Proseminar Theoretische Informatik 02.02.2016

Probabilistische Algorithmen

Diane Hanke, Alexander Korzec *Wolfgang Mulzer*

1 Einführung

1.1 Probabilistischer Algorithmus

Definition 1. *[Probabilistischer Algorithmus]: Ein probabilistischer Algorithmus ist ein Algorithmus, der **zufallsbasiert** arbeitet. Entscheidungen werden per Zufall getroffen. Ihre **Arbeitsweise** lässt sich interpretieren als Münzwurf mit fairer Münze, wobei das Ergebnis des Münzwurfes das Verhalten des probabilistischen Algorithmus beeinflusst.*

1.2 Probabilistische Turingmaschine

Definition 2. *[Probabilistische Turingmaschine]: Eine probabilistische Turingmaschine (PTM) ist ein Typ einer nichtdeterministischen Turingmaschine mit genau **zwei Übergangsfunktionen** δ_1 und δ_2.*

*In jedem **Berechnungsschritt** einer PTM M wird mittels Zufall eine der beiden Übergangsfunktionen für den nächsten Schritt gewählt. Es wird jeweils mit einer **Wahrscheinlichkeit** von $\frac{1}{2}$ die Übergangsfunktion δ_1 oder δ_2 gewählt. Diese Wahl ist **unabhängig** von den vorausgegangenen Wahlen.*

*Eine PTM kann am Ende einer Berechnung entweder eine Eingabe **akzeptieren oder verwerfen**. Mit $M(x)$ beschreiben wir eine **Zufallsvariable**, die den Wert von einer PTM M am Ende einer Berechnung angibt.*

*Sei $T : \mathbb{N} \to \mathbb{N}$ eine Funktion. Wir sagen, dass M in **T(n) Zeit** läuft, wenn M für alle Eingaben $x \in \Sigma^*$ in $T(|x|)$ Zeit hält, unabhängig von den zufälligen Entscheidungen von der Übergangsfunktion δ_1 und δ_2 in jedem Berechnungsschritt.*

1.2.1 PTM als Berechnungsbaum

Wir können uns die Berechnungen einer nichtdeterministischen TM N als einen Berechnungsbaum vorstellen. Die nichtdeterministischen Wahlen von N können wir als „Münzwürfe“ interpretieren.

Wenn b ein Knoten in einem solchen Berechnungsbaum ist, dann ist die Wahrscheinlichkeit zum Erreichen von b wie folgt definiert:

$$Pr[b] = 2^{-k}$$

Dabei ist k die Anzahl der Übergänge in diesem Berechnungsbaum bis zum Knoten b. Die Wahrscheinlichkeit, dass eine PTM M ein Wort $w \in \Sigma^*$ als Eingabe akzeptiert sei definiert als:

$$Pr[M \text{ akzeptiert } w] = \sum_{b \,=\, akz.\ Knoten} Pr[b]$$

Entsprechend ergibt sich für die Wahrscheinlichkeit, dass M eine Eingabe w verwirft:

$$Pr[M \text{ verwirft } w] = 1 - Pr[M \text{ akzeptiert } w]$$

2 Die Komplexitätsklasse $\mathcal{BPP}$

Wir wollen nun eine neue Komplexitätsklasse einführen, um Probleme untersuchen zu können. Dazu wollen wir zunächst definieren, was wir unter **Fehlerwahrscheinlichkeit** verstehen.

2.1 Fehlerwahrscheinlichkeit

Definition 3. *[Fehlerwahrscheinlichkeit]: Wenn eine PTM M eine Sprache $L \subseteq \Sigma^*$ akzeptiert, dann sei die Fehlerwahrscheinlichkeit ϵ, für $0 \leq \epsilon < \frac{1}{2}$ definiert durch:*

$$w \in L \Rightarrow Pr[M \textit{ akzeptiert } w] \geq 1 - \epsilon$$
$$w \notin L \Rightarrow Pr[M \textit{ verwirft } w] \geq 1 - \epsilon$$

Wir suchen nach probabilistischen Algorithmen, die **effizient** arbeiten in Bezug auf **Zeit und Platz**. Wir messen die **Komplexität** bei probabilistischen Turingmaschinen, wie auch bei nichtdeterministischen Turingmaschinen, durch die Betrachtung des **worst-case Pfades** für jede Eingabe $w \in \Sigma*$ (= längster Berechnungspfad).

Wir definieren eine neue Komplexitätsklasse $\mathcal{BPP}$, welche die Klasse von Sprachen beinhaltet, die von probabilistischen Turingmaschinen in polynomieller Zeit mit einer bestimmten Fehlerwahrscheinlichkeit erkannt werden.

2.2 BPTIME und BPP

Definition 4. *[BPTIME und $\mathcal{BPP}$]: Sei T: $\mathbb{N} \to \mathbb{N}$ eine Funktion und $L \subseteq \Sigma^*$ ein Entscheidungsproblem.*

Eine PTM M entscheidet eine Sprache L in $T(n)$ Zeit, wenn für alle $x \in \Sigma^*$ die PTM M in $T(|x|)$ Zeit hält, unabhängig von den zufälligen Wahlen der Berechnungsschritte mit einer Fehlerwahrscheinlichkeit von ϵ, wobei $0 < \epsilon < \frac{1}{2}$

Wir definieren BPTIME(T(n)) als die Menge aller Sprachen, welche sich durch eine PTM in $O(T(n))$ Zeit entscheiden lassen:

$$BPTIME(T(n)) := \{L \subseteq \Sigma^* \mid L \textit{ ist von einer PTM in } O(T(n)) \textit{ Zeit entscheidbar } \}$$

Wir wollen $\mathcal{BPP}$ wie folgt definieren:

$$\mathcal{BPP} := \bigcup_{c \in \mathbb{N}} BPTIME(n^c)$$

Das Erstaunliche ist, dass wir ϵ beliebig aus dem vorgegebenen Intervall auswählen können. Das bedeutet, wenn wir $\mathcal{BPP}$ mit jeweils $\epsilon_1 = 1/3$ und $\epsilon_2 = 4/129$ wählen, dann erhalten wir die gleiche Komplexitätsklasse. Daher kann man $\mathcal{BPP}$ auch folgendermaßen definieren:

Definition 5. *[Alternative Definition von $\mathcal{BPP}$]: $\mathcal{BPP}$ sei definiert als die Klasse von Sprachen, die von einer probabilistischen TM in polynomieller Zeit mit einer Fehlerwahrscheinlichkeit von $\epsilon = \frac{1}{3}$ erkannt werden.*

Das beide Definitionen äquivalent sind, rechtfertigt das **amplification lemma**.

Lemma 6. *[Amplification lemma für $\mathcal{BPP}$]: Jede konstante Fehlerwahrscheinlichkeit würde eine äquivalente Definition liefern, solange sich die Fehlerwahrscheinlichkeit, die gewählt wird, strikt zwischen 0 und $\frac{1}{2}$ liegt.*

Sei $0 < \epsilon < 1/2$ fest gewählt. Dann gilt für jedes Polynom poly(n), dass es eine PTM M_1 gibt, die mit einer Fehlerwahrscheinlichkeit von ϵ arbeitet, eine äquivalente PTM M_2 existiert, die mit einer Fehlerwahrscheinlichkeit von $2^{-poly(n)}$ arbeitet.

Beweis. Sei eine TM M_1 gegeben, die eine Sprache mit einer Fehlerwahrscheinlichkeit von $\epsilon < \frac{1}{2}$ erkennt. Wir konstruieren eine zweite TM M_2, die die gleiche Sprache erkennt mit einer Fehlerwahrscheinlichkeit von $2^{-poly(n)}$.

M_2 = "Für eine Eingabe w:

1. Berechne k
2. Führe $2k$ unabhängige Simulationen von M_1 auf der Eingabe w durch
3. wenn die Mehrheit der Durchläufe von M_1 w akzeptiert, dann akzeptiere die Eingabe, ansonsten verwirf die Eingabe "

Wir wollen abschätzen, wie hoch die Anzahl der Sequenzen der Länge $2k$ ist, sodass M_2 die falsche Antwort liefert. Bei jeder Simulation von M_1 ist das Ergebnis entweder falsch oder richtig; sind die meisten Ergebnisse richtig, gibt M_2 die richtige Antwort. Wir nehmen an, dass mindestens die Hälfte der Ergebnisse falsch sind.
Sei S die Sequenz von Ergebnissen, die M_2 in Schritt 2 liefert. Sei p_S die Wahrscheinlichkeit, dass M_2 die Sequenz S erhält. S hat r richtige Ergebnisse und f falsche Ergebnisse.

Dann gilt: $r + f = 2k$

Wenn r $\leq$ f und M_2 S erhält, gibt M_2 ein falsches Ergebnis aus und wir nennen eine solche Sequenz **bad sequence**.

Wenn S irgendeine *bad sequence* ist, dann gilt:

$$p_S \leq \epsilon^f (1-\epsilon)^r \leq \epsilon^k (1-\epsilon)^k, \text{ wegen } k \leq f \text{ , } \epsilon < 1 - \epsilon \text{ und } \epsilon < \frac{1}{2}$$

.

Die **Summe** aller p_S für jede S ergibt die Wahrscheinlichkeit, dass M_2 ein falsches Ergebnis liefert. Wir haben insgesamt 2^{2k} Sequenzen und wir nehmen an, dass alle S *bad sequences* sein können (also auch 2^{2k}). Dann gilt:

$$Pr[M_2 \text{ liefert bei Eingabe } w \text{ ein falsches Ergebnis}] = \sum_{S \text{ ist bad sequence}} p_s \leq 2^{2k} \cdot \epsilon^k (1-\epsilon)^k = (4\epsilon(1-\epsilon))^k$$

- wir haben angenommen: $\epsilon < \frac{1}{2}$, sodass gilt: $4\epsilon(1-\epsilon) < 1$
- die Wahrscheinlichkeit dafür, dass wir in M_2 einen Fehler machen, nimmt exponentiell mit der Anzahl an Durchläufen ab
- um k genau zu berechnen, sodass wir eine Fehlerwahrscheinlichkeit für M_2 von 2^{-t}, für jedes $t \geq 1$, festlegen können, definieren wir:

  $$\alpha = log(4\epsilon(1-\epsilon))$$

  und wählen

  $$k \geq t/\alpha$$

  Dann erhalten wir eine Fehlerwahrscheinlichkeit von $2^{-poly(n)}$ in polynomieller Zeit.

□

3 Primzahlen

In diesem Abschnitt betrachten wir das folgende Problem:

PRIMES
Gegeben: Eine Zahl $n \in \mathbb{N}$
Frage: Ist n eine Primzahl?

Es ist die Frage zu klären, wieso dieses Problem nicht trivial lösbar ist. Betrachten wir folgenden naiven Ansatz:

1. teste für alle $a = \{2, \ldots, \lfloor\sqrt{n}\rfloor\}$, ob ein a ein Teiler von n ist
2. akzeptiere Eingabe, falls kein Test in 1. erfolgreich, sonst verwirf

Die Anzahl der Iterationen ist $O(\sqrt{n})$. Das ist leider nur **pseudopolynomiell**, da man mit n Bits eine Eingabe der Größe $O(2^n)$ darstellen kann. Daher gilt:

$$Polynomiell \mathrel{\hat{=}} O(\log^c n) \text{ mit } c \in \mathbb{N}$$

Im Folgenden werden wir einen probabilistischen Ansatz entwickeln, der in polynomieller Laufzeit mit einer vernachlässigbaren Fehlerwahrscheinlichkeit bestimmt, ob eine Eingabe $n \in \mathbb{N}$ eine Primzahl ist. Wir benötigen zunächst einige Definitionen:

Definition 7. $[a]_n = \{a + k \cdot n \mid k \in \mathbb{Z}\}$

Definition 8. $\mathbb{Z}_n^+ = \{[a]_n \mid 1 \leq a \leq n-1\}$

Definition 9. $\mathbb{Z}_n^* = \{[a]_n \in \mathbb{Z}_n^+ \mid ggT(a,n) = 1\}$

Beobachtung 10. *Für Primzahlen p gilt $\mathbb{Z}_p^+ = \mathbb{Z}_p^*$*

Satz 11. *[Kleiner Satz von Fermat]: Wenn n eine Primzahl ist, dann gilt für alle $a \in \mathbb{Z}_n^*$*

$$a^{n-1} \equiv 1 \ (mod\ n)$$

Lemma 12. *Ist $n \geq 3$ eine Primzahl und $a \in \mathbb{Z}_n^*$ mit $a^2 \equiv 1 \ (mod\ n)$, dann gilt $a \equiv \pm 1 \ (mod\ n)$.*

Beweis.

$$\begin{aligned} a^2 \equiv 1 \ (\text{mod } n) &\Leftrightarrow a^2 - 1 \equiv 0 \ (\text{mod } n) \\ &\Leftrightarrow n \mid a^2 - 1 \\ &\Leftrightarrow n \mid (a-1)(a+1) \\ &\Leftrightarrow n \mid (a-1) \vee n \mid (a+1) \\ &\Leftrightarrow x - 1 \equiv 0 \ (\text{mod } n) \vee x + 1 \equiv 0 \ (\text{mod } n) \\ &\Leftrightarrow a \equiv 1 \ (\text{mod } n) \vee a \equiv -1 \ (\text{mod } n) \end{aligned}$$

□

Bei Algorithmen für zahlentheoretische Probleme verwendet man als Aufwandsmaß die Anzahl der Bitoperationen in den einzelnen Berechnungen. Die Anzahl der Bits einer Eingabe n erhalten wir mit $\log n$. Wir wollen nicht näher drauf eingehen und vereinbaren folgenden Aufwand für die auftretenden elementaren Rechenoperationen:

Satz 13. *Modulares Potenzieren ist in $O(\log^3 n)$ möglich (Zeigen es hier nicht, sondern nehmen an, dass eine „Black-Box“ existiert, die das tut). Außerdem nehmen wir an, dass die Multiplikation und die Modolo-Operation zweier n-stelliger Zahlen in $O(\log^2 n)$ möglich ist („Normale Schulmethode“)*

3.1 Ein erster Ansatz

Idee: Teste mit **Satz 11**. Nehme zunächst an, dass eine Eingabe $n \in \mathbb{N}$ prim sei. Versuche anschließend Zeugen für die Zusammengesetztheit von n zu finden. Falls das der Fall sein sollte, dann erhalten wir in **Satz 11** eine Implikation der Form $1 \Rightarrow 0$. Aus dem Widerspruch folgt dann, dass n zusammengesetzt sein muss.

Fermat Test: *Eingabe:* $n \in \mathbb{N}$, $k \in \mathbb{N}$ Anzahl der Iterationen

1. Wiederhole k mal
2. Wähle $a \in \{2, ..., n-2\}$ zufällig
3. Berechne $a^{n-1} \ (\text{mod } n)$
4. Falls einer dieser k Versuche in 3. $\neq 1$ war, dann verwirf, sonst akzeptiere die Eingabe

Problem: *Der kleine Satz von Fermat ist keine Äquivalenz!* Es existieren sogenannte *Carmichael Zahlen*, die für alle $a \in \mathbb{Z}_n^*$ $a^{n-1} \ (\text{mod } n) \equiv 1$ liefern, obwohl sie zusammengesetzt sind. Beispielsweise ist die $561 = 3 \cdot 11 \cdot 17$ eine solche Zahl. Der Fermat Test lässt sich von diesen Eingaben mit hoher Wahrscheinlichkeit „austricksen“ . Wir betrachten deswegen einen verbesserten Ansatz.

3.2 Miller-Rabin

Idee: Erweitere den Fermat-Test mit Test auf **Lemma 14** und suche wieder Zeugen für die Zusammengesetztheit von n. Dafür zerlegen wir $n-1$ in s ungerade und $t \geq 1$ wie folgt

$$n - 1 = s \cdot 2^t$$

Der Algorithmus:

Eingabe : $n \in \mathbb{N}$, $k \in \mathbb{N}$ Anzahl der Iterationen
Ausgabe : Ist n eine Primzahl?

solange *noch nicht k mal wiederholt* **tue**
  wenn $n = 2$ **dann**
    „nächster Versuch“
  Ende
  wenn $n \mid 2$ **dann**
    gebe „Zusammengesetzt“ **zurück**
  Ende
  finde s ungerade und $t \geq 1$ mit $n - 1 = s \cdot 2^t$
  wähle $a \in \{2, \ldots, n-2\}$ zufällig
  $b := a^s \pmod{n}$
  wenn $b = 1$ ***oder*** $b = n - 1$ **dann**
    „nächster Versuch“
  Ende
  für jedes $i = 1, \ldots, t-1$ **tue**
    $b := b^2 \pmod{n}$
    wenn $b = 1$ **dann**
      „nächster Versuch“
    Ende
    wenn $n = n - 1$ **dann**
      gebe „Zusammengesetzt“ **zurück**
    Ende
  Ende
  gebe „Zusammengesetzt“ **zurück**
Ende
gebe „Primzahl“**zurück**

Berechne anschließend die Folge

$$X =< s \cdot 2^0, s \cdot 2^1, \ldots, s \cdot 2^{t-1}, s \cdot 2^t >$$

Folgende Fälle sind möglich:
Fall 1a: $s \cdot 2^0 = 1$
Fall 1b: $s \cdot 2^0 \neq 1$ und es existiert ein $i \leq t-1$ mit $s \cdot 2^i = n - 1$
In den Fällen 1a und 1b haben wir keinen Zeugen dafür gefunden, dass die Eingabe n zusammengesetzt sein soll.
Fall 2: $s \cdot 2^t \neq 1 \Rightarrow$ Nach **Satz 11** ist n zusammengesetzt.

$s \cdot 2^0$	$s \cdot 2^1$	...	...	...	...	...	$s \cdot 2^{t_1}$	$s \cdot 2^t$	Fall
1	1	...	1	1	1	...	1	1	1a
$n-1$	1	...	1	1	1	...	1	1	1b
*	*	...	*	*	*	...	*	$n-1$	2
*	*	...	*	*	*	...	*	*	2
*	*	...	*	1	1	...	1	1	3
*	*	...	*	*	*	...	*	1	3

Fall 3: $s \cdot 2^0 \neq 1$, aber $s \cdot 2^t = 1 \Rightarrow$ Suche das minimale $i \geq 1$ mit $s \cdot 2^i = 1$. Es gilt $s \cdot 2^{i-1} \neq \pm 1$ und damit ist n zusammengesetzt nach **Lemma 12**.

Laufzeit: Die „interessanten“ Zeilen, wo wir tatsächlich was berechnen, sind die Zeilen 10 und 14-22. Zu Zeile 10: Nach **Satz 13** haben wir eine „Black-Box“, die uns in $O(\log^3 n)$ modulares Potenzieren ermöglicht. Damit ist die Laufzeit hier $O(\log^3 n)$
Zu den Zeilen 14-22: In Zeile 15 führen wir eine Multiplikation und eine Modolo Operation durch. Die Laufzeit dieser Berechnungen ist $O(\log^2 n)$ nach **Satz 13**. Die Schleife in den Zeilen 14-22 wird $O(\log n)$ mal ausgeführt. Damit ist die Laufzeit hier insgesamt:

$$O(\log n) \cdot O(\log^2 n) = O(\log^3 n)$$

Die äußere Schleife wird insgesamt k-mal ausgeführt. Damit ist die Gesamtlaufzeit $O(k \cdot \log^3 n)$. k gehört zwar bei unserer Beschreibung vom Algorithmus zur Eingabe, ist aber eher als zusätzlicher fest gewählter Parameter für die Anzahl der Durchläufe gemeint. Wählen wir also k vernünftig, dann ist die Laufzeit vom Miller-Rabin Test polynomiell.

Zur Fehlerwahrscheinlichkeit: Wegen **Satz 11** und **Lemma 12** gilt:

Satz 14. *Ist die Eingabe $n \in \mathbb{N}$ eine ungeradene Primzahl, dann gilt*

$$Pr[\textit{Miller-Rabin akzeptiert } n] = 1$$

Das scheint plausibel, weil für Primzahlen n keine Zeugen für die Zusammengesetztheit existieren und der Miller-Rabin Test, damit keinen Grund hat „Zusammengesetzt“ auszugeben.
Man kann außerdem zeigen:

Lemma 15. *Die Anzahl der Zeugen für die Zusammengesetztheit einer ungeradenen zusammengesetzten Zahl n ist $\geq \dfrac{n-1}{2}$*

Daraus folgt

Satz 16. *Ist $n \in \mathbb{N}$ eine ungeradene zusammengesetzte Zahl, dann gilt*

$$Pr[\textit{Miller-Rabin akzeptiert } n] \leq 2^{-k}$$

wobei $k \in \mathbb{N}$ die Anzahl der Wiederholungen ist.

Durch **Lemma 15** reduziert darauf zu bestimmen wie wahrscheinlich es ist in k Versuchen keinen Zeugen $a \in \{2, \ldots, n-2\}$ zu ziehen, der die Zusammengesetztheit einer Zahl n belegt.

4 weitere Komplexitätsklassen

Der Miller-Rabin Test macht nur dann einen Fehler, wenn die Eingabe n eine zusammengesetzte Zahl ist. Das bedeutet, wenn n eine Primzahl ist, dann gibt der Miller-Rabin Test garantiert „Primzahl“ aus. Man sagt auch, dass der Miller-Rabin Test nur einen sogenannten *einseitigen Fehler macht*

4.1 $\mathcal{RP}$

Definition 17. *[RTIME und $\mathcal{RP}$]: Sei $T : \mathbb{N} \to \mathbb{N}$. In $RTIME(T(n))$ ist jedes Entscheidungsproblem $L \subseteq \Sigma^*$ enthalten für die eine PTM M existiert, die in $T(n)$-Zeit läuft und für die*

$$x \in L \Rightarrow Pr[M(x) = 1] \geq \frac{1}{2} \tag{1}$$

$$x \notin L \Rightarrow Pr[M(x) = 0] = 1 \tag{2}$$

gilt. Darüber hinaus definieren wir

$$\mathcal{RP} := \bigcup_{c \in \mathbb{N}} RTIME(n^c)$$

Wie auch bei $\mathcal{BPP}$ kann man zeigen, dass die Konstante 2/3 willkürlich gewählt ist und man mit $1 - \epsilon$ mit $0 < \epsilon < 1/2$ eine äquivalente Beschreibung von $\mathcal{RP}$ erhält.

Definition 18. *[co-$\mathcal{RP}$]: co-$\mathcal{RP} := \{L \subseteq \Sigma^* \mid \overline{L} \in \mathcal{RP}\}$*

Definieren wir $\mathsf{COMPOSITES} := \overline{PRIMES}$. Dann haben wir mit unserem Algorithmus $\mathsf{COMPOSITES} \in \mathcal{RP}$ gezeigt, weil wir eine Zahl nur dann als zusammengesetzt erkennen, wenn wir einen Zeugen gefunden haben. Da für keine Primzahl ein solcher Zeuge existiert, wird keine Primzahl als zusammengesetzte Zahl erkannt. Daraus folgt $\mathsf{PRIMES} \in$ co-$\mathcal{RP}$.

4.2 $\mathcal{ZPP}$

Definition 19. *[$ZTIME$ und $\mathcal{ZPP}$]: Sei $T : \mathbb{N} \to \mathbb{N}$. In $ZTIME(T(n))$ ist jedes Entscheidungsproblem $L \subseteq \Sigma^*$ enthalten für die eine PTM M existiert, die in $O(T(n))$ erwarteter Laufzeit eine korrekte Antwort liefert oder „?“ ausgibt. Dafür erweitern wir die Menge der Endzustände unserer PTM M um den Zustand „weiß nicht“. Die Wahrscheinlichkeit für „?“ ist $< 1/2$.*

5 Beziehungen zu bekannten und neuen Komplexitätsklassen

Satz 20. $\mathcal{P} \subseteq \mathcal{BPP}$

Beweisidee: Simuliere DTM $M = (Q, \Sigma, \Gamma, q_0, F, \square, \delta)$ mit PTM $M' = (Q', \Sigma', \Gamma', q_0', F', \square, \delta_1, \delta_2)$, indem wir $Q' := Q, \Sigma' := \Sigma, \Gamma' := \Gamma, q_0' := q_0, F' := F$ übernehmen und $\delta_1 := \delta, \delta_2 := \delta$ setzen. Da $\delta_1 = \delta_2$ werden die zufälligen Münzwürfe „egalisiert “ .

Satz 21. $\mathcal{BPP} \subseteq \mathcal{PSPACE}$

Beweisidee: Eine PTM ist eine spezielle NTM. Eine DTM kann eine NTM mit polynomiell viel Speicher simulieren.

Satz 22. $\mathcal{RP} \subseteq \mathcal{BPP}$

Beweisidee: Siehe Definitionen.

Satz 23. *co-*$\mathcal{RP} \subseteq \mathcal{BPP}$

Beweisidee: Siehe Definitionen.

Satz 24. $\mathcal{RP} \subseteq \mathcal{NP}$

Beweisidee: Wenn eine NTM M für eine Eingabe $w \notin \Sigma^*$ $M(w) = 0$ ausgibt, dann existiert keine akzeptierende Berechnung. Genauso wie bei einer PTM mit einseitig beschränktem Fehler. Wenn eine NTM M eine Eingabe $w \in \Sigma^*$ akzeptiert, dann existiert mindestens eine akzeptierende Berechnung. M kann auch als PTM mit „sehr hoher“ Fehlerwahrscheinlichkeit interpretiert werden.

Satz 25. $\mathcal{ZPP} = \mathcal{RP} \cap$ *co-*$\mathcal{RP}$

Beweisidee: „$\supseteq$: “ Angenommen $L \in \mathcal{ZPP}$. Immer, wenn die $\mathcal{ZPP}$ TM „?“ ausgibt, dann ersetze die „?“ Antwort durch „Ja“ . Daraus folgt $L \in \mathcal{RP}$. Da $\mathcal{ZPP}$ unter Komplement abgeschlossen, gilt $L \in$ co-$\mathcal{RP}$.
„$\subseteq$: “Sei M die $\mathcal{RP}$ TM und M' die co-$\mathcal{RP}$ TM. Wir konstruieren nun eine $\mathcal{ZPP}$ TM, indem wir die Eingabe $w \in \Sigma^*$ für M'' auf M und M' ausführen. Wenn $M(w) = 0$, dann $M''(w) = 0$ und wenn $M'(w) = 1$, dann $M''(w) = 1$. Sonst gebe „?“ aus. M'' macht keinen Fehler, weil M für zuverlässig für $w \in L(M'')$ ist und M' zuverlässig für $w \notin L(M'')$ ist.

Folgerung 26. $\mathcal{ZPP} \subseteq \mathcal{BPP}$

Folgt aus **Satz 22**, **Satz 23** und **Satz 25**

6 Literaturhinweise

Sipser, Michael: Introduction to the Theory of Computation. International ed of 2nd revised ed. Boston: Thomson Course Technology, 2006.

Arora, Sanjeev ; Barak, Boaz: Computational Complexity : A Modern Approach. Cambridge: Cambridge University Press, 2009.

Goldreich, Oded: Computational Complexity : A Conceptual Perspective. Cambridge: Cambridge University Press, 2008.

Dietzfelbinger, Martin: Primality Testing in Polynomial Time : From Randomized Algorithms to "PRIMES Is in P". Berlin, Heidelberg: Springer, 2004.

Papadimitriou, Christos H.: Computational Complexity. New.. Amsterdam: Addison-Wesley, 1994.

Thomas H Cormen; Charles E Leiserson; Ronald Rivest; Clifford Stein: Introduction to Algorithms. Cambridge: MIT Press, 2009.

http://theory.stanford.edu/~valiant/teaching/CS265_2015/primality.pdf (Letzter Zugriff: 31.01.2016 20:30)

https://primes.utm.edu/prove/index.html (Letzter Zugriff: 31.01.2016 20:31)

https://complexityzoo.uwaterloo.ca/Complexity_Zoo (Letzter Zugriff: 31.01.2016 20:34)