# Grundlagen der diskreten Komplexitätstheorie II

Julian Bitterlich

SS 2011

## Inhaltsverzeichnis



# 1 Beziehungen zwischen den Komplexitätsklassen

Das Hauptziel dieses Abschnitts ist es, folgendes Resultat zu beweisen:

$$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} = \mathbf{NPSPACE} \subseteq \mathbf{EXP} \subseteq \mathbf{NEXP}.$$

Man kann auch zeigen – was hier aber nicht gemacht wird – dass $\mathbf{P} \subsetneq \mathbf{EXP}$ gilt. Es wird allgemein geglaubt, dass all die obigen Inklusionen strikt sind, aber bisher ist noch kein Beweis bekannt.
Einige der Inklusionen sind sofort klar, da jede deterministische Turingmaschine auch eine nichtdeterministische Turingmaschine ist.
Somit gelten folgende Inklusionen:

$$\mathbf{L} \subseteq \mathbf{NL},\ \mathbf{P} \subseteq \mathbf{NP},\ \mathbf{PSPACE} \subseteq \mathbf{NPSPACE},\ \mathbf{EXP} \subseteq \mathbf{NEXP}.$$

Es sind noch folgende Inklusionen zu zeigen:

- **NP** $\subseteq$ **PSPACE**
- **NL** $\subseteq$ **P** und **NPSPACE** $\subseteq$ **EXP**
- **NPSPACE** $\subseteq$ **PSPACE**.

Die Inklusionen wurden in der Reihenfolge zusammengefasst in der sie gezeigt werden.

## 1.1 NP $\subseteq$ PSPACE

Um **NP** $\subseteq$ **PSPACE** zu beweisen, werden wir ein allgemeineres Resultat beweisen.

*Bemerkung* 1.1. Wir betrachten Turingmaschinen (TM), die seperate Ein- und Ausgabebänder haben. Das Eingabeband ist „read-only“. Auf dem Ausgabeband kann der Kopf entweder stehen bleiben ohne das Zeichen auf dem er steht zu verändern, oder ein neues Zeichen schreiben und nach rechts rücken. Außerdem führen wir die Konvention ein, dass jedes Band nach links durch ein $\triangleright$ begrenzt ist, welches nicht überschritten werden oder gelöscht werden kann. Zum Start einer Maschine sind die Köpfe am linken Ende des Bandes.

**Definition 1.2.** Eine Funktion $f : \mathbb{N} \to \mathbb{N}$ heißt platzkonstruierbar/platzberechenbar, wenn es eine TM $T \in$ **SPACE**$(f(n))$ gibt, die, gestartet mit der unären Darstellung von $n \in \mathbb{N}$, die unäre Darstellung von $f(n)$ ausgibt.

*Beispiel* 1.3. Sei $k \in \mathbb{N}$. Dann sind folgende Funktionen in $n$ platzkonstruierbar:

$$k,\ k \cdot n,\ n^k,\ k^n,\ \log(n).$$

Auch die Summe und das Produkt zweier platzkonstruierbarer Funktionen sind platzkonstruierbar. Somit sind alle Polynome platzkonstruierbar.

**Theorem 1.4.** *Sei $f : \mathbb{N} \to \mathbb{N}$ platzkonstruierbar, dann ist*

$$\textbf{NTIME}(f(n)) \subseteq \textbf{SPACE}(f(n)).$$

*Proof.* Im Folgenden werden den Begriffen Konfiguration und Folgekonfiguration aus Mangel an besseren Wörtern andere Bedeutungen zugemessen. Eine Konfiguration einer TM besteht aus den aktuell gelesenen Zeichen auf den Bändern und dem Zustand. Eine Folgekonfiguration ist eine legitime Transition der Maschine anhand ihrer Übergangsrelation, also für jeden Kopf ein neues Zeichen und eine Bewegungsangabe sowie ein neuer Zustand der Maschine.
Sei $L \in$ **NTIME**$(f(n))$, dann gibt es eine nichtdeterministische TM $N$, die $L$ entscheidet, wobei die Laufzeit von $N$ durch $\alpha f(n)$ beschränkt ist für ein geeignetes $\alpha \in \mathbb{N}$ und bis auf endlich viele Ausnahmen. Wir nummerieren für jede Konfiguration von $N$ die möglichen Folgekonfigurationen durch. Es gibt eine Zahl $k \in \mathbb{N}$, so dass für eine gegebene Konfiguration von $N$ die Anzahl der möglichen Folgekonfigurationen kleiner als $k$ ist. Wir bauen nun eine deterministische Turingmaschine $D$, die im Wesentlichen so aufgebaut ist wie $N$ und auch deren Konfigurationen und Übergangsrelationen hat, mit einer Änderung, die $D$ deterministisch werden lässt. $D$ hat ein Extraband auf die sie, bevor sie $N$ simuliert, $\alpha f(|x|)$-mal 1 schreibt, wobei $x$ die Eingabe ist. Da $\alpha f$ platzkonstruierbar ist, kann dies in $\mathcal{O}(f(|x|))$ Platz geschehen. Nun simuliert $D$ die Maschine $N$, wobei sie aber bei jedem Schritt die Folgekonfiguration anhand des aktuell gelesen Wertes auf dem Extraband bestimmt: steht auf dem Extraband ein $i \in \{1, \ldots, k\}$ so wird die $i$-te mögliche Folgekonfiguration von $N$ gewählt (falls es keine $i$-te Folgekonfiguration gibt, wird einfach die erste genommen). Nach jedem Schritt wird der Kopf auf dem Extraband eine Position weitergerückt. Falls $D$ in ihrer Simulation einen akzeptierenden Zustand von $N$ erreicht, wird das Extraband um eins erhöht (dies kann geschehen ohne den Platzverbrauch zu vergrößern) um eine andere Wahl von Entscheidungen zu simulieren. Falls $D$ während der Simulation einen akzeptierenden Zustand von $N$ erreicht, akzeptiert auch $D$. $D$ verwirrft, wenn auf dem Extraband $k \ldots k$ steht und auch diese Simulation verwirrft.
Da jeder Berechnungspfad von $N$ durch $\alpha f(|x|)$ beschränkt ist, ist auch während der Simulation $D$ $\alpha f(|x|)$-platzbeschränkt und somit $L \in$ **SPACE**$(f(n))$. Da $D$ für eine Eingabe $x$ alle möglichen Berechnungspfade von $N(x)$ ausprobiert, solange bis ein akzeptierender gefunden wurde oder kein weiterer mehr existiert, akzeptiert $D$ die Eingabe $x$ genau dann wenn $N$ die Eingabe $x$ akzeptiert. □

Aus Beispiel 1.3 und Theorem 1.4 folgt nun:

**Korollar 1.5.** *Es gilt* **NP** $\subseteq$ **PSPACE**.

## 1.2 NL ⊆ P und NPSPACE ⊆ EXP

Wieder werden wir ein allgemeineres Resultat zeigen.

**Theorem 1.6.** *Sei $f : \mathbb{N} \to \mathbb{N}$ eine platzkonstruierbare Funktion, dann gilt:*

$$\mathbf{NSPACE}(f(n)) \subseteq \mathbf{TIME}(c^{\log(n)+f(n)})$$

*Proof.* Das folgende Vorgehen wird später wieder gebraucht und wird deshalb genauer erläutert. Eine Konfiguration einer $k$-Band TM $M$ ist eine Momentaufnahme der Berechnung von $M$. Sie kann als $2k+1$ Tupel dargestellt werden als $(q, w_1, u_1, \ldots, w_k, u_k)$. Dabei ist $q$ der aktuelle Zustand der Maschine und $w_i, u_i$ $i \in \{1, \ldots, k\}$ sind Wörter aus dem Arbeitsalphabet von $M$, so dass $w_i u_i$ den Inahlt des $i$-ten Bandes wiedergibt und der Kopf des $i$-ten Bandes auf dem letzten Buchstaben von $w_i$ liegt. Da das erste Band das Eingabeband ist, ändert sich dort der Bandinhalt nicht und es reicht die aktuelle Kopfposition als Binärzahl zu speichern. Somit kann eine Konfiguration von $M$ als $2k$-Tupel der Form $(q, i, w_2, u_2, \ldots, w_k, u_k)$ gespeichert werden, wobei $i$ die Binärdastellung der Kopfposition auf dem Eingabeband ist. Ist $f(n)$ der Platzverbauch der TM M, dann kann man die Anzahl der Konfigurationen wie folgt abschätzen:
Es gibt $|K|$ viele Möglichkeiten für den Zustand der TM. Es gibt $n$ mögliche Positionen für $i$ und es gibt pro Band $|\Sigma|^{f(n)}$ viele mögliche Wörter für den ersten Teil und dazu nochmal $|\Sigma|^{f(n)}$ viele mögliche Wörter für den zweiten Teil. Insgesamt gibt es also höchstens

$$n \cdot |K| \cdot \left(|\Sigma|^{2f(n)}\right)^k \leq c^{\log(n)+f(n)}$$

viele Konfigurationen, wobei $c$ eine geeignete Konstante ist, die nur von $M$ abhängt.
Wir definieren nun den Konfigurationsgraph $G(M, x)$ als denjenigen Graph, der als Knoten die Konfigurationen von $M$ auf der Eingabe $x$ hat, so dass es zwischen den Konfigurationen $C_1$ und $C_2$ genau dann eine Kante gibt, wenn $C_2$ eine Nachfolgekonfiguration von $C_1$ ist. $M$ akzeptiert die Eingabe $x$ genau dann, wenn es einen Pfad von $C_0 = (q_{start}, 0, \triangleright, \epsilon, \ldots, \triangleright, \epsilon)$ zu einen Knoten der Form $C = (q_{accept}, \ldots)$ gibt.
Sei nun $L \in \mathbf{NSPACE}(f(n))$, und die TM $N$ entscheide $L$. Dann entscheidet eine geeignete deterministische TM $D$ für ein gegebenes $x$, ob $x \in L$ gilt anhand von $G(N, x)$. Dabei wird der Graph nicht direkt ausgerechnet, sondern es wird anhand der Übergangsrelationen von $N$ geprüft, ob zwei Konfigurationen verbunden sind. Nun kann auf diesem Graphen die TM $D$ den DSF-Algorithmus ausführen, der in $\mathbf{TIME}(n^2)$ läuft, wobei $n$ die Anzahl der Knoten im Graph ist. Nach obiger Überlegung macht das $(c^2)^{\log(n)+f(n)}$ Operationen. Eine Operation des DFS hat aber nicht konstante Zeit, sondern es müssen mehrere Knoten markiert (also in eine Liste aufgenommen werden), demarkiert und auch noch weitere Knoten in eine andere Liste aufgenommen werden. Dennoch ist der Aufwand für eine dieser Operationen vernachlässigbar gegenüber dem exponentiellen Term aus der Laufzeit. Somit arbeitet unsere Maschine $D$ in einer Laufzeit von $\mathcal{O}(c^{\log(n)+f(n)})$. □

**Korollar 1.7.** *Es gilt* $\mathbf{NL} \subseteq \mathbf{P}$ *und* $\mathbf{NPSPACE} \subseteq \mathbf{EXP}$.

*Proof.* Jedes Polynom und $\log(n)$ sind platzberechenbar. Es gilt $c^{\alpha \log(n)+\log(n)} \leq n^l$ für $l$ groß genug. Für jedes Polynom $p$ gilt außerdem $c^{p(n)+\log(n)} \leq c^{n^l} = (c^l)^n$ für $l$ groß genug. □

## 1.3 NPSPACE ⊆ PSPACE und Savitch's Theorem

Wir zeigen nun einen Graphenalgorithmus, der in sehr effizientem Platz arbeitet. Auch bekannt als Savitch's Theorem.

**Theorem 1.8** (Savitch's Theorem)**.** *Für einen Graph $G$ mit $n$ Knoten und zwei Knoten $c_1, c_2$ aus $G$ kann in $\mathcal{O}(\log^2(n))$ Schritten entschieden werden, ob es einen Pfad von $c_1$ nach $c_2$ gibt.*

*Proof.* Wir führen ein dreistelliges Prädikat PATH$(x, y, i)$ ein, welches besagt, dass zwischen den Knoten $x$ und $y$ ein Weg mit höchstens $2^i$ Knoten existiert. Um zu entscheiden, ob $c_1$ und $c_2$ verbunden sind, müssen wir nur wissen, ob PATH$(c_1, c_2, \log(n))$ gilt.
Wir konzipieren eine TM $T$, die allgemein PATH$(x, y, i)$ entscheiden kann. Dafür arbeitet die Maschine rekursiv. Es gilt PATH$(x, y, 0)$ genau dann wenn $x = y$ ist oder $x$ und $y$ verbunden sind. Für $i > 0$ gilt PATH$(x, y, i)$ genau dann wenn es ein $z \in G$ gibt, so dass PATH$(x, z, i-1)$ und PATH$(z, y, i-1)$ gelten. Dieses Konzept soll auch unsere TM $T$ umsetzen. Sie startet mit $(x, y, i)$ auf einem Arbeitsband und prüft nun, ob PATH$(x, y, i)$ gilt, indem sie für jedes $z \in G$ $(x, z, i-1)$ auf das Arbeitsband schreibt und prüft, ob PATH$(x, z, i-1)$ gilt. Diesen Vorgang wiederholt sie bis der Rekursionsanker $(i = 0)$ erreicht wird. Wenn die Antwort negativ ausfällt, dann löschen wir $(x, z, i-1)$ und machen mit einem anderen $\tilde{z} \in G$ weiter (sofern dieses existiert, sonst wird ein „negativ“ rückgemeldet. Falls die Antwort positiv ausfällt löschen wir $(x, z, i-1)$ und arbeiten mit $(z, y, i-1)$ weiter. Fällt die Antwort negativ aus, wird das $(z, y, i-1)$ gelöscht und es wird mit dem nächsten $\tilde{z} \in G$ weitergemacht. Andernfalls ist PATH$(x, y, i)$ bestätigt.
Anhand der Konstruktion der Maschine $T$ sieht man, dass diese PATH$(x, y, i)$ entscheidet.
Auf dem Arbeitsband stehen höchstens $\log(n)$ viele Tripel, die jeweils die eine Länge von hchstens $3\log(n)$ haben, somit arbeitet die Maschine in Zeit $\mathcal{O}(\log^2(n))$.
Nun müssen wir nur noch die Maschine $T$ mit $(c_1, c_2, \log(n))$ laufen lassen um zu prüfen, ob es einen Weg von $c_1$ nach $c_2$ gibt. □

**Korollar 1.9.** *Sei $f : \mathbb{N} \to \mathbb{N}$ eine platzkonstruierbare Funktion mit $f(n) \geq \log(n)$, dann ist*

$$\mathbf{NSPACE}(f(n)) \subseteq \mathbf{SPACE}(f^2(n)).$$

*Proof.* Sei $L \in \mathbf{NSPACE}(f(n))$ und $N$ eine nichtdeterministische TM, die $L$ in Platz $\mathcal{O}(f(n))$ entscheidet. Wir können annehmen, dass, wenn $N$ die Eingabe $x$ akzeptiert, sie in der Konfiguration $(q_{accept}, \triangleright, x, \ldots, \triangleright, \epsilon)$ hält. Wir konstruieren eine deterministische Turingmaschine $D$, die $x \in L$ anhand von $G(N, x)$ entscheidet. Diese benutzt den Algorithmus aus Savitch's Theorem um zu überprüfen, ob $C_0 = (q_{start}, 0, \square, \epsilon, \ldots, \square, \epsilon)$ und $C_1 = (q_{accept}, \triangleright, x, \ldots, \triangleright, \epsilon)$ verbunden sind. Da die Anzahl der Konfigurationen kleiner gleich $c^{f(n)}$ ist für ein geeignetes $c$, arbeitet der Algorithmus in $\mathbf{SPACE}(f^2(n))$. □

**Korollar 1.10.** *Es gilt* **PSPACE** = **NPSPACE**

# 2 NP, coNP und Vollständigkeit

In diesem Kapitel wollen wir eine alternative Definition von **NP** finden. Desweiteren wird die Komplexitätsklasse **coNP** eingeführt.
Danach werden wir den Begiff der Vollständigkeit einführen und für **NP** und **PSPACE** vollständige Probleme angeben.

## 2.1 NP und coNP

Wir wiederholen zunächst die Definition von **NP**.

**Definition 2.1.** Eine Sprache $L \subseteq \Sigma^*$ ist in **NP**, wenn es ein $k \in \mathbb{N}$ und eine nichtdeterministische TM $T$ gibt, so dass $T$ die Sprache $L$ entscheidet und $T \in \mathbf{TIME}(n^k)$ gilt.

**Theorem 2.2.** *Eine Sprache $L \subseteq \Sigma^*$ ist genau dann in* **NP**, *wenn es ein Polynom $p$ und eine TM $M$ mit polynomieller Laufzeit (Prüfer genannt) gibt, so dass für jedes $x \in \Sigma^*$ gilt*

$$x \in L \Leftrightarrow \exists u \in \Sigma^{p(|x|)} \text{ so dass } M(x, u) \text{ akzeptiert.} \tag{1}$$

*Ist $x \in L$ und $u \in \Sigma^{p(|x|)}$, so dass $M(x, u)$ akzeptiert, dann nennen wir $u$ ein Zertifikat für $x$.*

*Proof.* Sei $L \in \mathbf{NP}$. Dann gibt es eine nichtdeterministische TM $N$ die $L$ in polynomieller Laufzeit entscheidet. Sei $D$ eine deterministische TM, die – in Analogie zum Beweis von Theorem 1.4 – einen Berechnungsast von $N$ simulieren kann, indem die Wahl der Folgekonfiguration durch ein Extraband deterministisch gestaltet wird. Aus $D$ konstruieren wir eine Maschine $\tilde{D}$, die auf Eingabe $x, u$ den Teil hinter dem Komma, also $u$ auf das Extraband kopiert und dann das Komma und $u$ löscht . Danach wird mit dem Programm von $D$ fortgefahren. Nun ist also ein Zertifikat für $x \in L$ eine Beschreibung eines Berechnungspfades von $N$ der zu einem akzeptierenden Zustand führt. Dieser existiert natürlich nur dann, wenn $x \in L$ gilt. Da außerdem $N$ polynomielle Laufzeit hat, ist der Berechnungspfad nur polynomiell lang. Somit hat $\tilde{D}$ alle Eigenschaften aus (1).
Sein nun $L$ eine Sprache, die alle Eigenshaften aus (1) hat. Sei nun $D$ ein Prüfer von $L$, dann müssen wir nur alle möglichen Zertifikate $u$ mit Länge kleiner gleich $p(n)$ erzeugen und schauen, ob $D(x, u)$ für eines dieser $u$ akzeptiert. Wenn ja, dann gilt $x \in L$, falls nicht, dann gilt $x \notin L$. Diese ganzen potentiellen Zertifikate kann man alle durch einen nichtdeterministischen Prozess erzeugen, was wegen der polynomiellen Beschränkung der $u$ in polynomieller Laufzeit geschehen kann. Danach muss nur noch $D$ ausgeführt werden. Diese nichtdeterministische TM entscheidet nun $L$ in polynomieller Laufzeit. □

**Definition 2.3.** Eine Sprache $L \subset \Sigma^*$ ist in **coNP**, wenn das Komplement $L^c = \Sigma^* \setminus L$ von $L$ in **NP** liegt, kurz: $\mathbf{coNP} = \{L : L^c \in \mathbf{NP}\}$

Man kann nun analog zu Theorem 2.2 für **coNP** eine alternative Charakterisierung durchführen:

**Theorem 2.4.** *Eine Sprache $L \subset \Sigma^*$ ist in* **coNP***, genau dann wenn es ein Polynom $p$ und eine polynomiell zeitbeschränkte* TM *$M$ gibt, so dass für jedes $x \in \Sigma^*$ gilt*

$$x \in L \Leftrightarrow \forall u \in \Sigma^{p(|x|)} \quad M(x, u) \text{ akzeptiert.} \tag{2}$$

*Bemerkung* 2.5.

- Für jede Komplexitätsklasse $\mathcal{C}$ lässt sich $\mathbf{co}\mathcal{C} := \{L : L^c \in \mathcal{C}\}$ definieren.
- Für jede deterministische Komplexitätsklasse $\mathcal{C}$, also $\mathcal{C} \in \mathbf{L}, \mathbf{P}, \mathbf{PSPACE}$,usw. gilt $\mathbf{co}\mathcal{C} = \mathcal{C}$.
- Es ist bisher unbekannt, ob $\mathbf{NP} \neq \mathbf{coNP}$.

- Nach Korollar 1.10 gilt $\mathbf{coNSPACE} = \mathbf{coPSPACE} = \mathbf{PSPACE} = \mathbf{NSPACE}$.
- Es gilt nicht $\mathbf{coNP} = \mathbf{NP}^c$ , da $\mathbf{P}$ eine Teilmenge von **NP** und von **coNP** ist.

## 2.2 Reduktion, Vollständigkeit und Härte

Die Begriffe der Reduktion, Härte und Vollständigkeit werden nun eingeführt.

**Definition 2.6.** Eine Sprache $K$ ist polynomiell reduzierbar auf die Sprache $L$, wenn es eine polynomielle TM $T$ mit Ein- und Ausgabe gibt, so dass

$$x \in K \Leftrightarrow T(x) \in L. \tag{3}$$

Man schreibt $K \leq_p L$.
Eine Komplexitätsklasse $\mathcal{C}$ heißt abgeschlossen unter (polynomieller) Reduktion, wenn für alle Sprachen $K, L \subseteq \Sigma^*$

$$K \leq_p L \wedge L \in \mathcal{C} \Rightarrow K \in \mathcal{C} \tag{4}$$

gilt.
Eine Sprache $L \subseteq \Sigma^*$ heißt $\mathcal{C}$-hart, wenn sich alle Sprachen $K \in \mathcal{C}$ auf $L$ reduzieren lassen ($K \leq_p L$).
Eine Sprache $L \subseteq \Sigma^*$ heißt $\mathcal{C}$-vollständig, wenn sie $\mathcal{C}$-hart ist und in $\mathcal{C}$ liegt.

*Bemerkung* 2.7.

- Die Klassen **P**, **NP**, **coNP**, **PSPACE**, **EXP** und **NEXP** sind abgeschlossen unter Reduktion.
- Reduktion ist eine Äquivalenzrelation (insbesondere transitiv).
- Diese Art der Reduktion ist für **P** nicht besonders sinnvoll, da jede nichttriviale Sprache **P**-vollständig ist.
- Es gibt noch andere Arten von Reduktion, z.B. Reduktion in logarithmischen Platz. Diese können dann auch für weitere Komplexitätsklassen (z.B. **P** und **NL**) interessant sein.

**Theorem 2.8.** *Seien $\mathcal{C}, \mathcal{D}$ zwei Komplexitätsklassen und $L \in \mathcal{C}$ eine Sprache. Ist $\mathcal{C}$ abgeschlossen unter Reduktion und $L$ $\mathcal{D}$-vollständig, dann ist $\mathcal{D} \subseteq \mathcal{C}$.*

*Proof.* Sei $P \in \mathcal{D}$, dann gilt $P \leq_p L \in \mathcal{C}$ und da $\mathcal{C}$ abgeschlossen unter Reduktion ist auch $P \in \mathcal{C}$ □

## 2.3 Ein NP-vollständiges Problem

Wir stellen nun ein **NP** vollständiges Problem vor:

**Definition 2.9.** Eine Formel $\phi(u_1, \ldots, u_n)$ ist in CNF (*Conjunctive Normal Form*), wenn sie von der Form

$$\bigwedge_i \left( \bigvee_j \nu_{i_j} \right)$$

ist, wobei $\nu_{i_j}$ entweder eine Variable $u_k$ ist, oder eine negierte Variable $\neg u_k$ (Auch genannt Literal).
Mit anderen Worten: Eine Formel ist in CNF, wenn sie die Verundung von veroderten Literalen ist.
Die Subformeln $\bigvee_j \nu_{i_j}$ nennt man auch Klauseln.

**Definition 2.10.** SAT ist die Sprache aller erfüllbaren CNF-Formeln (in einer geeigneten Codierung).
Ein Beispiel:

- $(a \vee b \vee c \vee d) \wedge (\neg a \vee b \vee e) \wedge (a \vee \neg c \vee \neg d) \wedge (\neg e) \wedge (c \vee \neg b) \in$ SAT

**Theorem 2.11.** SAT *ist* **NP***-vollständig.*

*Proof.* Zuerst zeigen wir, dass SAT $\in$ **NP** gilt. Dafür benutzen wir die Charakterisierung aus Theorem 2.2. Wir konstruieren eine deterministische TM $T$ die gegeben eine Variablenbelegung und eine CNF diese auswertet. Diese Maschine $T$ ersetzt nun jede Variable mit ihrem gegebenen Wert und überprüft danach, ob in jeder Klausel mindestens eine 1 vorkommt. Dies kann alles in polynomieller Zeit geschehen. Somit ist $T$ der Prüfer und ein Zertifikat ist eine erfüllende Belegung.
Nun zeigen wir, dass SAT **NP**-hart ist. Sei $L \in$ **NP**, dann gibt es nach Theorem 2.2 eine Polynmialzeit-TM $M$ und ein Polynom $p$, so dass $x \in L \Leftrightarrow M(x, u) = 1$ für ein $u \in \Sigma^{p(|x|)}$.
Zuersteinmal braucht unsere Maschine $M$ kein Ausgabeband, da sie sowieso nichts ausgibt. Außerdem soll sie, bevor sie in den Endzustand geht, alle Köpfe nach ganz links fahren. Man kann zeigen, dass jede k-Band TM durch eine 2-Band TM $S$ bestehend aus Eingabeband und Arbeitsband simuliert werden kann, so dass $S$ noch zwei weiteren Bedingugnen genügt: Die Simulation braucht asymptotisch nur quadratisch mehr Zeit und sie ist eingabeunabhängig, d.h. alle Kopfpositionen hängen nur von der Länge der Eingabe ab. Desweiteren kodieren wir den Zeichensatz der Maschine $M$ in Binärzahlen, so das das Arbeitsalphabet der Maschine nur noch aus $\{0, 1\}$ besteht und der Code für unterschiedliche Zeichen gleich lang ist. Dabei wird unsere Maschine um einen konstanten Faktor, der von $|\Sigma|$ abhängt, langsamer. Natürlich muss für diese Maschine auch die Eingabe umkodiert werden, dazu kommen wir später wieder. Somit können wir also annehmen, dass $M$ eine eingabeunabhängige 2-Band TM mit Alphabet $\{0, 1\}$ ist.

Die Idee des Beweises liegt darin, dass jeder Berechnungsschritt von $M$ auf der kodierten Eingabe $x, u$ nur von wenigen Zeitpunkten vorher abhängt. Wir führen den Begriff des Schnappschusses ein. Dieser beinhaltet bei einer gegebenen Konfiguration einer TM ihren momentan gelesenen Buchstaben und ihren Zustand. Bei uns ist das also ein Tripel $(a, b, q) \in \{0,1\} \times \{0,1\} \times \Gamma$, wobei $a$ der aktuell gelesene Buchstabe auf dem Eingabeband ist und $b$ der auf dem Arbeitsband. Den Schnappschuss für den $i$-ten Berechnungsschritt kürzen wir mit $z_i$ ab. Wenn wir $z_i$ bestimmen wollen, brauchen wir nur die Kopfposition des Eingabekopfes zum $i$-ten Berechnungsschritt $inputpos(i)$, den Schnappschuss zum vorherigen Zeitpunkt $z_{i-1}$ und den Wert des Arbeitsbandes, der zum letzten mal zum Zeitpunkt $prev(i)$ geändert wurde, was aber auch in $z_{prev(i)}$ vorhanden ist. Da die Maschine eingabenunabhängig ist, können die Werte $inputpos(i)$ und $prev(i)$ anhand des Inputs der nur aus Nullen besteht aber genauso lang ist wie die Codierung von $x, u$ (kurz $code(x, u)$) festgestellt werden. Da $M$ in Polynomialzeit läuft, kann dies in polynomieller Zeit geschehen. Nun kodieren wir wieder jeden Schnappschuss in Binärcode, so dass für jeden Schnappschuss $z_i \in \{0,1\}^c$ gilt für ein geeignetes $c$. Dann kann man anhand der Übergangsfunktion von $M$ eine Funktion $F$ konstruieren, so dass $z_i = F(z_{i-1}, z_{prev(i)}, code(x, u)_{inputpos(i)})$.
Nach der ganzen Vorarbeit geben wir nun die Reduktion an: Für jedes $x \in \Sigma^n$ gilt $x \in L$ genau dann wenn es ein $u \in \Sigma^{p(n)}$ gibt mit $M(x, u) = 1$. Nach der obigen Diskussion gilt dies aber nur, wenn es ein $y \in \{0,1\}^{|code(x,u)|}$ gibt und eine Folge von Zeichenketten $z_1, \ldots, z_{T(n)} \in \{0,1\}^c$ (wobei $T(n)$ die Anzahl der Schritte ist, die die Maschnine bei der Bearbeitung von Eingaben der Länge $|code(0^{n+p(n)}|$ macht) so dass folgende Bedingungen erfüllt sind:

- Die ersten $|code(0)^n|$ Zeichen sind identisch zu $|code(x)|$.
- $z_1$ ist die Codierung eines Startschnappschusses, also von $(code(\triangleright), code(\triangleright), q_{start})$.
- Für jedes $i \in \{2, \ldots, T(n)\}$ gilt $z_i = F(z_{i-1}, z_{prev(i)}, code(x, u)_{inputpos(i)})$
- $z_{T(n)}$ ist die Codierung eines Halteschnappschusses der akzeptiert, also von $(code(\triangleright), code(\triangleright), q_{accept})$.

Die Formel $\phi_x$ besitzt die Variablen $y \in \{0,1\}^{code(0^{n+p(n)}}$ und $z \in \{0,1\}^{cT(n)}$ und wird die Konjunktion der obigen vier Konditionen sicherstellen. Somit gilt dann $x \in L \Leftrightarrow \phi_x \in$ **SAT**. All die obigen Bedingungen können durch CNF-Formeln von fester Größe ausgedrückt werden, somit ist die Reduktion in polynomieller Zeit machbar. □

Aus dem eben bewiesenen Resultat und Theorem 2.8 folgt:

**Korollar 2.12.** *Es ist genau dann* **SAT** $\in \mathbf{P}$ *wenn* $\mathbf{P} = \mathbf{NP}$ *gilt.*

## 2.4 Ein PSPACE-vollständiges Problem

**Definition 2.13.** **QSAT** ist die Sprache aller wahren *quantifizierten* Booleschen Formeln in pränex Normalform $\psi = Q_1 \ldots Q_n \phi(x_1, \ldots, x_n)$, wobei $Q_i$ einer der beiden Quantoren $\exists$ oder $\forall$ ist.

*Beispiel* 2.14.

- $\phi_1 \equiv \forall x_1 \exists x_2 (x_1 \vee x_2) \in$ **QSAT**
- $\phi_2 \equiv \forall x_1 \exists x - 2 (x_1 \wedge x_2) \notin$ **QSAT**

**Theorem 2.15.** **QSAT** *ist* **PSPACE***-vollständig.*

*Proof.* Zuerst zeigen wir, dass **QSAT** $\in$ **PSPACE** ist. Der Algorithmus wird rekursiv beschrieben. Sei

$$Q_1 x_1 \ldots Q_n x_n \phi(x_1, \ldots, x_n)$$

eine quantifizierte Boolesche Formel. Wenn $Q_1 = \exists$, dann müssen wir prüfen, ob eine der Formeln

$$Q_2 x_2 \ldots, Q_n x_n \phi(0, x_2, \ldots, x_n) \ , \ Q_2 x_2 \ldots, Q_n x_n \phi(1, x_2, \ldots, x_n)$$

erfüllbar ist. Im Falle $Q_1 = \forall$ müssen beide Formeln

$$Q_2 x_2 \ldots, Q_n x_n \phi(0, x_2, \ldots, x_n) \ , \ Q_2 x_2 \ldots, Q_n x_n \phi(1, x_2, \ldots, x_n)$$

erfüllbar sein. Die Formeln $\phi(0, x_2, \ldots, x_n)$ werde nicht extra auf das Band geschrieben, sondern es wird in einem Array der aktuelle Variablenwert oder ein Extrasymbol, was bedeutet, dass der Variable gerade kein Wert zugwießen ist, abgespeichert. Dieser rekursive Algorithmus löst das Problem in linearem Platz. Also ist **QSAT** $\in$ **PSPACE**. Sei nun $L \in$ **PSPACE** und $T$ eine entsprechende TM, die die Sprache in polynomiellem Platz $S(n)$ entscheidet und O.B.d.A genau eine akzeptierende Konfiguration besitzt. Unsere Reduktion einer Eingabe $x$ auf **QSAT** geht über den Konfigurationsgraphen $G(T, x)$. Wir wollen eine **QSAT**-Formel finden die besagt, dass es einen Weg in $G(T, x)$ gibt von der Startkonfiguration zur Endkonfiguration. Dafür kodieren wir jede Konfiguration wieder in einem geeigneten Binärcode die für jede Konfiguration die gleiche Länge besitzen. So ein Code kann in Länge $m \in \mathcal{O}(S(n))$ angegeben werden. Desweiteren kann man eine Formel $\phi$ finden, anhängig von $T$, die für zwei Kodierungen von Konfigurationen $C_1$ und $C_2$ genau dann wahr ist, wenn $C_2$ die Folgekonfiguration von $C_1$ ist. Daraus konstruieren wir nun eine Formel $\psi$, die für $C_1$ und $C_2$ genau dann wahr ist, wenn es einen Pfad von $C_1$ nach $C_2$ gibt. Dafür benutzen wir die Hilfsformeln $\psi_i$ mit $i \in \mathbb{N}$. $\psi_i(C_1, C_2)$ ist genau dann wahr, wenn es einen Pfad der Länge $2^i$ von $C_1$ nach $C_2$ gibt. Für $i = 0$ ist $\psi_0$ gegeben durch $\psi_0(C_1, C_2) \equiv \phi(C_1, C_2) \vee C_1 = C_2$. Wir definieren $\psi_i(C_1, C_2) \equiv \exists C'' \forall D_1 \forall D_2((D_1 = C_1 \wedge D_2 = C'') \vee (D_1 = C'' \wedge D^2 = C')) \Rightarrow \psi_{i-1}(D_1, D_2)$. Man überzeugt sich leicht, dass diese Definition der gewünschten Forderung entspricht. Somit gilt $size(\phi_i) = size(\phi_{i-1}) + K$, wobei $K$ eine Konstante ist und *size* die Länge er Formel ausdrückt. Da die Knotenmenge beschränkt ist durch $c^{S(n)}$ für ein geeignetes $c$, besagt $\psi_{S(n)\log(c)}(C_1, C_2)$, dass es einen Weg von $C_1$ nach $C_2$ gibt. Die Länge dieser Formel ist aber $\mathcal{O}(S(n))$, also ist die Formel selber polynomiell lang. Demnach konstruiert die Reduktion für ein gegebenes $x$ erst die Formel $\psi$, was in polynomieller Zeit geschehen kann, bringt diese in Pränexnormalform und setzt dann in $\psi$ die kodierte Startkonfiguration und die kodierte Endkonfiguration ein. Damit ist die Reduktion angegeben. □

# Literatur


[1] Arora, S., and Barak, B. *Computational Complexity: A Modern Approach.*

[2] Papadimitriou, C. *Computational Complexity.*