# Eine Vorstellung einiger probabilistischer Komplexitätsklassen

Andreas Feldmann
Matrikelnummer: 228962

6. Februar 2007

**Zusammenfassung**

Die wohl bekannten Komplexitätsklassen P, NP und PSPACE sind über deterministische und nicht-deterministische Turingmaschinen definiert. Was aber passiert, wenn man Zufallsereignisse in die Modelle integriert? In diesem Fall kriegt man so genannte probabilistische Turingmaschinen, die eigene Komplexitätsklassen definieren.

In dieser Arbeit werden beispielhaft einige der probabilistischen Komplexitätsklassen herausgegriffen und vorgestellt. Für einige Probleme wird gezeigt, warum sie einer bestimmten Klasse angehören, und die Klassen werden untereinander und mit den nicht probabilistischen Klassen in Beziehung gesetzt.

## 1 Einführung

Der Zufall spielt in der Informatik eine nicht unbedeutende Rolle. Man kann unter ihrer Zuhilfenahme zum Beispiel Probleme in polynomieller Zeit lösen, die mit deterministischen Methoden (unter der üblichen Annahme P $\neq$ NP) exponentiellen Zeitbedarf haben. Allerdings muss dabei aber beachtet werden, dass entweder die Laufzeit des Algorithmus nur im Erwartungswert polynomiell beschränkt werden kann, oder dass das Ergebnis nur mit einer bestimmten Wahrscheinlichkeit gültig ist.

Diese Arbeit stellt diese Arten von Algorithmen aus der Sicht der Komplexitätstheorie vor. Dafür wird, wie in der Theorie üblich, der Begriff des Algorithmus in Form von Turingmaschinen abstrahiert, um dann Zufallsereignisse in das Modell zu integrieren. Diese Ereignisse sollen echtem Zufall genügen, also von nicht-deterministischer Art sein von denen lediglich die Auftretenswahrscheinlichkeit bekannt ist, und von welchen nicht berechnet werden kann, ob sie in einem Berechnungsschritt eintreten werden oder nicht. Diese so genannten probabilistischen Turingmaschinen definieren eigene Komplexitätsklassen, für welche untersucht werden soll, in welcher Beziehung sie zueinander und zu den Klassen P, NP und PSPACE stehen.

Kapitel 2 stellt den grundlegenden Begriff der probabilistischen Turingmaschinen vor und diskutiert die Klassen BPP, RP, co-RP und ZPP. Diese können als randomisierte Erweiterungen der Klasse P gesehen werden. Dagegen wird in Kapitel 3 die Klasse IP vorgestellt, welche als probabilistische Erweiterung der Klasse NP gesehen werden kann. Wie wir sehen werden enthält diese bemerkenswert viele Probleme.

# 2 Probabilistische Turingmaschinen und Komplexitätsklassen mit polynomieller Laufzeit

Um probabilistische Komplexitätsklassen zu definieren braucht man den Begriff der probabilistischen Turingmaschinen. Diese funktionieren im Wesentlichen wie nicht-deterministische Turingmaschinen mit dem Unterschied, dass jeder nicht-deterministische Berechnungsschritt mit einer bestimmten Wahrscheinlichkeit ausgeführt wird.

**Definition 1.** Eine *probabilistische Turingmaschine* $M$ ist ein 5-Tupel $M = (Q, \Sigma, q_0, F, \Delta)$. O.B.d.A. gilt für jeden Zustand $q \in Q$ und jedes $a \in \Sigma$ aus dem Eingabealphabet, dass $|\Delta(q,a)| \leq 2$, und falls $|\Delta(q,a)| = 2$ so wird $q' \in \Delta(q,a)$ mit Wahrscheinlichkeit $\frac{1}{2}$ ausgewählt. Wenn $k$ die Anzahl der probabilistischen Schritte einer Berechnung $B$ ist, dann sei die Wahrscheinlichkeit von $B$ gegeben durch $\Pr[B] = 2^{-k}$. Somit ist für eine Eingabe $w$ die Wahrscheinlichkeit, dass $M$ das Wort $w$ akzeptiert

$$\Pr[M \text{ akzeptiert } w] = \sum_{B \text{ akzeptiert } w} \Pr[B]$$

und entsprechend $\Pr[M \text{ akzeptiert } w \text{ nicht }] = 1 - \Pr[M \text{ akzeptiert } w]$.

Wir sagen, dass $M$ eine Sprache $L$ mit *Fehlerwahrscheinlichkeiten* $\epsilon_1$ und $\epsilon_2$ erkennt wenn gilt:

1. falls $w \in L$ ist $\Pr[M \text{ akzeptiert } w] \geq 1 - \epsilon_1$ und
2. falls $w \notin L$ ist $\Pr[M \text{ akzeptiert } w \text{ nicht }] \geq 1 - \epsilon_2$.

Hierbei gilt, dass sie die Fehlerwahrscheinlichkeiten auch von $|w|$ abhängig seien können.

Nun können wir mit Hilfe der obigen Definition die probabilistischen Komplexitätsklassen definieren. Wir interessieren uns wie bei den nicht-probabilistischen Turingmaschinen für den Zeit- und Platzbedarf. Die ersten Komplexitätsklassen, die hier vorgestellt werden sollen, können als Erweiterung der Klasse P gesehen werden. Sie verlangen, dass die probabilistischen Turingmaschinen höchstens eine polynomielle Zeit in Abhängigkeit der Eingabelänge brauchen. Eine solche Maschine nennt man eine *polynomielle probabilistische Turingmaschine*.

**Definition 2.** Die Komplexitätsklasse *BPP* (Bounded-error Probabilistic Polynomial-Time) ist die Menge aller Sprachen, die von einer polynomiellen probabilistischen Turingmaschine mit Fehlerwahrscheinlichkeit $0 \leq \epsilon_1 = \epsilon_2 < \frac{1}{2}$ erkannt werden.

Die Komplexitätsklasse *RP* (Randomized Polynomial-Time) ist die Menge aller Sprachen, die von einer polynomiellen probabilistischen Turingmaschine mit Fehlerwahrscheinlichkeit $0 \leq \epsilon_1 < \frac{1}{2}$ und $\epsilon_2 = 0$ erkannt werden.

Die Komplexitätsklasse *co-RP* (complement of RP) ist die Menge aller Sprachen, die von einer polynomiellen probabilistischen Turingmaschine mit Fehlerwahrscheinlichkeit $0 \leq \epsilon_2 < \frac{1}{2}$ und $\epsilon_1 = 0$ erkannt werden.

Die Algorithmen, die in den obigen Komplexitätsklassen enthalten sind, nennt man auch Monte-Carlo-Algorithmen. Sie sind dadurch gekennzeichnet, dass die Laufzeit polynomiell beschränkt werden kann, aber das Ergebnis der Berechnung nur mit einer bestimmten Wahrscheinlichkeit gültig ist. Die Klasse BPP enthält die beidseitigen Monte-Carlo-Algorithmen, die Klassen RP und co-RP dagegen die einseitigen. Da die Fehlerwahrscheinlichkeiten auf 0 gesetzt werden können ist P offensichtlich eine Teilmenge jeder der soeben definierten Komplexitätsklassen. Beachte, dass die Klassen aus Definition 2 jeweils nur eine Fehlerwahrscheinlichkeit haben, wohingegen die probabilistischen Turingmaschinen über zwei Fehlerwahrscheinlichkeiten definiert sind. Zwar können diese in der Definition frei gewählt werden, doch zeigt Satz 1, dass eine beliebige feste Wahl ausreichend ist. Um diesen beweisen zu können brauchen wir jedoch folgendes Lemma, für welchen hier kein Beweis präsentiert werden soll, aber welcher unter [5] nachgeschlagen werden kann.

**Lemma 1.** *Sei* $Y = \sum_{i=1}^{c} X_i$ *eine Zufallsvariable, die durch die Summe der 0/1-wertigen unabhängigen Zufallsvariablen* $X_i$ *dargestellt werden kann, und* $\mu \geq E[Y] = \sum_{i=1}^{c} \Pr[X_i = 1]$. *Dann gilt für jedes* $\delta > 0$

$$\Pr[Y > (1+\delta)\mu] < \left( \frac{e^{\delta}}{(1+\delta)^{(1+\delta)}} \right)^{\mu}$$

**Satz 1.** *Zu jeder polynomiellen probabilistischen Turingmaschine* $M_1$ *mit* $L(M_1) \in BPP$ *(bzw. RP oder co-RP) und Fehlerwahrscheinlichkeit* $0 < \epsilon < \frac{1}{2}$ *gibt es eine äquivalente Turingmaschine* $M_2$, *so dass* $L(M_2) = L(M_1)$, *mit* $L(M_2) \in BPP$ *(bzw. RP oder co-RP) und Fehlerwahrscheinlichkeit* $\epsilon^t$ *für ein beliebiges* $t \in \mathbb{N}$.

*Beweis.* Wir wollen hier nur den Fall beweisen, dass die Sprache aus BPP ist. Analoge Beweise für die Klassen RP und co-RP können aber leicht abgeleitet werden.

Sei $c$ ein Wert den wir später in Abhängigkeit von $\epsilon$ und $t$ bestimmen. Dann arbeite $M_2$ wie folgt:

1. Simuliere $M_1$ $c$ mal auf der Eingabe $w$
2. Akzeptiere $w$ wenn $M_1$ $w$ mindestens $\frac{c}{2}$ mal akzeptiert

Wir wollen $\Pr[M_2 \text{ akzeptiert } w \text{ nicht}]$ bestimmen für den Fall, dass $w \in L(M_1)$. Sei $Y$ die Zufallsvariable, die angibt wie oft $M_1$ in Schritt 1 $w$ nicht akzeptiert, und $X_i$ die Zufallsvariable, die gleich 1 ist wenn $M_1$ in Iteration $i$ nicht akzeptiert und 0 sonst. Dann gilt $Y = \sum_{i=1}^{c} X_i$ und die $X_i$ sind unabhängig. Es ist $\Pr[M_2 \text{ akzeptiert } w \text{ nicht}] = \Pr[Y > \frac{1}{2}c]$. Da $w \in L(M_1)$ wissen wir, dass $\Pr[M_1 \text{ akzeptiert } w \text{ nicht}] \leq \epsilon$ und somit ist der Erwartungswert $E[Y] = c\epsilon$. Da $\epsilon < \frac{1}{2}$ gilt $\frac{1}{2\epsilon} > 1$ und um Lemma 1 anwenden zu können brauchen wir nur noch $\delta = \frac{1}{2\epsilon} - 1$ zu setzen:

$$\Pr[M_2 \text{ akzeptiert } w \text{ nicht}] = \Pr[Y > \frac{1}{2}c] < \left( \frac{e^{\frac{1}{2\epsilon}-1}}{(\frac{1}{2\epsilon})^{\frac{1}{2\epsilon}}} \right)^{c\epsilon} = (e^{\frac{1}{2}-\epsilon}\sqrt{2\epsilon})^c$$

Wir wollen nun $c$ bestimmen. Damit der Satz erfüllt ist muss gelten $(e^{\frac{1}{2}-\epsilon}\sqrt{2\epsilon})^c \leq \epsilon^t$. Wenn diese Ungleichung nach $c$ aufgelöst wird erhält man $c \geq \frac{2t \ln \epsilon}{\ln(2\epsilon)+1-2\epsilon}$

Analog ist auch $\Pr[M_2 \text{ akzeptiert } w] < \epsilon^t$ für den Fall, dass $w \notin L(M_1)$.

□

Eine weitere wichtige Gruppe von probabilistischen Algorithmen sind die Las-Vegas-Algorithmen, die in der folgenden Komplexitätsklasse enthalten sind. Diese Algorithmen sind dadurch charakterisiert, dass die gefällte Entscheidung immer richtig ist, aber die Laufzeit $Z$ je nach Ausgang der Zufallsexperimente variiert, sich aber im Erwartungswert polynomiell beschränken lässt.

**Definition 3.** Die Komplexitätsklasse *ZPP* (Zero-Error Probabilistic Polynomial-Time) ist die Menge der Sprachen, die von einer probabilistischen Turingmaschine mit Fehlerwahrscheinlichkeit $\epsilon_1 = \epsilon_2 = 0$ erkannt werden, wobei die erwartete Laufzeit $E(Z)$ polynomiell in der Länge der Eingabe ist.

Wie der nächste Satz zeigt kann jeder Las-Vegas-Algorithmus mit Hilfe der einseitigen Monte-Carlo-Algorithmen simuliert werden und umgekehrt.

**Satz 2.** *Es gilt* $ZPP = RP \cap co\text{-}RP$.

*Beweis.* Sei $L \in \mathrm{RP} \cap \text{co-RP}$. Dann gibt es probabilistische Turingmaschinen $M_1$ mit $L(M_1) \in \mathrm{RP}$ und $M_2$ mit $L(M_2) \in \text{co-RP}$, wobei $L(M_1) = L(M_2) = L$. Die Turingmaschine $N$ arbeite wie folgt:

1. Simuliere $M_1$ auf Eingabe $w$. Falls $M_1$ das Wort $w$ akzeptiert muss $w \in L$ gelten und die Berechnung kann beendet werden.
2. Simuliere $M_2$ auf Eingabe $w$. Falls $M_2$ das Wort $w$ nicht akzeptiert muss $w \notin L$ gelten und die Berechnung kann beendet werden.
3. Weiter bei 1.

Sei $I$ die Zufallsvariable, die die Anzahl der Iterationen von $N$ beschreibt. Dann gilt, falls $w \in L$, dass $\Pr[M_1 \text{ akzeptiert } w \text{ nicht}] \leq \epsilon$ und $\Pr[M_1 \text{ akzeptiert } w] \leq 1$. Da $\epsilon < \frac{1}{2}$ gilt also:

$$E(I) = \sum_j j \Pr[I = j] \leq \sum_j j\epsilon^{j-1} \cdot 1 = \frac{1}{\epsilon} \cdot \frac{\epsilon}{(1-\epsilon)^2} = \frac{1}{(1-\epsilon)^2} < 4.$$

Analoges ergibt sich für den Fall, dass $w \notin L$. Da $M_1$ und $M_2$ polynomielle Laufzeiten haben ist die erwartete Laufzeit $E(Z)$ von $N$ polynomiell in der Eingabelänge. Damit ist $L(N) \in \mathrm{ZPP}$ und es gilt, da offensichtlich $L(N) = L$, dass $\mathrm{RP} \cap \text{co-RP} \subseteq \mathrm{ZPP}$.

Sei $L \in \mathrm{ZPP}$. Dann gibt es eine probabilistische Turingmaschine $M$ mit $L(M) \in \mathrm{ZPP}$ und $L(M) = L$, wobei die erwartete Laufzeit von $M$ mit $E(Z)$ bezeichnet sei. Die Turingmaschine $N$ mit $L(N) = L$ arbeite mit Hilfe einer Konstante $c \geq 2$ wie folgt:

1. Simuliere $M$ auf Eingabe $w$ für $c \cdot E(Z)$ Berechnungsschritte.
2. Falls $M$ zu einem Ergebnis kam gebe es aus. Ansonsten verwerfe die Eingabe.

Wenn $Z$ die Laufzeit beschreibt, die $M$ in Schritt 1 braucht, dann gilt nach der Markov-Ungleichung $\Pr[Z > c \cdot E(Z)] < \frac{1}{c}$. Für die Fehlerwahrscheinlichkeit von $N$ gilt also $\epsilon < \frac{1}{c} \leq \frac{1}{2}$ und damit liegt $L$ in RP.

Analog kann man zeigen, dass $L$ auch in co-RP liegt, indem in Berechnungsschritt 2 von $N$ die Eingabe akzeptiert anstatt verworfen wird. Daher folgt $\mathrm{ZPP} \subseteq \mathrm{RP} \cap \text{co-RP}$ und insgesamt der zu beweisende Satz.

□

# 3 Interaktive Beweissysteme und die Klasse IP

Im vorangegangenen Kapitel waren die Komplexitätsklassen dadurch gekennzeichnet, dass die verwendeten Turingmaschinen durch eine polynomielle Laufzeit beschränkt waren, was in Analogie zur nicht-probabilistischen Komplexitätsklasse P gesehen werden kann. Diese Einschränkung soll in diesem Kapitel gelockert werden, so dass eine neue Komplexitätsklasse entsteht, die in Analogie zur Klasse NP steht. Es ist bekannt, dass man die Klasse NP auf zwei äquivalente Arten definieren kann. Einmal enthält NP genau die Sprachen, die von einer nicht-deterministischen polynomiellen Turingmaschine erkannt werden. Diese Sprachen sind aber auch genau die, für die es einen Polynomialzeitverifizierer für ein Zertifikat der Zugehörigkeit eines Wortes zur Sprache gibt [4]. Wenn man nun davon ausgeht, dass das Zertifikat von einem Algorithmus, den wir Beweiser nennen wollen, berechnet wurde, wird für uns nun der Fall interessant wenn man erstens zulässt, dass der Verifizierer eine polynomielle probabilistische Turingmaschine ist, und zweitens, dass nicht nur ein Zertifikat zwischen Verifizierer und Beweiser verschickt wird sondern eine polynomielle Anzahl an Nachrichten ausgetauscht werden können. Man erhält so ein interaktives Beweissystem bei dem ein Beweiser mit uneingeschränkter Berechnungskapazität einem probabilistischen Polynomialzeitverifizierer in mehreren Phasen beweisen muss, dass ein gegebenes Wort einer Sprache zugehört.

Für eine zwischen dem Beweiser $B$ und dem Verifizierer $V$ ausgetauschte Nachrichtensequenz $M_i = m_1\#\cdots\#m_i$ schreiben wir $V(w, M_i) = m_{i+1}$ (bzw. $B(w, M_i) = m_{i+1}$) wenn $V$ (bzw. $B$), bei der Ausführung eines gegebenen Beweisprotokolls mit Eingabe $w$, nach der Nachrichtensequenz $M_i$ die Nachricht $m_{i+1}$ an $B$ (bzw. $V$) schickt. Nach der letzten Phase $k$ des Berechnungsprotokolls für das Beweissystem schickt der Verifizierer entweder die Nachricht *accept* oder *reject*, so dass $V(w, M_{2k}) = accept$ bzw. $reject$ (in jeder Phase werden zwei Nachrichten verschickt: vom Verifizierer zum Beweiser und zurück).

Für einen Verifizierer $V$ und einen Beweiser $B$ soll das Ereignis „$V \leftrightarrow B$ akzeptiert $w$" den Fall beschreiben bei dem eine Nachrichtensequenz $M_{2k}$ existiert, so dass

1. für $0 \leq i < 2k$, wobei $i$ gerade ist, gilt $V(w, m_1\#\cdots\#m_i) = m_{i+1}$,
2. für $0 < i < 2k$, wobei $i$ ungerade ist, gilt $B(w, m_1\#\cdots\#m_i) = m_{i+1}$ und
3. die letzte Nachricht $m_{2k}$ ist *accept*.

**Definition 4.** Eine Sprache $L$ liegt in der Komplexitätsklasse *IP* (Interactive Proof) wenn es einen probabilistischen Polynomialzeitverifizierer $V$ und einen Beweiser $B$ gibt, so dass für alle Beweiser $\tilde{B}$ und Wörter $w$ gilt

1. aus $w \in L$ folgt $\Pr[V \leftrightarrow B \text{ akzeptiert } w] \geq 1 - \epsilon$ und
2. aus $w \notin L$ folgt $\Pr[V \leftrightarrow \tilde{B} \text{ akzeptiert } w] \leq \epsilon$

für eine Fehlerwahrscheinlichkeit $0 \leq \epsilon < \frac{1}{2}$.

Genau wie im vorherigen Kapitel kann die Fehlerwahrscheinlichkeit mit Hilfe einer Methode ähnlich der in Satz 1 beliebig verkleinert werden. Offensichtlich sind die Klassen NP und BPP in IP enthalten.

Im folgenden betrachten wir ein Beispiel für eine Sprache von der nicht bekannt ist ob sie in BPP oder NP liegt [2]. Wie wir sehen werden liegt sie aber in IP. Diese Sprache sei wie folgt definiert:

$$\text{NONISO} = \{(G_1, G_2) |\ G_1 \text{ und } G_2 \text{ sind nicht-isomorphe Graphen}\}$$

Ein Berechnungsprotokoll um NONISO zu entscheiden ist dadurch gegeben, dass der Verifizierer einen der beiden Graphen zufällig auswählt und seine Knoten zufällig vertauscht. Den so erhaltenen Graphen $H$ schickt er an den Beweiser. Der Beweiser muss nun entscheiden ob $G_1$ oder $G_2$ der Ursprungsgraph war, um dann das Ergebnis seiner Bemühungen an den Verifizierer zurückzuschicken. Falls die beiden Graphen isomorph sind kann der Beweiser dies nur mit einer Wahrscheinlichkeit von $\frac{1}{2}$ entscheiden. Wenn die Graphen dagegen nicht isomorph sind, kann der Beweiser immer den Ursprungsgraphen identifizieren, da er beliebig lange Berechnungen anstellen darf. Hieraus ergibt sich, dass der Verifizierer mit Fehlerwahrscheinlichkeit $\frac{1}{2}$ die Sprache NONISO entscheiden kann.

Wenn dieses Verfahren nun $p$ mal wiederholt wird, wobei $p$ polynomiell in der Eingabelänge ist, so sinkt die Fehlerwahrscheinlichkeit, aufgrund der Unabhängigkeit der Zufallsereignisse, auf einen exponentiell kleinen Faktor $(\frac{1}{2})^p$. Der Verifizierer hat sofort einen Beweis dafür, dass die Graphen isomorph sind, sobald der Beweiser einmal eine falsche Antwort liefert. Auf der anderen Seite hat der Verifizierer gute Gründe anzunehmen, dass die Graphen nicht isomorph sind, falls der Beweiser immer die richtige Antwort zurückschickt. Beachte, dass der Beweiser ein Gegenspieler des Verifizierers ist, d.h. er kann auch absichtlich versuchen den Verifizierer dazu zu bringen zu glauben, dass die Graphen nicht isomorph sind. Es folgt aufgrund des gegebenen Protokolls, dass NONISO in IP liegt.

Wir wollen nun zu einem bedeutenden Satz der Komplexitätstheorie kommen. Dieser besagt, dass das Beweissystem der Klasse NP durch Hinzunahme von probabilistischen Berechnungen des Verifizierers und einer polynomiellen Anzahl an austauschbaren Nachrichten zur Klasse PSPACE erweitert wird. Diese Aussage soll in zwei Teilen gezeigt werden.

**Satz 3.** *IP $\subseteq$ PSPACE.*

*Beweis.* Sei $L$ eine Sprache aus IP. Dann gibt es einen Verifizierer $V$ für $L$, der für $|w| = n$ und ein Polynom $p(x)$ genau $p(n)$ Nachrichten mit dem Beweiser $B$ austauscht. Wir konstruieren eine Turingmaschine, die nur polynomiellen Platzbedarf hat und den Verifizierer $V$ simuliert. Für ein Wort $w$ sei

$$\Pr[V \text{ akzeptiert } w] = \max_B \Pr[V \leftrightarrow B \text{ akzeptiert } w].$$

Dieser Wert ist mindestens $\frac{1}{2}$ falls $w \in L$ und kleiner als $\frac{1}{2}$ falls $w \notin L$. Um also entscheiden zu können ob $w$ in der Sprache liegt reicht es $\Pr[V$ akzeptiert $w]$ zu berechnen. Da wir dies mit Hilfe einer rekursiven Funktion bewerkstelligen wollen müssen wir die Begriffe der Wahrscheinlichkeitsereignisse erweitern. Für eine Nachrichtensequenz $M_j$ bezeichne „$V \leftrightarrow B$ akzeptiert $w \mid M_j$" das Ereignis, dass $M_j$ durch Nachrichten $m_{j+1}$ bis $m_{p(n)}$ erweitert werden kann, so dass

1. für $j \leq i < p(n)$, wobei $i$ gerade ist, gilt $V(w, m_1\#\cdots\#m_i) = m_{i+1}$,
2. für $j \leq i < p(n)$, wobei $i$ ungerade ist, gilt $B(w, m_1\#\cdots\#m_i) = m_{i+1}$ und
3. die letzte Nachricht $m_{p(n)}$ ist *accept*.

Für ein Wort $w$ und das Ereignis „$V$ akzeptiert $w \mid M_j$", für welches wir auch „$V$ akzeptiert $w$ *beginnend bei* $M_j$" sagen wollen, gilt dann

$$\Pr[V \text{ akzeptiert } w \mid M_j] = \max_B \Pr[V \leftrightarrow B \text{ akzeptiert } w \mid M_j]. \tag{1}$$

Für jedes $0 \leq j \leq p(n)$ und entsprechender Nachrichtensequenz $M_j$ sei die Funktion $N_{M_j}$ induktiv definiert wie folgt:

$$N_{M_j} = \begin{cases} 0, & \text{falls } j = p(n) \text{ und } m_{p(n)} = \textit{reject} \\ 1, & \text{falls } j = p(n) \text{ und } m_{p(n)} = \textit{accept} \\ \max_{m_{j+1}} N_{M_{j+1}}, & \text{falls } j < p \text{ ungerade} \\ \sum_{m_{j+1}} (\Pr[V(w, M_j) = m_{j+1}] \cdot N_{M_{j+1}}), & \text{falls } j < p \text{ gerade} \end{cases}$$

Sei nun $M_0$ die leere Nachrichtensequenz. Der Algorithmus kann $N_{M_0}$ mit polynomiellem Platzbedarf berechnen, da die Rekursionstiefe genau $p(n)$, also polynomiell in der Eingabelänge, ist. Was noch zu zeigen bleibt ist, dass $N_{M_0} = \Pr[V$ akzeptiert $w]$, so dass der Algorithmus mit Hilfe von $N_{M_0}$ entscheiden kann ob $w$ in der Sprache liegt. Hierfür soll induktiv gezeigt werden, dass für jedes $0 \leq j \leq p(n)$ und jedes $M_j$ gilt:

$$N_{M_j} = \Pr[V \text{ akzeptiert } w \mid M_j]$$

Induktionsanfang: Für $j = p(n)$ gilt, dass $m_{p(n)}$ entweder *accept* oder *reject* ist. Falls $m_{p(n)} = \textit{accept}$ ist per Definition $N_{M_{p(n)}} = 1$ und auch $\Pr[V$ akzeptiert $w \mid M_j] = 1$, da die Nachrichtensequenz schon auf das akzeptieren von $V$ hinweist. Falls $m_{p(n)} = \textit{reject}$ gilt analog $N_{M_{p(n)}} = \Pr[V$ akzeptiert $w \mid M_j] = 0$.

Induktionsschritt von $j+1$ nach $j$: Angenommen die Behauptung stimmt für $j+1 \leq p(n)$ und eine Nachrichtensequenz $M_{j+1}$. Falls $j$ gerade ist, ist $m_{j+1}$ eine Nachricht von $V$ nach $B$. Dann gilt

$$\begin{aligned} N_{M_j} &= \sum_{m_{j+1}} (\Pr[V(w, M_j) = m_{j+1}] \cdot N_{M_{j+1}}) \\ &= \sum_{m_{j+1}} (\Pr[V(w, M_j) = m_{j+1}] \cdot \Pr[V \text{ akzeptiert } w \mid M_{j+1}]) \\ &= \Pr[V \text{ akzeptiert } w \mid M_j]. \end{aligned}$$

Die erste Gleichung folgt aus der Definition von $N_{M_j}$ und die zweite aus der Induktionsannahme. Die dritte Gleichung folgt aus der Unabhängigkeit der Ereignisse: Falls $V$ beginnend bei $M_j$ akzeptieren soll muss für jede Nachricht $m_{j+1}$ gelten, dass, wenn $V$ von $m_{j+1}$ beginnend akzeptiert, $V$ diese Nachricht auch senden muss.

Zu zeigen bleibt, dass die Behauptung gilt wenn $j$ ungerade ist. In diesem Fall ist $m_{j+1}$ eine Nachricht von $B$ nach $V$ und es gilt

$$\begin{aligned} N_{M_j} &= \max_{m_{j+1}} N_{M_{j+1}} \\ &= \max_{m_{j+1}} \Pr[V \text{ akzeptiert } w \mid M_{j+1}] \\ &= \Pr[V \text{ akzeptiert } w \mid M_j]. \end{aligned}$$

Die erste Gleichung folgt wieder aus der Definition von $N_{M_j}$ und die zweite aus der Induktionsannahme. Die dritte Zeile folgt aus Gleichung (1): Ein entsprechender Beweiser wird genau die Nachricht $m_{j+1}$ senden, die die letzte Zeile maximiert. □

Um die zweite Hälfte der Behauptung beweisen zu können brauchen wir folgendes Lemma.

**Lemma 2.** *Für jedes $d \geq 0$ hat ein Polynome $p$ mit Grad $d$ höchstens $d$ Nullstellen oder ist immer Null.*

*Beweis.* (per Induktion)

Für $d = 0$ ist $p$ eine Konstante. Falls $p$ nicht 0 ist hat $p$ offensichtlich keine Nullstellen.

Die Behauptung gelte für $d-1$. Falls $p$ ein Polynom mit Grad $d$ ist, ungleich 0 ist und eine Nullstelle bei $a$ hat so teilt $x-a$ das Polynom $p$. Somit ist $\frac{p}{x-a}$ ein Polynom ungleich 0 mit Grad $d-1$, welches nach Induktionsvoraussetzung höchstens $d-1$ Nullstellen hat. Daher kann $p$ auch nur höchstens $d$ Nullstellen haben. □

**Satz 4.** *PSPACE $\subseteq$ IP*

*Beweis.* Um dies zu zeigen reicht es für eine PSPACE-vollständige Sprache nachzuweisen, dass sie in IP liegt. Deshalb wollen wir für die Sprache TQBF ein Beweisprotokoll für einen Verifizierer angegeben. Die Sprache TQBF besteht aus der Menge der wahren booleschen quantifizierten Sätze in Pränex-Normalform, also Formeln der Form

$$\psi = Q_1 x_1 Q_2 x_2 \cdots Q_m x_m[\phi],$$

wobei $\phi$ eine FO-Formel und jedes $Q_i$ entweder $\exists$ oder $\forall$ ist. Um ein Protokoll zur Bestimmung des Wahrheitswert von $\psi$ angeben zu können müssen wir eine Arithmetisierung von $\psi$ vornehmen. Hierfür definieren wir Funktionen $f_i : \{0,1\}^i \to \{0,1\}$ für $0 \leq i \leq m$ wie folgt:

$$f_i(a_1, \cdots, a_i) = \begin{cases} 1, & \text{wenn } Q_{i+1}x_{i+1} \cdots Q_m x_m[\phi(a_1, \cdots, a_i)] \text{ wahr ist und} \\ 0, & \text{sonst,} \end{cases}$$

wobei $\phi(a_1, \cdots, a_i)$ die Formel $\phi$ ist, nachdem für $x_1$ bis $x_i$ die Werte $a_1$ bis $a_i$ eingesetzt wurden. $f_0$ ist also der Wahrheitswert von $\psi$ und $f_m(a_1, \cdots, a_m)$ der Wahrheitswert von $\phi$ nach einsetzen von booleschen Werten. Wir wollen nun die $f_i$ durch arithmetische Funktionen ausdrücken. Dazu wird erst $\phi$ wie folgt ein Polynom $f_m(x_1, \cdots, x_m)$ zugeordnet: Wenn $\alpha$ und $\beta$ Unterformeln von $\phi$ sind dann ersetzen wir die Ausdrücke

$$\begin{aligned} \alpha \wedge \beta &\quad \text{durch} \quad \alpha \cdot \beta, \\ \neg\alpha &\quad \text{durch} \quad 1 - \alpha \text{ und} \\ \alpha \vee \beta &\quad \text{durch} \quad \alpha \star \beta := 1 - (1-\alpha)(1-\beta). \end{aligned}$$

Eine wichtige Beobachtung die man hier machen kann ist, dass der Grad der Polynome $\alpha \cdot \beta$ und $\alpha \star \beta$ höchstens die Summe der Grade von $\alpha$ und $\beta$ ist. Daher hat das Polynom $f_m$ höchstens den Grad $|\phi| = n$. Die Arithmetisierung der restlichen $f_i$ sei gegeben durch:

$$\begin{array}{lll} \text{Falls } Q_{i+1} = \forall & \text{ist} & f_i(a_1, \cdots, a_i) = f_{i+1}(a_1, \cdots, a_i, 0) \cdot f_{i+1}(a_1, \cdots, a_i, 1) \text{ und} \\ \text{falls } Q_{i+1} = \exists & \text{ist} & f_i(a_1, \cdots, a_i) = f_{i+1}(a_1, \cdots, a_i, 0) \star f_{i+1}(a_1, \cdots, a_i, 1). \end{array}$$

Da der Beweiser später im Beweisprotokoll dem Verifizierer in jeder Phase $j$ das Polynom $f_j(r_1, \cdots, r_{j-1}, x_j)$ in $x_j$ für geeignete Werte $r_1$ bis $r_{j-1}$ schicken soll ergibt sich hier ein Problem. Wie das folgende Beispiel zeigt kann es passieren, dass die Arithmetisierung der Quantoren dazu führt, dass der Grad des Polynoms exponentiell wächst. Da der Verifizierer aber in polynomieller Zeit arbeiten muss kann er das erhaltene Polynom nicht verarbeiten.

Betrachte die QBF-Formel $\delta = \forall x_1 \cdots \forall x_m [\ \bigwedge_{i=1}^m \neg(x_i \wedge \neg x_i)\ ]$ mit $|\delta| \in O(m)$. Wie sich leicht nachprüfen lässt sind dann die entsprechenden arithmetisierten Funktionen

$$f_j(x_1, \ldots, x_j) = \left( \prod_{i=1}^{j} (1 - x_i(1 - x_i)) \right)^{2^{m-j}}$$

für $1 \leq j \leq m$ und $f_0() = 1$. Der Grad des Polynoms $f_j(r_1, \cdots, r_{j-1}, x_j)$ wäre also $2^{m-j+1}$.

Um dieses Problem zu beheben müssen wir einen neuen Quantor $R$ einführen. Dieser hat keine semantische Bedeutung und ist lediglich für die Arithmetisierung wichtig. Die ursprüngliche Formel $\psi$ wird umgewandelt in die Form

$$\psi' = Qx_1Rx_1Qx_2Rx_1Rx_2Qx_3Rx_1Rx_2Rx_3 \cdots Qx_mRx_1 \cdots Rx_m[\phi].$$

Dies geht in polynomieller Zeit, da $|\psi'| \in O(|\psi|^2)$.

Die neue Formel $\psi'$ hat also nun die Form $S_1y_1S_2y_2 \cdots S_ky_k[\phi]$ wobei $S_i \in \{\forall, \exists, R\}$ und $y_i \in \{x_1, \cdots, x_m\}$. Wir definieren nun für jedes $i \leq k$ eine Funktion $f_i$, wobei wie vorher $f_k(x_1, \cdots, x_m)$ die Arithmetisierung von $\phi$ ist:

$$\begin{array}{lll} \text{Falls } S_{i+1} = \forall & \text{ist} & f_i(\cdots) = f_{i+1}(\cdots, 0) \cdot f_{i+1}(\cdots, 1), \\ \text{falls } S_{i+1} = \exists & \text{ist} & f_i(\cdots) = f_{i+1}(\cdots, 0) \star f_{i+1}(\cdots, 1) \text{ und} \\ \text{falls } S_{i+1} = R & \text{ist} & f_i(\cdots, a) = (1-a) f_{i+1}(\cdots, 0) + a f_{i+1}(\cdots, 1). \end{array}$$

Beachte, dass $f_i$ im Allgemeinen nicht von $i$ Variablen abhängt. Daher sei in der obigen Definition die letzte Variable jeweils $y_{i+1}$. Beachte außerdem, dass der Quantor $R$ den booleschen Wert eines Polynoms nicht verändert. $f_0$ ist also immer noch der Wahrheitswert von $\psi$. Allerdings führt der Ausdruck $Rx$ zu einem Polynom in welchem die Variable $x$ den Grad 1 oder 0 hat. Vor jeder Arithmetisierung eines All- oder Existenzquantors wird der Grad des Polynoms also auf 1 oder 0 geschrumpft, so dass ein Polynom $f_i$ in der Variablen $y_{i+1}$ (nachdem also für die anderen Variablen entsprechende Werte eingesetzt wurden) immer höchstens den Grad 2 hat.

Betrachten wir nun wieder das obige Beispiel. Die QBF-Formel $\delta$ wird umgewandelt in $\delta' = \forall x_1 Rx_1 \forall x_2 Rx_1 Rx_2 \cdots \forall x_m Rx_1 \cdots Rx_m [\ \bigwedge_{i=1}^m \neg(x_i \wedge \neg x_i)\ ]$ und die Arithmetisierungen haben nun keine exponentiellen Grade mehr:

$$
\begin{aligned}
f_k(x_1,\ldots,x_m) &= \prod_{i=1}^{m}(1-x_i(1-x_i)) \\
f_{k-1}(x_1,\ldots,x_m) &= \prod_{i=1}^{m-1}(1-x_i(1-x_i)) \\
&\vdots \\
f_{k-m+1}(x_1,\ldots,x_m) &= (1-x_1(1-x_1)) \\
f_{k-m}(x_1,\ldots,x_m) &= 1 \\
&\vdots \\
f_0() &= 1
\end{aligned}
$$

Nun können wir endlich das Protokoll für einen Verifizierer $V$ und einen Beweiser $B$ angegeben. Alle arithmetischen Operationen sind im folgenden über einem Körper $\mathbb{F}$ der Größe mindestens $n^4$, wobei $n = |\psi|$ die Länge der Eingabe ist. Da PRIMES in P liegt kann $V$ eigenständig eine entsprechende Primzahl suchen [1]. In die Arithmetisierungen werden also auch Werte eingesetzt, die nicht 0 oder 1 sind.

**Phase 0** ($B$ sendet $f_0()$):
$B$ sendet $f_0()$ an $V$.
$V$ überprüft, dass $f_0() = 1$. Falls nicht verwirft $V$ die Eingabe.

**Phase 1** ($B$ überzeugt $V$ davon, dass $f_0()$ korrekt ist falls $f_1(r_1)$ korrekt ist):
$B$ sendet die Koeffizienten des Polynoms $f_1(z)$ an $V$.
$V$ überprüft ob das Polynom höchstens Grad $n$ hat und benutzt die Koeffizienten um $f_1(0)$ und $f_1(1)$ zu berechnen. $S_1$ kann nur ein All- oder Existenzquantor sein, daher überprüft $V$ ob $f_0() = f_1(0) \cdot f_1(1)$ falls $S_1 = \forall$ oder $f_0() = f_1(0) \star f_1(1)$ falls $S_1 = \exists$. Falls eine der Überprüfungen fehlschlägt verwirft $V$ die Eingabe.
$V$ wählt eine Zufallszahl $r_1$ aus $\mathbb{F}$ und schickt sie an $B$.

**Phase 2** ($B$ überzeugt $V$ davon, dass $f_1(r_1)$ korrekt ist falls $f_2(r_2)$ korrekt ist):
$B$ sendet die Koeffizienten des Polynoms $f_2(z)$ an $V$.
$V$ überprüft ob das Polynom höchstens Grad $n$ hat und benutzt die Koeffizienten um $f_2(0)$ und $f_2(1)$ zu berechnen. $S_2$ kann nur der Quantor $R$ sein, daher überprüft $V$ ob $f_1(r_1) = (1-r_1)f_2(0) + r_1 f_2(1)$. Falls eine der Überprüfungen fehlschlägt verwirft $V$ die Eingabe.
$V$ wählt eine Zufallszahl $r_2$ aus $\mathbb{F}$ und schickt sie an $B$, wobei $r_2$ nun die Zahl $r_1$ ersetzt.

**Phase 3** ($B$ überzeugt $V$ davon, dass $f_2(r_2)$ korrekt ist falls $f_3(r_2, r_3)$ korrekt ist):
$B$ sendet die Koeffizienten des Polynoms $f_3(r_2, z)$ an $V$.
$V$ überprüft ob das Polynom höchstens Grad $n$ hat und benutzt die Koeffizienten um $f_3(r_2, 0)$ und $f_3(r_2, 1)$ zu berechnen. $S_3$ kann nur ein All- oder Existenzquantor sein, daher überprüft $V$ ob $f_2(r_2) = f_3(r_2, 0) \cdot f_3(r_2, 1)$ falls $S_3 = \forall$ oder $f_2(r_2) = f_3(r_2, 0) \star f_3(r_2, 1)$ falls $S_3 = \exists$. Falls eine der Überprüfungen fehlschlägt verwirft $V$ die Eingabe.
$V$ wählt eine Zufallszahl $r_3$ aus $\mathbb{F}$ und schickt sie an $B$.

$\vdots$

**Phase i** ($B$ überzeugt $V$ davon, dass $f_{i-1}(r_2 \cdots)$ korrekt ist falls $f_i(r_2 \cdots, r)$ korrekt ist):
$B$ sendet die Koeffizienten des Polynoms $f_i(r_2, \cdots, z)$ an $V$.

$V$ überprüft ob das Polynom höchstens Grad $n$ hat und benutzt die Koeffizienten um $f_i(r_2, \cdots, 0)$ und $f_i(r_2, \cdots, 1)$ zu berechnen.
$V$ überprüft ob $f_{i-1}(r_2, \cdots, r_{i-1}) =$

$$\begin{cases} f_i(r_2, \cdots, r_{i-1}, 0) \cdot f_i(r_2, \cdots, r_{i-1}, 1), & \text{falls } S_i = \forall \\ f_i(r_2, \cdots, r_{i-1}, 0) \star f_i(r_2, \cdots, r_{i-1}, 1), & \text{falls } S_i = \exists \\ (1 - r_{i-1}) f_i(r_2, \cdots, r_{i-1}, 0) + r_{i-1} f_i(r_2, \cdots, r_{i-1}, 1), & \text{falls } S_i = R. \end{cases}$$

Falls eine der obigen Überprüfungen fehlschlägt verwirft $V$ die Eingabe.
$V$ wählt eine Zufallszahl $r_i$ aus $\mathbb{F}$ und schickt sie an $B$, wobei $r_i$ die Zahl $r_{i-1}$ ersetzt falls $S_i = R$.

⋮

**Phase $\mathbf{k+1}$** ($V$ überprüft direkt, dass $f_k(r_2, \cdots, r_k)$ korrekt ist):
$V$ arithmetisiert $\phi$ und überprüft ob das resultierende Polynom nach einsetzen von $r_2$ bis $r_k$ das selbe Ergebnis liefert wie $f_k(r_2, \cdots, r_k)$. Falls nicht verwirft $V$ die Eingabe.

Falls $\psi$ wahr ist gibt es offensichtlich einen Beweiser $B$ der dem Protokoll folgen kann und somit $V$ davon überzeugen kann, dass $\psi$ in TQBF ist. Wenn dies aber nicht so ist müssen wir zeigen, dass es keinen Beweiser $\tilde{B}$ gibt, der $V$, abgesehen von einer Fehlerwahrscheinlichkeit $\epsilon < \frac{1}{2}$, davon überzeugen kann, dass $\psi$ wahr ist. Dem Protokoll folgend muss $\tilde{B}$ in Phase 0 einen falschen Wert $\tilde{f}_0()$ für $f_0()$ schicken. Da $V$ aber $f_0()$ in Phase 1 anhand der Werte $f_1(0)$ und $f_1(1)$ überprüft muss $\tilde{B}$ ein falsches Polynom $\tilde{f}_1(z)$ schicken. Am Ende der Phase kann $\tilde{B}$ Glück haben, so dass $\tilde{f}_1(r_1) = f_1(r_1)$. In diesem Fall kann $\tilde{B}$ das Polynom $f_2(z)$ in Phase 2 an $V$ schicken. Wenn $\tilde{B}$ aber kein Glück hatte muss er ein falsches Polynom $\tilde{f}_2(z)$ versenden, da $V$ den Wert $f_1(r_1)$ anhand von $f_2(0)$ und $f_2(1)$ überprüft. Dies setzt sich so fort, so dass in Phase $i$ gilt, dass, sofern $\tilde{B}$ bisher noch kein Glück hatte, er ein falsches Polynom $\tilde{f}_i(r_2, \cdots, z)$ schicken muss, da $V$ den Wert $f_{i-1}(r_2, \cdots, r_i)$ anhand von $f_i(r_2, \cdots, 0)$ und $f_i(r_2, \cdots, 1)$ überprüft. Falls $\tilde{B}$ in keiner Phase Glück hat wird $V$ die Eingabe in der letzten Phase $k+1$ verwerfen, da $f_k$ hier direkt überprüft wird.

Wie groß ist nun die Wahrscheinlichkeit, dass $\tilde{B}$ in einer Phase Glück hat? Da $V$ in jeder Phase überprüft ob das gesendete Polynom höchstens den Grad $n$ hat wissen wir, dass das Polynom $\tilde{f}_i(\cdots, z) - f_i(\cdots, z)$ auch höchstens den Grad $n$ hat. Nach Lemma 2 hat das Polynom $\tilde{f}_i(\cdots, z) - f_i(\cdots, z)$ höchstens $n$ Nullstellen, woraus folgt, dass $\tilde{f}_i(\cdots, z)$ und $f_i(\cdots, z)$ in höchstens $n$ Punkten übereinstimmen können. Da für die Zufallszahlen nur $|\mathbb{F}| = n^4$ Werte zur Verfügung stehen ist die Wahrscheinlichkeit, dass $\tilde{f}_i(r_2, \cdots, r_i) = f_i(r_2, \cdots, r_i)$ höchstens $\frac{n}{n^4}$. Da die Anzahl der Phasen in $O(n^2)$ liegt ist die Wahrscheinlichkeit dass $\tilde{B}$ in irgendeiner Phase Glück hat höchstens $\frac{1}{n}$. Damit ist also die Fehlerwahrscheinlichkeit $\epsilon \leq \frac{1}{n}$.

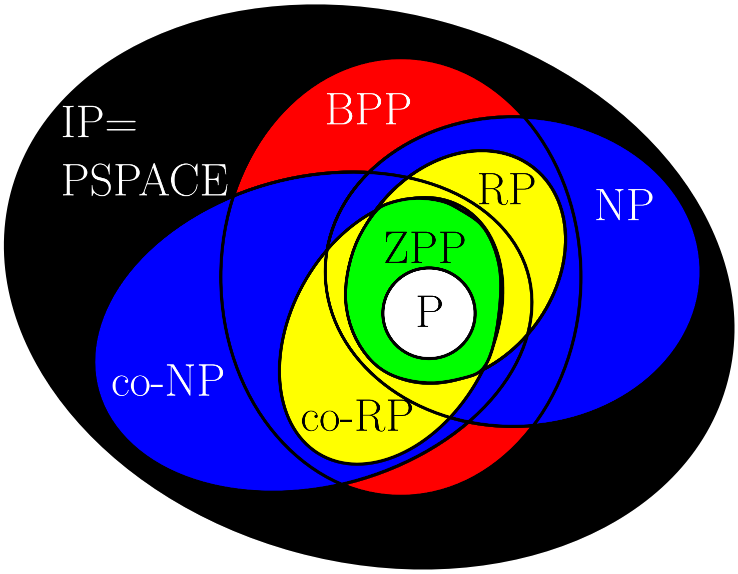
□

**Korollar 1.** *IP=PSPACE.*

# 4 Zusammenfassung

Wir haben gesehen, dass die Einführung von Zufallsexperimenten in das Modell der Turingmaschine eine interessante Erweiterung des Modells darstellt. Im Falle von polynomiellen Turingmaschinen erhält man die Monte-Carlo- und Las-Vegas-Algorithmen. Die für die Monte-Carlo-Algorithmen so typische Eigenschaft, dass das Wiederholen der Berechnung zu einer exponentiell kleinen Fehlerwahrscheinlichkeit führt, konnte anhand des Satzes zur freien Wahl der Fehlerwahrscheinlichkeit aufgezeigt werden.

Abbildung 1: Mengendiagramm der vorgestellten Komplexitätsklassen




Für die Las-Vegas-Algorithmen dagegen war interessant, dass man sie durch einseitige Monte-Carlo-Algorithmen simulieren konnte und umgekehrt.

Wenn man nun nicht die Klasse P sondern NP als Ausgangsklasse wählte kam man zu einem bemerkenswerten Ergebnis. Tatsächlich konnten wir ein Modell eines Beweissystems entwickeln, welches sich durch die Definition der Klasse NP lediglich in zwei Punkten unterschied: Für den Verifizierer sollte eine probabilistische Turingmaschine benutzt werden und es durften eine polynomielle Anzahl an Nachrichten ausgetauscht werden. Das Ergebnis war die Klasse IP, welche beachtlich viele Probleme enthielt, nämlich genau diejenigen die in der Klasse PSPACE enthalten sind. Leider konnte der Beweis dieser Äquivalenz in dieser Arbeit aufgrund der knapp bemessenen zur Verfügung stehenden Seitenzahl nicht sehr ausführlich betrieben werden. Für eine verständlichere Version siehe [2].

Abbildung 1 gibt abschließend eine grobe Übersicht darüber wie die in dieser Arbeit vorgestellten Komplexitätsklassen zusammenhängen (siehe auch [3]).

## Literatur

[1] Manindra Agrawal, Neeraj Kayal, Nitin Saxena: PRIMES is in P. *Annals of Mathematics 160(2): 781-793, 2004.*

[2] Michael Sipser: Introduction to the Theory of Computation, p. 336-347,354-367. *Boston, 1997.*

[3] Wikipedia Seite: `http://en.wikipedia.org/wiki/List_of_complexity_classes`. *Stand: November 2006.*

[4] Jurai Hromkovic: Algorithmische Konzepte der Informatik, Kapitel 6.5. *Teubner, 2001.*

[5] Jon Kleinberg, Éva Tardos: Algorithm Design, Kapitel 13.9. *Eddison-Wesley, 2006.*