE}$. Also gilt insgesamt die Gleichheit $\mathcal{PSPACE} = \mathcal{NPSPACE}$.

□

Betrachtet man den Satz von Savitch hingegen für Exponentialfunktionen, so erhält man analog das folgende Korollar.

Korollar 4.3. Es gilt

$$\mathcal{EXPSPACE} = \mathcal{NEXPSPACE}.$$

Beweis. Nach Proposition 3.2 gilt, dass $\mathcal{EXPSPACE} \subseteq \mathcal{NEXPSPACE}$ ist. Aus dem Satz von Savitch 4.1 folgt außerdem $\mathcal{NEXPSPACE} \subseteq \mathcal{EXPSPACE}$. Also gilt insgesamt die Gleichheit $\mathcal{EXPSPACE} = \mathcal{NEXPSPACE}$.

□

5 Logplatz ist effizient lösbar

Ziel dieses Kapitels ist es, festzustellen, dass alle Probleme aus der Platzkomplexitätsklasse $\mathcal{NL}$, also gemäß Proposition 3.2 auch alle Probleme aus der Platzkomplexitätsklasse $\mathcal{L}$, in der Zeitkomplexitätsklasse $\mathcal{P}$ enthalten sind und somit als effizient lösbar gelten.

Um dies zu erreichen, betrachten wir das aus Beispiel 2.41 bekannte Problem STCON näher.

Proposition 5.1. Es gilt

$$\text{STCON} \in \mathcal{NLC}.$$

Beweis. Wir haben bereits in Beispiel 2.41 gesehen, dass STCON $\in \mathcal{NL}$ gilt. Um die Aussage zu beweisen, müssen wir also noch zeigen, dass STCON auch $\mathcal{NL}$–schwer ist.

Sei $L \in \mathcal{NL}$, N die nichtdeterministische Turingmaschine, die L auf logarithmischem Platz entscheidet und k eine passend gewählte und feste Konstante. Die Idee der logplatz Reduktion ist, dass eine Eingabe w in eine modifizierte Variante des Berechnungsbaumes von $N(w)$ überführt wird. Bei dieser Variante werden ausgehende Kanten von allen Blättern, die normalerweise akzeptierende Konfigurationen repräsentieren, zu einem neuen eindeutigen Blatt t, welches dann als einziges Blatt eine akzeptierende Konfiguration repräsentiert, ergänzt. Des Weiteren werden identische Knoten zusammengelegt. Bezeichne s die Wurzel des so entstandenen Graphen. Wir sind fertig, denn $N(w)$ akzeptiert genau dann, wenn es einen Pfad vom Knoten s zum Knoten t gibt.

Konkret gibt der Logplatz–Transduktor bei Eingabe w die Beschreibung des Graphen G aus. Diese besteht aus der Liste der Knoten, gefolgt von der Liste der Kanten. N kann bei Eingabe w nur endlich viele Konfigurationen annehmen. Also hat der Graph nur endlich viele Knoten und somit auch nur endlich viele Kanten. Es ist leicht diese jeweils auf $k * \log n$ Platz zu kodieren. Der Logplatz–Transduktor geht also der Reihe nach alle möglichen Kodierungen der Länge $k * \log n$ durch und schreibt jene auf sein Ausgabeband, welche eine legale Konfiguration von $N(w)$ beschreiben. Danach geht der Logplatz–Transduktor erneut der Reihe nach alle möglichen Kodierungen der Länge $k * \log n$ durch und testet, ob es sich um eine auszugebende Kante handelt. Da $N(w)$ auf logarithmischem Platz arbeitet, kann mithilfe ihrer Übergangsfunktion auf logarithmischem Platz überprüft werden, ob eine Konfiguration c_1 in eine Konfiguration c_2 überführt werden kann. Falls ja, schreibt der Logplatz–Transduktor die entsprechende Kante auf sein Ausgabeband.

□

Lemma 5.2. Eine Turingmaschine, die auf $f(n)$ Platz arbeitet, arbeitet in $n * 2^{\mathcal{O}(f(n))}$ Zeit.

Beweis. Betrachten wir die Anzahl der Konfigurationen einer Turingmaschine M, die bei einer Eingabe der Länge n auf $f(n)$ Platz arbeitet. Bezeichne $q := |Q|$ die Anzahl der Zustände und $g := |\Gamma|$ die Anzahl der Zeichen im Arbeitsalphabet. Es folgt, dass es $g^{f(n)}$ mögliche Bandinhalte gibt.

Der Kopf der Turingmaschine kann sich an $f(n)$ unterschiedlichen Positionen befinden. Besitzt die Turingmaschine zusätzlich ein ausschließlich lesbares Eingabeband, so kann sich der Kopf hier an n unterschiedlichen Positionen befinden.

Wir stellen fest, dass die Anzahl der Konfigurationen also ohne separates Eingabeband $q * f(n) * g^{f(n)}$ und mit separatem Eingabeband $q * f(n) * n * g^{f(n)}$ beträgt.

Da M ein Entscheider ist und somit keine Konfiguration mehr als einmal auftritt, arbeitet M definitiv in $q * f(n) * n * g^{f(n)}$ Zeit, oder anders ausgedrückt in $n * 2^{\mathcal{O}(f(n))}$ Zeit.

□

Proposition 5.3. Es gilt

$$\text{STCON} \in \mathcal{P}.$$

Beweis. Die deterministische Turingmaschine M, die STCON in polynomieller Zeit entscheidet, arbeitet wie folgt:

Modulbeschreibung $M(\langle G, s, t\rangle)$

1. Markiere den Knoten s
2. Für alle Kanten (m, u) im gerichteten Graphen G, für die gilt, dass m ein markierter und u ein unmarkierter Knoten ist, markiere auch Knoten u
3. Falls in Schritt 2 ein neuer Knoten markiert wurde, gehe wieder zu Schritt 2
4. Falls der Knoten t markiert ist, dann AKZEPTIERE
5. VERWERFE

Schritte 1, 3, 4 und 5 können offensichtlich in polynomieller Zeit durchgeführt werden. Sei a die Anzahl der Kanten in G, dann werden in Schritt 2 jeweils a Kanten betrachtet. Schritte 2 und 3 werden außerdem höchstens $(a-1)$–mal ausgeführt, da bei jedem Durchlauf mindestens eine neue Kante markiert werden muss. Also wird STCON in polynomieller Zeit von M entschieden.

□

Satz 5.4. Es gilt

$$\mathcal{NL} \subseteq \mathcal{P}.$$

Beweis. Sei $L \in \mathcal{NL}$. In Proposition 5.1 haben wir bewiesen, dass STCON $\in \mathcal{NLC}$ ist. Also wissen wir, dass es eine logplatz Reduktion von L auf STCON gibt. Nach Lemma 5.2 ist bekannt, dass diese Reduktion bereits eine polyzeit Reduktion ist. Weiter wissen wir aus Proposition 5.3, dass STCON $\in \mathcal{P}$ gilt. Satz 3.8 sagt uns nun, dass $L \in \mathcal{P}$ ist. Also gilt insgesamt $\mathcal{NL} \subseteq \mathcal{P}$.

□

6 Mehr Platz bedeutet mehr Macht

Die Frage, die wir als nächstes behandeln wollen, ist, ob eine Turingmaschine mehr Probleme lösen kann, falls ihr mehr Platz zur Verfügung steht. Konkret werden wir mithilfe des Platzhierarchiesatzes die Ungleichheit von $\mathcal{NL}$, $\mathcal{PSPACE}$ und $\mathcal{EXPSPACE}$ beweisen.

Definition 6.1. Eine Funktion $f : \mathbb{N} \longrightarrow \mathbb{N}$ mit $f(n) \in \Omega(\log n)$ heißt *platzkonstruierbar*, falls eine deterministische Turingmaschine, die auf $\mathcal{O}(f(n))$ Platz arbeitet, existiert, die die Eingabe 1^n in die Ausgabe $1^{f(n)}$ überführt.

Bemerkung 6.2. Äquivalent dazu kann die deterministische Turingmaschine die Eingabe 1^n auch in eine andere, $f(n)$ repräsentierende, Ausgabe überführen, also statt einer unären zum Beispiel eine binäre, wie in [Sip12, Definition 9.1], oder dezimale Darstellung ausgeben.

Satz 6.3 (Platzhierarchiesatz)**.** Sei $f : \mathbb{N} \longrightarrow \mathbb{N}$ eine platzkonstruierbare Funktion. Dann gibt es eine Sprache L, die auf $\mathcal{O}(f(n))$, aber nicht auf $o(f(n))$ Platz entscheidbar ist.

Beweis. Betrachte folgende deterministische Turingmaschine:

Modulbeschreibung $D(w)$

---

1. Definiere $n \coloneqq |w|$
2. Markiere $f(n)$ viel Platz auf dem Band. Falls D in einem der folgenden Schritte versucht mehr Platz zu verwenden, dann VERWERFE
3. Falls keine TM M existiert, sodass $w = \langle M \rangle 10^*$ gilt, dann VERWERFE
4. Simuliere $M(w) = M(\langle M \rangle 10^*)$ und zähle die Anzahl der Schritte. Sobald die Anzahl $2^{f(n)}$ übersteigt, also insbesondere, falls M zykelt, VERWERFE
5. Falls M akzeptiert, dann VERWERFE, sonst AKZEPTIERE

Da jeder Schritt zeitlich beschränkt ist, ist D ein Entscheider. Bezeichne die Sprache, die von D entschieden wird, mit L. Da f platzkonstruierbar ist, garantiert uns Schritt 2, dass L auf $\mathcal{O}(f(n))$ Platz von D entschieden wird.

Da das Arbeitsalphabet Γ_M der simulierten Turingmaschine M aus beliebig vielen Zeichen bestehen kann, D jedoch ein festes Arbeitsalphabet Γ_D hat, wird jedes Zeichen aus Γ_M durch k Zeichen aus Γ_D kodiert. Die passend zu wählende Konstante k hängt hierbei nur von M ab. Diese Kodierung hat zur Folge, dass D für jede Zelle des Bandes von M selbst k Zellen benötigt, anders ausgedrückt, falls M auf $g(n)$ Platz arbeitet, so arbeitet D bei der Simulation von M auf $k * g(n)$ Platz.

Sei $g(n) \in o(f(n))$. Angenommen es existiert eine Turingmaschine M, welche L auf $g(n)$ Platz entscheidet. Dann kann D diese Turingmaschine M auf $k * g(n)$ Platz simulieren. Da $g(n) \in o(f(n))$ ist, muss eine Konstante n_0 existieren, sodass für alle $n \geq n_0$ gilt, dass $k * g(n) < f(n)$ ist.

Hat die Eingabe w also mindestens Länge n_0, so wird die Simulation nicht abgebrochen. Sei nun $w = \langle M \rangle 10^{n_0}$. Akzeptiert $M(w)$, dann verwirft $D(w)$ und es gilt $w \notin L$, verwirft $M(w)$ stattdessen, dann akzeptiert $D(w)$ und es gilt $w \in L$. Also folgt, dass M die Sprache L nicht entscheidet und somit L nicht auf $o(f(n))$ Platz entscheidbar ist.

□

Die beiden folgenden Korollare formulieren die erwähnten echten Inklusionen, welche wir jeweils direkt im Anschluss beweisen.

Korollar 6.4. Es gilt

$$\mathcal{NL} \subsetneq \mathcal{PSPACE}.$$

Beweis. Per Definition 2.40 ist $\mathcal{NL} = \mathcal{NSPACE}(\log n)$. Aus dem Satz von Savitch 4.1 folgt nun, dass $\mathcal{NSPACE}(\log n) \subseteq \mathcal{SPACE}\left(\log^2 n\right)$ ist. Zudem folgt aus dem Platzhierarchiesatz 6.3, dass $\mathcal{SPACE}\left(\log^2 n\right) \subsetneq \mathcal{SPACE}(n)$ ist. Da $\bigcup_{k \in \mathbb{N}} \mathcal{SPACE}\left(n^k\right) = \mathcal{PSPACE}$ nach Definition 2.43 gilt, folgt insgesamt $\mathcal{NL} \subsetneq \mathcal{PSPACE}$.

□

Korollar 6.5. Es gilt

$$\mathcal{PSPACE} \subsetneq \mathcal{EXPSPACE}.$$

Beweis. Per Definition 2.43 ist $\mathcal{PSPACE} = \bigcup_{k \in \mathbb{N}} \mathcal{SPACE}\left(n^k\right)$. Da für jedes $k \in \mathbb{N}$ gilt, dass $\mathcal{SPACE}\left(n^k\right) \subseteq \mathcal{SPACE}\left(n^{\log n}\right)$ ist, folgt $\bigcup_{k \in \mathbb{N}} \mathcal{SPACE}\left(n^k\right) \subseteq \mathcal{SPACE}\left(n^{\log n}\right)$. Mit dem Platzhierarchiesatz 6.3 gilt hier, dass $\mathcal{SPACE}\left(n^{\log n}\right) \subsetneq \mathcal{SPACE}(2^n)$ ist. Da nach Definition 2.48 gilt, dass $\bigcup_{k \in \mathbb{N}} \mathcal{SPACE}\left(2^{n^k}\right) = \mathcal{EXPSPACE}$ ist, folgt insgesamt $\mathcal{PSPACE} \subsetneq \mathcal{EXPSPACE}$.

□

7 Das Tic–Tac–Toe Problem

Dieses Kapitel widmet sich dem bekannten Spiel Tic–Tac–Toe, welches wir in in die Komplexitätsklasse $\mathcal{PSPACE}$ einordnen werden. Dazu verallgemeinern wir Tic–Tac–Toe und geben dann eine polyzeit Reduktion eines Problems, für welches bereits bekannt ist, dass es in $\mathcal{PSPACE}$ liegt, auf dieses verallgemeinerte Tic–Tac–Toe an. Unser Vorgehen führen wir in Analogie zu [Rei80] durch.

Bemerkung 7.1. Sprechen wir im Folgenden von Geographie, so meinen wir allgemein die Idee hinter Geographie. Sprechen wir hingegen von einem Geographiespiel, so meinen wir eine konkrete Spielsituation von Geographie, bei der eventuell bereits Züge gemacht sind.

Bemerkung 7.2. Sprechen wir im Folgenden von Tic–Tac–Toe, so meinen wir allgemein die Idee hinter Tic–Tac–Toe. Sprechen wir hingegen von einem Tic–Tac–Toe–Spiel, so meinen wir eine konkrete Spielsituation von Tic–Tac–Toe, bei der eventuell bereits Züge gemacht sind.

Bemerkung 7.3. Sagen wir, dass sich eine gewisse Anzahl an Symbolen in einer Reihe befindet, soll dies bedeuten, dass sich keine anderen Symbole zwischen diesen befinden.

Definition 7.4. Tritt eine Spielsituation ein, in der ein Spieler, unabhängig von den Zügen seines Gegners, sicher gewonnen hat, so sagt man, dass dieser Spieler eine *Gewinnstrategie* hat.

7.1 Standard Tic–Tac–Toe

Tic–Tac–Toe ist ein Strategiespiel für zwei Personen. Es wird klassischerweise auf einem Spielfeld mit 3×3 Feldern gespielt. Abwechselnd setzt der erste Spieler ein Kreuz und der zweite Spieler einen Kreis in ein freies Feld des Spielfelds, mit dem Ziel vor dem Gegner drei der eigenen Symbole waagerecht, senkrecht oder diagonal in einer Reihe zu haben. Gelingt dies keinem der beiden, so endet das Spiel in einem Unentschieden.

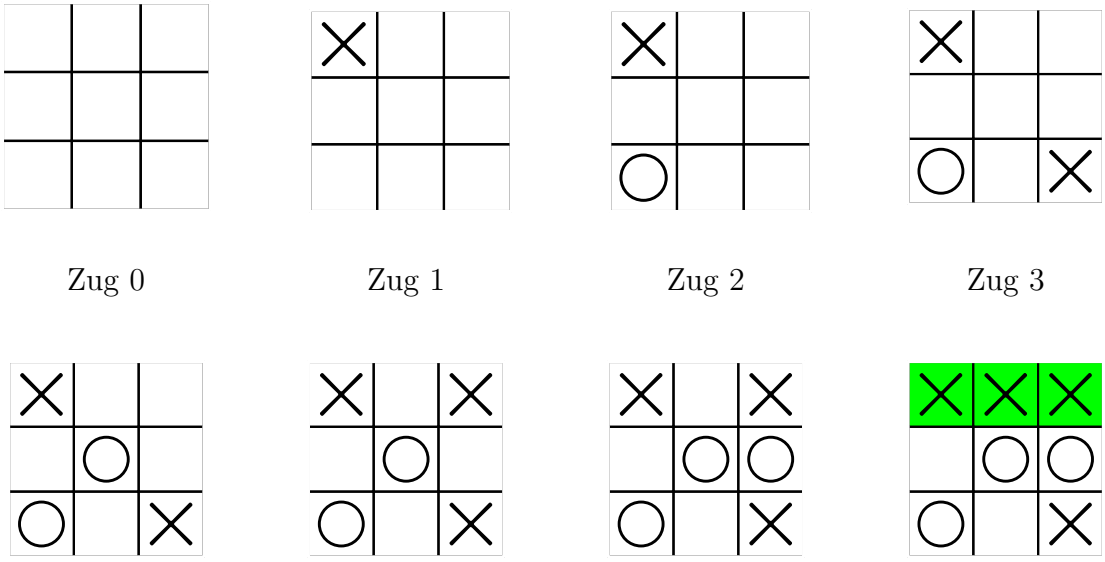

Abbildung 7.1: Beispielverlauf eines klassischen Tic–Tac–Toe–Spiels

Auf einem leeren Spielfeld hat der erste Spieler noch keine Gewinnstrategie. Verläuft das Tic–Tac–Toe–Spiel allerdings, wie in Abbildung 7.1 dargestellt, so hat der erste Spieler ab der entstandenen Situation Zug 2 eine Gewinnstrategie.

7.2 Tic–Tac–Toe auf größeren Spielfeldern

Spielt man Tic–Tac–Toe auf einem Spielfeld mit 4×4 oder mehr Feldern, wobei zum Sieg weiterhin drei eigene Symbole in einer Reihe benötigt werden, so hat der erste Spieler sofort von Beginn an eine Gewinnstrategie.

1	4	9	16	⋯
2	3	8	15	⋯
5	6	7	14	⋯
10	11	12	13	⋯
⋮	⋮	⋮	⋮	⋱

Abbildung 7.2: Nummeriertes Tic–Tac–Toe Spielfeld mit $k \times k$ Feldern für $k \geq 4$

Eine mögliche Gewinnstrategie für den ersten Spieler wäre mit einem Kreuz im Feld 3 zu beginnen, woraufhin der zweite Spieler einen Kreis in ein beliebiges freies Feld setzt.

Falls der zweite Spieler den Kreis in eines der Felder 4, 6 oder 11 setzt, so reagiert der erste Spieler mit einem Kreuz im Feld 8. Nach einem weiteren Zug des zweiten Spielers ist mindestens eines der Felder 2 und 15 noch frei, mit welchem der erste Spieler dann gewinnt.

Falls der zweite Spieler seinen ersten Kreis in keines der Felder 4, 6 oder 11 setzen sollte, so setzt der erste Spieler selbst sein zweites Kreuz in Feld 6. Nach einem weiteren Zug des zweiten Spielers ist diesmal mindestens eines der Felder 4 und 11 noch frei, mit welchem der erste Spieler dann gewinnt.

7.3 Standard Geographie

Geographie ist ein Spiel für zwei Personen, bei dem die beiden Spieler abwechselnd Städtenamen nennen. Es kann eingeschränkt werden, welche Städte zulässig sind. Beispielsweise kann nur mit Städten gespielt werden, die in Deutschland liegen. Es dürfen allerdings in jedem Fall keine Städte wiederholt werden und nur solche Städte genannt werden, die mit dem Buchstaben beginnen, mit dem die zuvor genannte Stadt endete. Das Spiel beginnt mit einer zuvor festgelegten Stadt und es verliert derjenige Spieler, der als erstes nicht mehr in der Lage ist eine weitere Stadt zu nennen. Ein möglicher Spielverlauf für ein Geographiespiel mit deutschen Städten beginnend mit Münster wäre also: Münster — Rostock — Karlsruhe — Essen — Norderney

Ein solches Geographiespiel kann in einem gerichteten Graphen dargestellt werden. Abbildung 7.3 zeigt ein Beispiel eines Spiels, bei dem alle zulässigen Städte bis auf Erfurt, Essen, Karlsruhe, Münster, Norderney, Recklinghausen, Rostock und Trier bereits in vorherigen Zügen genannt wurden und der zweite Spieler zuletzt eine auf m endende

Stadt genannt hat. Der erste Spieler hat auf diesem Geographiespiel eine Gewinnstrategie.

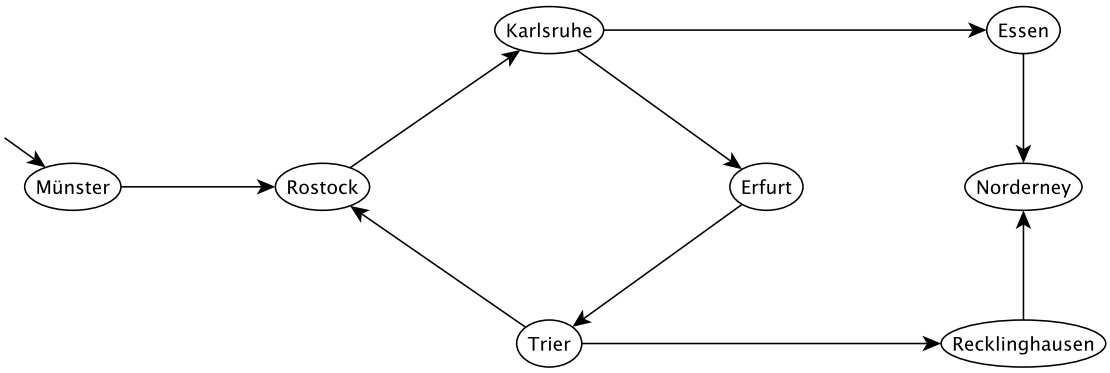

Abbildung 7.3: Beispiel eines Geographiespiels

7.4 Verallgemeinertes Geographie

Wir wollen nun Geographie verallgemeinern, um es in eine Komplexitätsklasse einordnen zu können. Also definieren wir die Probleme GG und PB3GG und verweisen auf zwei Beweise, die zeigen, dass GG und PB3GG in $\mathcal{PSPACE}$ liegen.

Das Geographiespiel aus Abbildung 7.3 verändern wir, indem wir die Knoten nicht mehr mit konkreten Städtenamen beschriften.

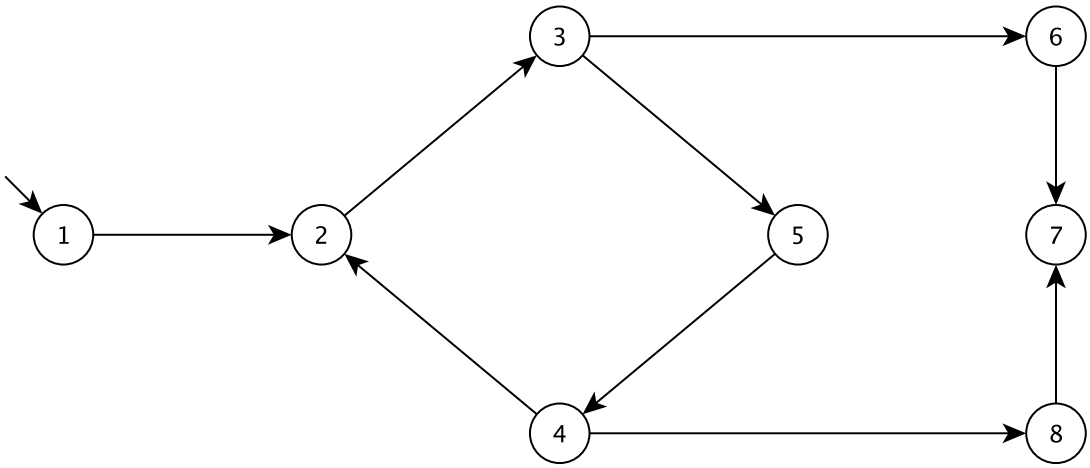

Abbildung 7.4: Verallgemeinertes Geographiespiel

Definition 7.5. Verallgemeinertes Geographie, kurz GG, nach dem Englischen *generalized geography*, wird auf einem gerichteten Graphen mit gekennzeichnetem Startknoten

gespielt, also

$$\mathrm{GG} \coloneqq \left\{ \langle G, b \rangle \,\middle|\, \begin{array}{l} \text{Der erste Spieler hat eine Gewinnstrategie für} \\ \text{das verallgemeinerte Geographiespiel auf dem} \\ \text{gerichteten Graphen } G \text{ mit Startknoten } b \end{array} \right\}.$$

Proposition 7.6 ([Sip12, Theorem 8.14])**.** Es gilt

$$\mathrm{GG} \in \mathcal{PSPACEC}.$$

Definition 7.7. Ein Graph heißt *planar*, falls er in einer Ebene dargestellt werden kann, sodass sich keine Kanten schneiden.

Definition 7.8. Ein Graph heißt *bipartit*, falls sich seine Knotenmenge in zwei disjunkte Teilmengen aufteilen lässt, sodass alle Kanten einen Knoten aus der einen Menge mit einem Knoten aus der anderen Menge verbinden.

Definition 7.9. Der *Grad eines Knoten* ist die Anzahl der Knoten, die über eine Kante mit diesem Knoten verbunden sind.

Definition 7.10. Planares bipartites dreigradiges verallgemeinertes Geographie, kurz PB3GG, nach dem Englischen *planar bipartite generalized geography*, wird auf einem gerichteten Graphen mit gekennzeichnetem Startknoten gespielt, also

$$\mathrm{PB3GG} \coloneqq \left\{ \langle G, b \rangle \,\middle|\, \begin{array}{l} \text{Der erste Spieler hat eine Gewinnstrategie für das} \\ \text{verallgemeinerte Geographiespiel auf dem planaren} \\ \text{bipartiten gerichteten Graphen } G, \text{ dessen Knoten} \\ \text{maximal Grad 3 haben, mit Startknoten } b \end{array} \right\}.$$

Notation 7.11. Sei $G = (V, E)$ ein planarer bipartiter Graph mit Startknoten b, dann sei (V_1, V_2) die Bipartition, für die gilt, dass $b \in V_1$ ist.

Notation 7.11 sagt uns, dass der erste Spieler nur auf Knoten aus V_1 und der zweite Spieler nur auf Knoten aus V_2 setzt.

Der Graph aus Abbildung 7.4 ist bereits planar und bipartit. Seine Knoten haben ebenfalls bereits maximal Grad 3. Es gilt $V_1 = \{1, 3, 4, 7\}$ und $V_2 = \{2, 5, 6, 8\}$.

Proposition 7.12 ([LS80, Corollary zu Proposition 1])**.** Es gilt

$$\mathrm{PB3GG} \in \mathcal{PSPACEC}.$$

7.5 Verallgemeinertes Tic–Tac–Toe

Wir wollen im Folgenden GTTT, eine verallgemeinerte Tic–Tac–Toe Variante, betrachten. Wir beweisen, unter anderem mit einer polyzeit Reduktion von PB3GG auf GTTT, dass GTTT selbst auch in $\mathcal{PSPACEC}$ liegt.

Definition 7.13. Verallgemeinertes Tic–Tac–Toe, kurz GTTT, nach dem Englischen *generalized tic–tac–toe*, wird auf einem Spielfeld mit $k \times k$ Feldern gespielt und zum Sieg werden fünf eigene Symbole in einer Reihe benötigt, also

$$\text{GTTT} \coloneqq \left\{ \langle S \rangle \,\middle|\, \begin{array}{l} \text{Der erste Spieler hat eine Gewinnstrategie} \\ \text{für das verallgemeinerte Tic–Tac–Toe–Spiel} \\ \text{auf dem Spielfeld } S \end{array} \right\}.$$

Notation 7.14. Sei S ein Spielfeld auf dem die, voneinander verschiedenen, Felder F und F' frei sind, dann bezeichne $S|_{F,F'}$ das Spielfeld, das entsteht indem der erste Spieler auf das Feld F und der zweite Spieler auf das Feld F' des Spielfelds S setzen.

Notation 7.15. Sei S ein Spielfeld auf dem das Felder F frei ist, dann bezeichne $S|_F$ das Spielfeld, das entsteht indem der erste Spieler auf das Feld F des Spielfelds S setzt.

Proposition 7.16. Es gilt

$$\text{GTTT} \in \mathcal{PSPACE}.$$

Vorüberlegung 7.17. Betrachtet man ein Spielfeld mit genau $k \times k$ Feldern, so können die Felder wie in Abbildung 7.2 nummeriert werden. Daraus ergibt sich, dass auch Paare von Feldern (F, F') geordnet werden können. Eine mögliche Ordnung ist:

$$(1,1), (1,2), \ldots, (1,k), (2,1), (2,2), \ldots, (2,k), \ldots, (k,1), (k,2), \ldots, (k,k)$$

Beweis der Proposition 7.16. Die deterministische Turingmaschine M, die GTTT auf polynomiellem Platz entscheidet, arbeitet wie folgt:

Modulbeschreibung $M(\langle S \rangle)$

1. Falls auf S fünf Kreuze in einer Reihe sind, dann AKZEPTIERE
2. Falls auf S fünf Kreise in einer Reihe sind, dann VERWERFE
3. Für alle Paare von freien Feldern (F, F') mit $F \neq F'$ rufe gemäß der Ordnung aus Vorüberlegung 7.17 nacheinander $M(\langle S|_{F,F'} \rangle)$ auf
4. Falls nur noch genau ein Feld F frei ist, dann rufe $M(\langle S|_F \rangle)$ auf
5. Falls für ein F aus Schritt 3 alle Aufrufe akzeptieren, dann AKZEPTIERE
6. Falls Schritt 4 akzeptiert, dann AKZEPTIERE
7. VERWERFE

Sei m die Anzahl freien Felder. Da die Rekursionstiefe höchstens $m * (m-1)$ beträgt und wir auf jeder Rekursionsebene lediglich die beiden ausgewählten Felder speichern müssen, arbeitet M auf polynomiellem Platz.

□

Lemma 7.18. Es gilt

$$\text{PB3GG} \preceq_p \text{GTTT}\,.$$

Beweis. Unser Vorgehen gliedert sich in zwei Teile. Zunächst überführen wir den gegebenen Graphen mit Startknoten in einen anderen Graphen mit Startknoten, sodass weitere Eigenschaften erfüllt sind. Den so erhaltenen Graphen überführen wir dann im Anschluss in ein Spielfeld.

Konstruiere zu dem gegebenen planaren bipartiten gerichteten Graphen $G = (V, E)$, dessen Knoten maximal Grad 3 haben, einen neuen Graphen $G' = (V', E')$, der zusätzlich folgende Bedingungen erfüllt:

1. G' kann wie folgt in den $\mathbb{R}^2$ eingebettet werden:
   i) $V' \subsetneq \mathbb{N}^2$, wobei $(i, j) \in V' \Longrightarrow i, j \leq |V'|$
   ii) Alle Kanten stehen waagerecht oder senkrecht:
   $((x_1, y_1), (x_2, y_2)) \in E' \Longrightarrow x_1 = x_2 \vee y_1 = y_2$
   iii) Alle Kanten haben entweder Länge 1 oder Länge 2:
   $((x_1, y_1), (x_2, y_2)) \in E' \Longrightarrow |x_1 - x_2| + |y_1 - y_2| \in \{1, 2\}$
   iv) Vom gleichen Knoten ausgehende Kanten stehen im Winkel von 180° zueinander:
   $\{((x_1, y_1), (x_2, y_2)), ((x_1, y_1), (x_3, y_3))\} \subseteq E' \Longrightarrow x_1 = x_2 = x_3 \vee y_1 = y_2 = y_3$
2. $|V'| \leq 9 * |V|^2$
3. Der Startknoten b und Knoten ohne ausgehende Kante haben genau Grad 1.
4. Der beginnende Spieler in einem Geographie Spiel auf G' hat genau dann eine Gewinnstrategie, wenn er eine auf G besitzt.

Dies kann erreicht werden, indem Kanten von G jeweils durch Kantenzüge, die aus einer ungeraden Anzahl von Kanten bestehen, ersetzt werden.

Damit der Graph aus Abbildung 7.4 den Bedingungen genügt, modifizieren wir ihn und erhalten auf diese Weise den Graphen, den wir anschließend in ein Tic–Tac–Toe–Spiel überführen wollen.

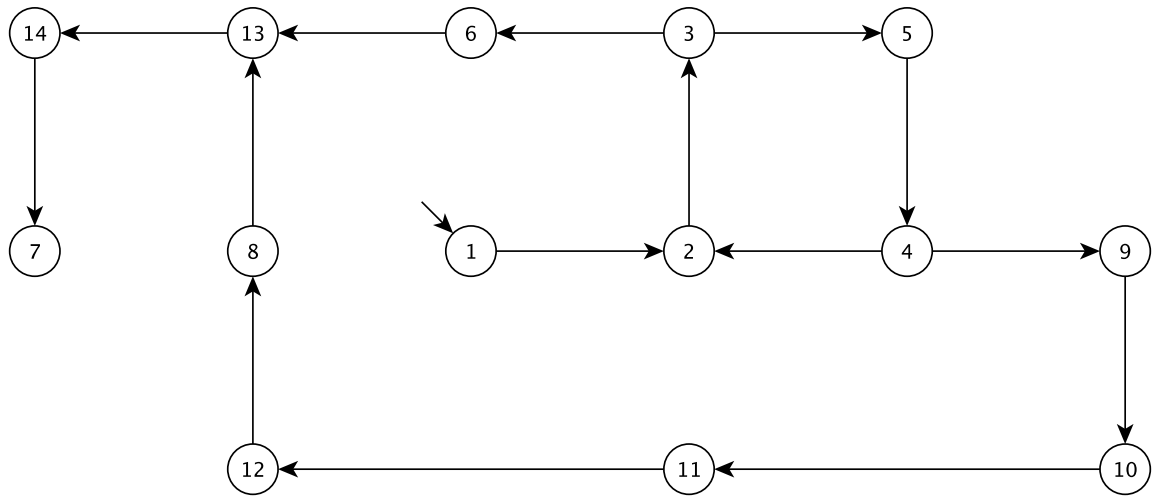

Abbildung 7.5: Modifiziertes Geographiespiel

Den erhaltenen Graphen $G' = (V', E')$ transformieren wir nun in ein Spielfeld der Größe $k \times k$, wobei $k = 30|V'|$ ist. Die Knoten $(i, j) \in V'$ ordnen wir den Feldern $(15i, 15j)$ zu. Kanten des Graphen ordnen wir Ketten im Spielfeld zu. Für Kanten ausgehend von einem Knoten $v_1 \in V_1'$ zu einem Knoten $v_2 \in V_2'$ ist eine solche Kette in Abbildung 7.6 zu sehen, falls es sich um eine Kante der Länge 1 handelt und in Abbildung 7.7 zu sehen, falls es sich um eine Kante der Länge 2 handelt.

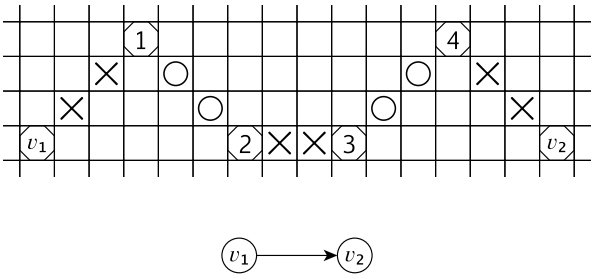

Abbildung 7.6: Kette entsprechend Kante der Länge 1

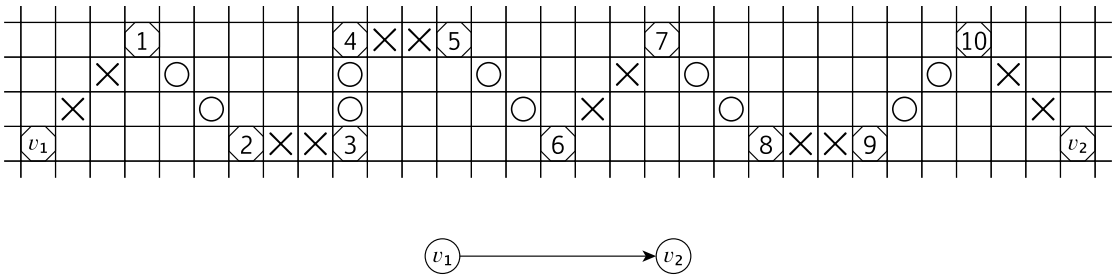

Abbildung 7.7: Kette entsprechend Kante der Länge 2

Um zu bewirken, dass die beiden Spieler ihre Symbole jeweils an den vorgesehenen Stellen machen, ergänzen wir nach der gesamten Überführung des Graphen noch Kreuze und Kreise entsprechend Abbildung 7.8 an einer freien Stelle auf dem Spielfeld, sodass genügend Abstand zu den anderen Symbolen vorhanden ist. Der so erzeugte Zugzwang basiert darauf, dass der Spieler, dem es als erstes gelingt auf das Feld 1 zu setzen, eine Gewinnstrategie hat, vorausgesetzt der andere Spieler hat zuvor nicht zwei Reihen mit jeweils drei eigenen Symbolen und mindestens zwei freien Feldern pro Seite.

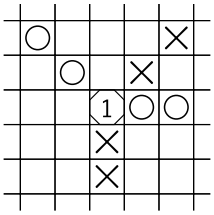

Abbildung 7.8: Zugzwang

Damit der erste Spieler beginnt, indem er auf Feld b setzt, ergänzen wir drei Kreise, wie in Abbildung 7.9 gezeigt.

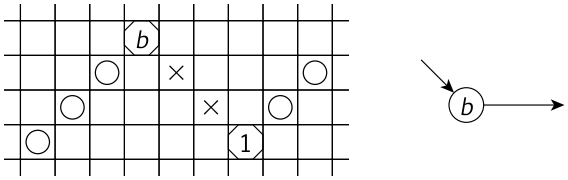

Abbildung 7.9: Startknoten

Der Spieler, der im Graphen einen Knoten ohne ausgehende Kanten erreicht, also nach Bedingung 3 einen Knoten vom Grad 1, gewinnt. Wir realisieren dies, indem wir Symbole entsprechend Abbildung 7.10 ergänzen.

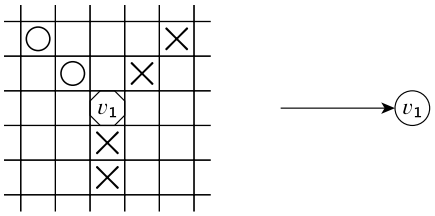

Abbildung 7.10: Knoten vom Grad 1

Im Folgenden betrachten wir die Verknüpfungspunkte an den Kanten genauer. Verknüpfungen an Knoten mit genau einer eingehenden und einer ausgehenden Kante sind wie in den Abbildungen 7.11 und 7.12 dargestellt umzusetzen.

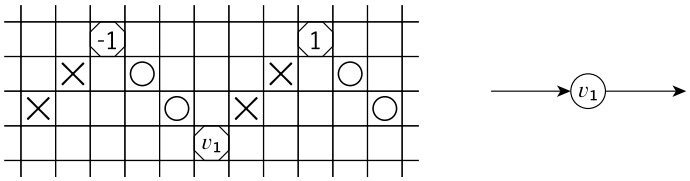

Abbildung 7.11: 1–auf–1–Verknüpfung im 180° Winkel

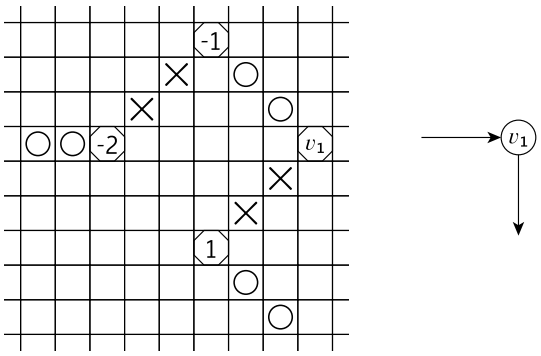

Abbildung 7.12: 1–auf–1–Verknüpfung im 90° Winkel

Abbildung 7.13 zeigt, wie Verknüpfungen an Knoten mit einer eingehenden und zwei ausgehenden Kanten zu realisieren sind. Die Verschiebung dreier Symbole ist zu beachten.

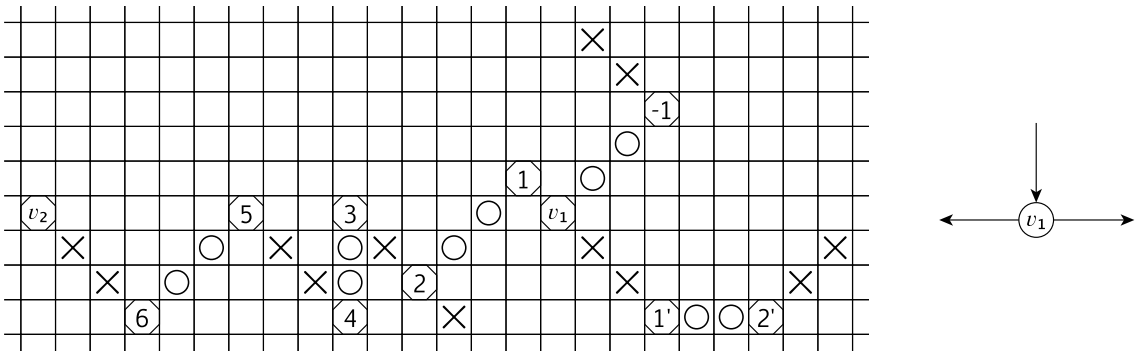

Abbildung 7.13: 1–auf–2–Verknüpfung

Handelt es sich um Verknüpfungen an Knoten mit zwei eingehenden und einer ausgehenden Kante, so unterscheiden wir zwei Fälle. Im ersten Fall stehen die beiden eingehenden Kanten im Winkel von 180° zueinander und es folgt die Umsetzung wie in Abbildung 7.14. Im zweiten Fall stehen die beiden eingehenden Kanten im Winkel von 90° zueinander und Abbildung 7.15 zeigt die Umsetzung.

Da in [Rei80] nicht klar wird, dass die dort angegebenen Vorgehensweisen funktionieren, handelt es sich bei den Abbildungen 7.14 und 7.15 um Abwandlungen der entsprechenden Umsetzungen.

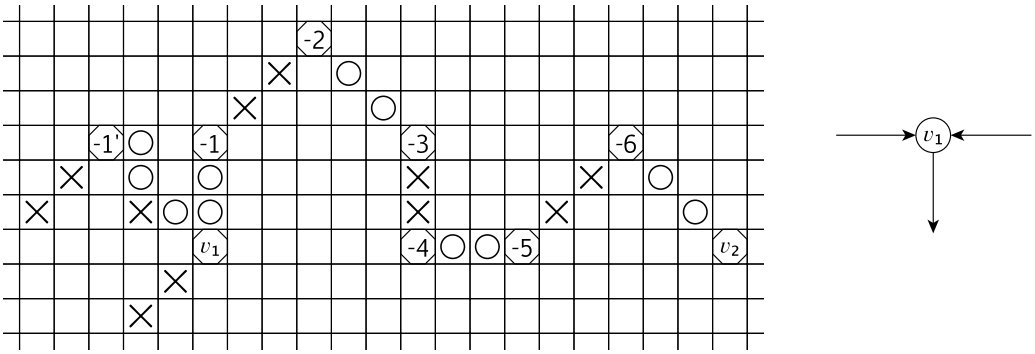

Abbildung 7.14: 2–auf–1–Verknüpfung im 180° Winkel

Durch den zusätzlichen Kreis im Feld rechts neben Feld -1′ ergibt sich eine weitere Siegesmöglichkeit für den zweiten Spieler. Falls es also zu der Situation kommt, dass der zweite Spieler Kreise in die Felder -1 und -1′ gesetzt hat, so muss der erste Spieler sein nächstes Kreuz in das Feld links neben Feld -1 setzen. Abbildung 7.15 ist nun analog aufgebaut.

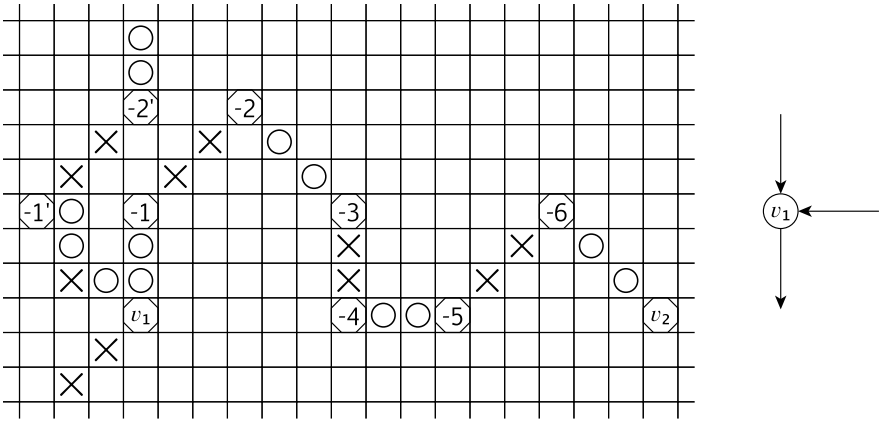

Abbildung 7.15: 2–auf–1–Verknüpfung im 90° Winkel

Alle hier dargestellten Verknüpfungen sind für Knoten aus V_1' angegeben. Für Verknüpfungen an Knoten aus V_2' ändern sich die Kreuz–Symbole zu Kreis–Symbolen und umgekehrt. Dreht und spiegelt man die einzelnen Situationen, so erhält man alle möglichen Kombinationen.

Bezeichne die hier angegebene Vorgehensweise mit f. Offensichtlich ist f polyzeit berechenbar und total. Zur Beantwortung der Frage, ob f eine polyzeit Reduktion von PB3GG auf GTTT ist, bleibt also noch Bedingung 2 zu überprüfen.

Wir erinnern uns kurz daran, dass der erste Spieler ein Geographiespiel gewinnt, falls er einen Knoten erreicht, von dem keine Kante ausgeht oder nur solche Kanten ausgehen, die auf bereits besuchte Knoten zeigen.

Die Vorgehensweise f transformiert einen planaren bipartiten gerichteten Graphen in ein $k \times k$ Spielfeld so, dass der erste Spieler eine Gewinnstrategie für das Geographiespiel auf dem Graphen hat, genau dann, wenn er eine Gewinnstrategie für das Tic–Tac–Toe–Spiel auf dem Spielfeld hat. Um dies zu verifizieren, betrachten wir insbesondere die Abbildungen 7.10, 7.14 und 7.15. Sollte der erste Spieler einen Knoten vom Grad 1 erreichen, so entspricht dies zwei Reihen mit jeweils drei eigenen Symbolen und mindestens zwei freien Feldern pro Seite; sollte der erste Spieler einen Knoten erreichen, von dem nur Kanten zu bereits besuchten Knoten ausgehen, so ergibt sich eine Situation, in der der erste Spieler für seinen nächsten Zug ein beliebiges Feld, also insbesondere auch das Feld 1 aus Abbildung 7.8, wählen kann. Beides garantiert einen Sieg für den ersten Spieler. Da auch alle weiteren einzelnen Überführungen die Struktur des Graphen erhalten, ist die Aussage korrekt.

Damit erfüllt f auch Bedingung 2 und es ist bewiesen, dass f eine polyzeit Reduktion von PB3GG auf GTTT ist.

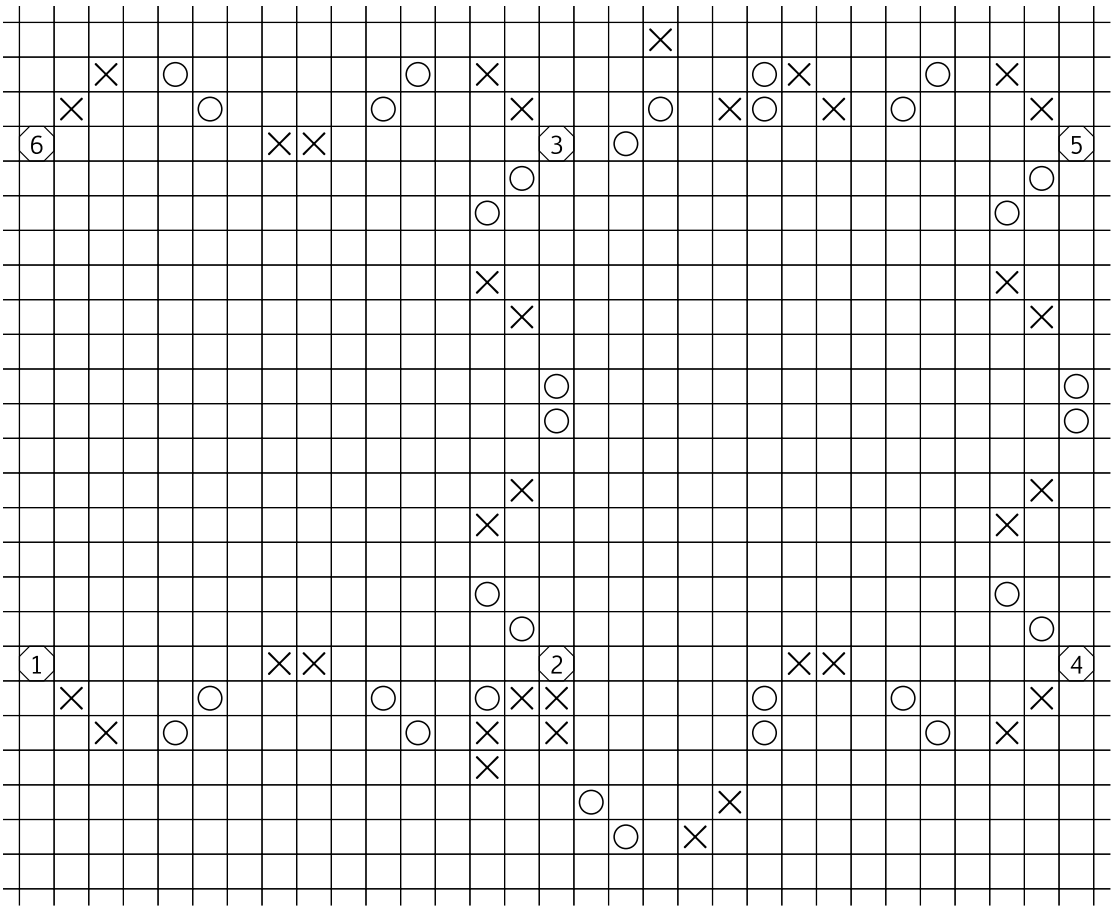

Abbildung 7.16: Ausschnitt des zugehörigen Tic–Tac–Toe–Spiels

Das zugehörige Spielfeld des Graphen aus Abbildung 7.5 hat eine Größe von 420×420 Feldern. Wir zeigen daher in Abbildung 7.16 nur den Ausschnitt des Spielfelds, der die Überführung der Knoten 1 bis 6 zeigt.

□

Satz 7.19. Es gilt

$$\mathrm{GTTT} \in \mathcal{PSPACEC}.$$

Beweis. Aus Proposition 7.12 wissen wir, dass $\mathrm{PB3GG} \in \mathcal{PSPACEC}$ ist, wobei nach Lemma 7.18 gleichzeitig PB3GG polyzeit reduzierbar auf GTTT ist und laut Proposition 7.16 bereits gilt, dass $\mathrm{GTTT} \in \mathcal{PSPACE}$ ist. Also folgt gemäß Proposition 3.13, dass $\mathrm{GTTT} \in \mathcal{PSPACEC}$ ist.

□

Man kann weitere Verallgemeinerungen von Tic–Tac–Toe betrachten, bei denen zum Sieg l eigene Symbole in einer Reihe benötigt werden. Für GTTT gilt $l = 5$. Die weiteren Verallgemeinerungen mit $l \geq 6$ liegen ebenfalls in $\mathcal{PSPACE}$, was gezeigt werden kann, indem die einzelnen Ketten jeweils um $(l - 5)$–viele Symbole verlängert werden.

8 Zusammenfassung

Dieses Kapitel gibt einen Überblick über die Inhalte dieser Arbeit. Dazu werden, wie bereits angekündigt, die Abbildungen 2.3 und 2.4 ergänzt.

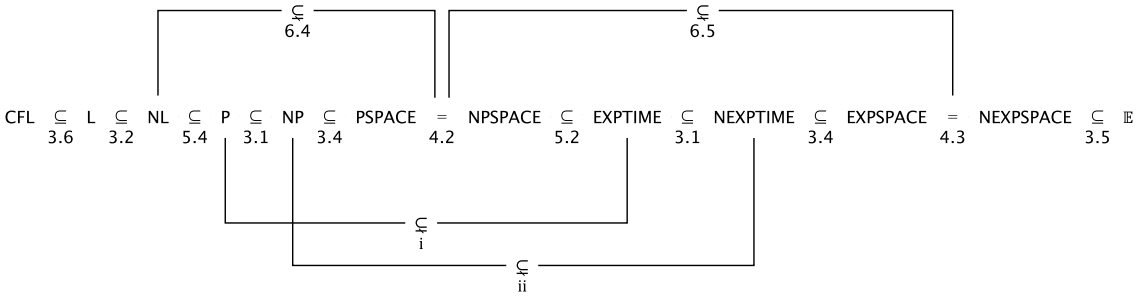

Abbildung 8.1: Komplexitätsklassen mit Verweisen

i) [Sip12, Corollary 9.13]

ii) [GJ79, Seite 184]

Die entsprechenden Aussagen können anhand der Verweise in Abbildung 8.1 gefunden werden.

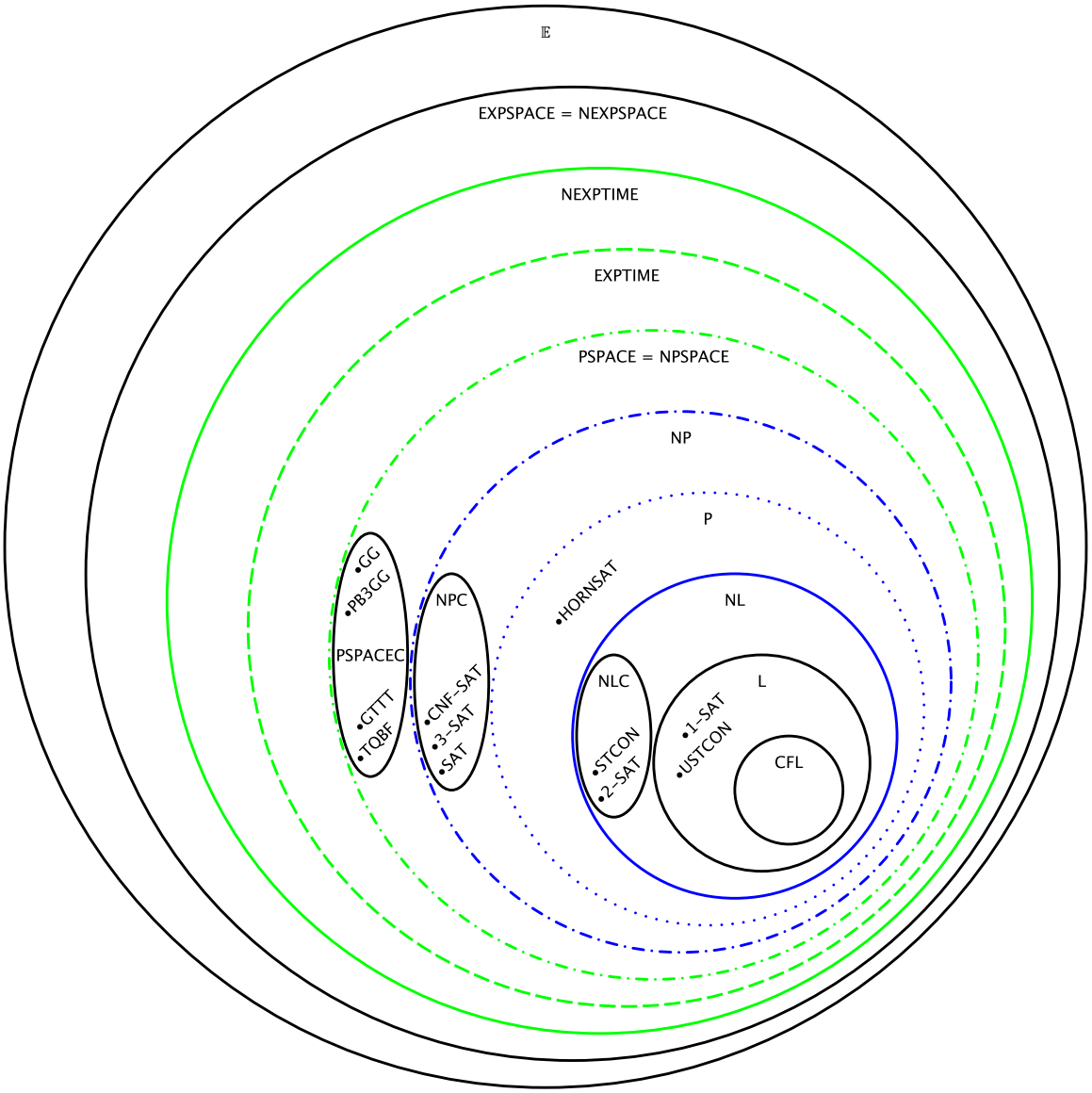

Abbildung 8.2: Komplexitätsklassen mit Sprachbeispielen

Aufgrund echter Inklusionen müssen mindestens folgende Linien existieren:

- Eine der blauen Linien (———) zwischen $\mathcal{NL}$ und $\mathcal{PSPACE}$
- Eine der grünen Linien (———) zwischen $\mathcal{PSPACE}$ und $\mathcal{EXPSPACE}$
- Eine der gepunkteten Linien (· · · · · · · oder -·-·-·-) zwischen $\mathcal{P}$ und $\mathcal{EXPTIME}$
- Eine der gestrichelten Linien (– – – – – oder -·-·-·-) zwischen $\mathcal{NP}$ und $\mathcal{NEXPTIME}$

Literaturverzeichnis

[GJ79] Michael R. Garey und David S. Johnson: *Computers and Intractability: A Guide to the Theory of NP–Completeness*. W. H. Freeman, 1. Auflage, 1979.

[HMU11] John E. Hopcroft, Rajeev Motwani und Jeffrey D. Ullman: *Einführung in Automatentheorie, Formale Sprachen und Berechenbarkeit*. Pearson Studium, 2. Auflage, 2011.

[HU69] John E. Hopcroft und Jeffrey D. Ullman: *Formal Languages and Their Relation to Automata*. Addison–Wesley, 1. Auflage, 1969.

[Jah16] Franziska Jahnke: *Vorlesung Berechenbarkeitstheorie*, 2016. http://wwwmath.uni-muenster.de/u/franziska.jahnke/bt/.

[LS80] David Lichtenstein und Michael Sipser: *GO is polynomial–space hard*. JACM, 27(2):393–401, 1980.

[Pap94] Christos H. Papadimitriou: *Computational Complexity*. Addison–Wesley, 1994.

[Rei80] Stefan Reisch: *Gobang ist PSPACE–vollständig*. Acta Inform., 13(1):59–66, 1980.

[Rei04] Omer Reingold: *Undirected Connectivity in Log–space*. Electronic Colloquium on Computational Complexity, (94), November 2004.

[Sip12] Michael Sipser: *Introduction to the Theory of Computation*. Cengage Learning, 3. Auflage, 2012.

Symbolverzeichnis

$\bigcup$	Vereinigung
$\subsetneq$	echte Teilmenge
$\subseteq$	Teilmenge
$\in$	Element
$\notin$	nicht Element
$\mathbb{N}$	Menge der natürlichen Zahlen echt größer 0
$\mathbb{R}$	Menge der reellen Zahlen
$\mathcal{P}$	Potenzmenge
$\setminus$	Mengendifferenz
$\mathcal{O}$	groß $\mathcal{O}$–Notation
$\forall$	Allquantor
$\exists$	Existenzquantor
$\exists!$	Anzahlquantor
$\wedge$	Konjunktion
$\vee$	Disjunktion
$\neg$	Negation
$\longrightarrow$	Konditional (materiale Implikation)
$\longleftrightarrow$	Bikonditional (materiale Äquivalenz)
$\Longleftrightarrow$	logische Äquivalenz (formalen Äquivalenz)
$\Longrightarrow$	logische Implikation (formale Implikation)
$\langle\cdot\rangle$	Codierung
$\preceq_l$	logplatz reduzierbar
$\preceq_p$	polyzeit reduzierbar
$\square$	Blank–Symbol

Index

Plagiatserklärung des Studierenden

Hiermit versichere ich, dass die vorliegende Arbeit über **Komplexitätsklassen und Hierarchiesätze** selbständig verfasst worden ist, dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt worden sind und dass die Stellen der Arbeit, die anderen Werken — auch elektronischen Medien — dem Wortlaut oder Sinn nach entnommen wurden, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht worden sind.

_____

(Datum, Unterschrift)

Ich erkläre mich mit einem Abgleich der Arbeit mit anderen Texten zwecks Auffindung von Übereinstimmungen sowie mit einer zu diesem Zweck vorzunehmenden Speicherung der Arbeit in eine Datenbank einverstanden.

_____

(Datum, Unterschrift)