Script generated by TTT

Title: Nipkow: Theo (01.07.2019)

Date: Mon Jul 01 14:16:33 CEST 2019

Duration: 80:10 min

Pages: 112

6. Komplexitätstheorie

- Was ist mit beschränkten Mitteln (Zeit&Platz) berechenbar?
- Wieviel Zeit&Platz braucht man,
  um ein bestimmtes Problem/Sprache zu entscheiden?
- **Komplexität eines Problems**, nicht eines Algorithmus!

Polynomielle und exponentielle Komplexität
Taktfrequenz: 1GHz

	Problemgröße n					
Zeit	10	20	30	40	50	60
n	.01 ms	.02 ms	.03 ms	.04 ms	.05 ms	.06 ms
n^2	.1 ms	.4 ms	.9 ms	1.6 ms	2.5 ms	3.6 ms
n^5	100ms	0.003s	0.02s	0.1 s	0.3 s	0.7 s
2^n	1 ms	0.001s	1 s	18 m	13 T	36 J
3^n	59 ms	3 s	2 T	385J	$2 \cdot 10^7$J	10^{12}J

ms=Mikrosek., s=Sek., m=Minute, T=Tag, J=Jahr

Polynomielle und exponentielle Komplexität
Taktfrequenz: 1GHz

	Problemgröße n					
Zeit	10	20	30	40	50	60
n	.01 ms	.02 ms	.03 ms	.04 ms	.05 ms	.06 ms
n^2	.1 ms	.4 ms	.9 ms	1.6 ms	2.5 ms	3.6 ms
n^5	100ms	0.003s	0.02s	0.1 s	0.3 s	0.7 s
2^n	1 ms	0.001s	1 s	18 m	13 T	36 J
3^n	59 ms	3 s	2 T	385J	$2 \cdot 10^7$J	10^{12}J

ms=Mikrosek., s=Sek., m=Minute, T=Tag, J=Jahr

Problemgröße lösbar in fester Zeit: Speedup

Komplexität	n	n^2	n^5	2^n	3^n
1 MHz Prozessor	N_1	N_2	N_3	N_4	N_5
1 GHz Prozessor	1000 N_1	32 N_2	4 N_3	$N_4 + 10$	$N_5 + 6$

Zentrale Frage: Für welche Probleme gibt es/gibt es keine polynomielle Algorithmen?

Zentrale Frage: Für welche Probleme gibt es/gibt es keine polynomielle Algorithmen?

Komplexitätsklasse P:

P = die von DTM in polynomieller Zeit lösbaren Probleme

Zentrale Frage: Für welche Probleme gibt es/gibt es keine polynomielle Algorithmen?

Komplexitätsklasse P:

P = die von DTM in polynomieller Zeit lösbaren Probleme
= die „leichten“ Probleme (*feasible* problems)

Zentrale Frage: Für welche Probleme gibt es/gibt es keine polynomielle Algorithmen?

Komplexitätsklasse P:

P = die von DTM in polynomieller Zeit lösbaren Probleme
= die „leichten“ Probleme (*feasible* problems)

- $A \in P$ wird durch Angabe eines Algorithmus bewiesen

Zentrale Frage: Für welche Probleme gibt es/gibt es keine polynomielle Algorithmen?

Komplexitätsklasse P:

P = die von DTM in polynomieller Zeit lösbaren Probleme
= die „leichten“ Probleme (*feasible* problems)

- $A \in P$ wird durch Angabe eines Algorithmus bewiesen
  Bsp: CYK beweist

$$\{(G, w) \mid G \in \text{CFG}, w \in L(G)\} \in \text{P}$$

Zentrale Frage: Für welche Probleme gibt es/gibt es keine polynomielle Algorithmen?

Komplexitätsklasse P:

P = die von DTM in polynomieller Zeit lösbaren Probleme
= die „leichten“ Probleme (*feasible* problems)

- $A \in P$ wird durch Angabe eines Algorithmus bewiesen
  Bsp: CYK beweist

$$\{(G, w) \mid G \in \text{CFG}, w \in L(G)\} \in \text{P}$$

- $A \notin P$ zu zeigen ist viel schwieriger!

Zentrale Frage: Für welche Probleme gibt es/gibt es keine polynomielle Algorithmen?

Komplexitätsklasse P:

P = die von DTM in polynomieller Zeit lösbaren Probleme
= die „leichten“ Probleme (*feasible* problems)

- $A \in P$ wird durch Angabe eines Algorithmus bewiesen
  Bsp: CYK beweist

$$\{(G, w) \mid G \in \text{CFG}, w \in L(G)\} \in \text{P}$$

- $A \notin P$ zu zeigen ist viel schwieriger!
  Für sehr viele Probleme ist es nicht bekannt,
  ob sie zu P gehören oder nicht.

Viele der wichtigsten Problem der Informatik sind der Gestalt:

Gegeben X, gibt es Y mit $R(X, Y)$?

Viele der wichtigsten Problem der Informatik sind der Gestalt:

Gegeben X, gibt es Y mit $R(X,Y)$?

- SAT: X = Boole'sche Formel, Y = Belegung der Variablen von X, $R(X,Y) :=$ „Y erfüllt X“.

Viele der wichtigsten Problem der Informatik sind der Gestalt:

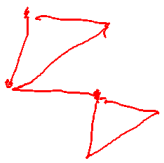
Gegeben X, gibt es Y mit $R(X,Y)$?

- SAT: X = Boole'sche Formel, Y = Belegung der Variablen von X, $R(X,Y) :=$ „Y erfüllt X“.
- HAMILTONKREIS: X = Graph, Y = Kreis von X, $R(X,Y) :=$ „Y besucht alle Knoten von X genau einmal“.

Viele der wichtigsten Problem der Informatik sind der Gestalt:

Gegeben X, gibt es Y mit $R(X,Y)$?

- SAT: X = Boole'sche Formel, Y = Belegung der Variablen von X, $R(X,Y) :=$ „Y erfüllt X“.
- HAMILTONKREIS: X = Graph, Y = Kreis von X, $R(X,Y) :=$ „Y besucht alle Knoten von X genau einmal“.
- EULERKREIS: X = Graph, Y = Kreis von X, $R(X,Y) :=$ „Y besucht alle Kanten von X genau einmal“.



Viele der wichtigsten Problem der Informatik sind der Gestalt:

Gegeben X, gibt es Y mit $R(X,Y)$?

- SAT: X = Boole'sche Formel, Y = Belegung der Variablen von X, $R(X,Y) :=$ „Y erfüllt X“.
- HAMILTONKREIS: X = Graph, Y = Kreis von X, $R(X,Y) :=$ „Y besucht alle Knoten von X genau einmal“.
- EULERKREIS: X = Graph, Y = Kreis von X, $R(X,Y) :=$ „Y besucht alle Kanten von X genau einmal“.

Die Y sind „Lösungskandidaten“. In allen diesen Problemen:

Viele der wichtigsten Problem der Informatik sind der Gestalt:

Gegeben X, gibt es Y mit $R(X,Y)$?

- SAT: X = Boole'sche Formel, Y = Belegung der Variablen von X, $R(X,Y) :=$ „Y erfüllt X“.
- HAMILTONKREIS: X = Graph, Y = Kreis von X, $R(X,Y) :=$ „Y besucht alle Knoten von X genau einmal“.
- EULERKREIS: X = Graph, Y = Kreis von X, $R(X,Y) :=$ „Y besucht alle Kanten von X genau einmal“.

Die Y sind „Lösungskandidaten“. In allen diesen Problemen:

- Prüfen, ob ein Kandidat tatsächlich eine Lösung ist, ist in P.

Viele der wichtigsten Problem der Informatik sind der Gestalt:

Gegeben X, gibt es Y mit $R(X,Y)$?

- SAT: X = Boole'sche Formel, Y = Belegung der Variablen von X, $R(X,Y) :=$ „Y erfüllt X“.
- HAMILTONKREIS: X = Graph, Y = Kreis von X, $R(X,Y) :=$ „Y besucht alle Knoten von X genau einmal“.
- EULERKREIS: X = Graph, Y = Kreis von X, $R(X,Y) :=$ „Y besucht alle Kanten von X genau einmal“.

Die Y sind „Lösungskandidaten“. In allen diesen Problemen:

- Prüfen, ob ein Kandidat tatsächlich eine Lösung ist, ist in P.
- Es gibt jedoch exponentiell viele Kandidaten.

Viele der wichtigsten Problem der Informatik sind der Gestalt:

Gegeben X, gibt es Y mit $R(X,Y)$?

- SAT: X = Boole'sche Formel, Y = Belegung der Variablen von X, $R(X,Y) :=$ „Y erfüllt X“.
- HAMILTONKREIS: X = Graph, Y = Kreis von X, $R(X,Y) :=$ „Y besucht alle Knoten von X genau einmal“.
- EULERKREIS: X = Graph, Y = Kreis von X, $R(X,Y) :=$ „Y besucht alle Kanten von X genau einmal“.

Die Y sind „Lösungskandidaten“. In allen diesen Problemen:

- Prüfen, ob ein Kandidat tatsächlich eine Lösung ist, ist in P.
- Es gibt jedoch exponentiell viele Kandidaten.
- Deshalb hat ein naïver Suchalgorithmus, der alle Kandidaten aufzählt und prüft, exponentielle Laufzeit.

Für kein solches Problem ist bewiesen worden, dass es *nicht* in P liegt.

Für kein solches Problem ist bewiesen worden, dass es *nicht* in P liegt.

Die Frage, ob *alle* solche Probleme in P liegen, ist die wichtigste offene Frage der Informatik.

Für kein solches Problem ist bewiesen worden, dass es *nicht* in P liegt.

Die Frage, ob *alle* solche Probleme in P liegen, ist die wichtigste offene Frage der Informatik.

Äquivalente Formulierung

Diese Probleme können nichtdeterministisch in polynomieller Zeit gelöst werden: Wähle einen Kandidaten und prüfe, ob er eine Lösung ist.

Für kein solches Problem ist bewiesen worden, dass es *nicht* in P liegt.

Die Frage, ob *alle* solche Probleme in P liegen, ist die wichtigste offene Frage der Informatik.

Äquivalente Formulierung

Diese Probleme können nichtdeterministisch in polynomieller Zeit gelöst werden: Wähle einen Kandidaten und prüfe, ob er eine Lösung ist.

Komplexitätsklasse NP:

NP = die von NTM in polynomieller Zeit lösbaren Probleme

Für kein solches Problem ist bewiesen worden, dass es *nicht* in P liegt.

Die Frage, ob *alle* solche Probleme in P liegen, ist die wichtigste offene Frage der Informatik.

Äquivalente Formulierung

Diese Probleme können nichtdeterministisch in polynomieller Zeit gelöst werden: Wähle einen Kandidaten und prüfe, ob er eine Lösung ist.

Komplexitätsklasse NP:

NP = die von NTM in polynomieller Zeit lösbaren Probleme

Zentrale Frage:

$P \neq NP$?

Überblick:

1. Die Komplexitätsklassen P und NP

Überblick:

1. Die Komplexitätsklassen P und NP
2. NP-Vollständigkeit

Überblick:

1. Die Komplexitätsklassen P und NP
2. NP-Vollständigkeit
3. SAT: Ein NP-vollständiges Problem

Überblick:

1. Die Komplexitätsklassen P und NP
2. NP-Vollständigkeit
3. SAT: Ein NP-vollständiges Problem
4. Weitere NP-vollständige Probleme

6.1 Die Komplexitätsklasse P

Berechnungsmodell:

DTM = deterministische Mehrband-TM

6.1 Die Komplexitätsklasse P

Berechnungsmodell:

DTM = deterministische Mehrband-TM

Definition 6.1

$time_M(w)$:= Anzahl der Schritte bis die DTM $M[w]$ hält
$\in \mathbb{N} \cup \{\infty\}$

6.1 Die Komplexitätsklasse P

Berechnungsmodell:

DTM = deterministische Mehrband-TM

Definition 6.1

$time_M(w)$:= Anzahl der Schritte bis die DTM $M[w]$ hält
$\in \mathbb{N} \cup \{\infty\}$

Sei $f : \mathbb{N} \to \mathbb{N}$ eine totale Funktion.
Klasse der in Zeit $f(n)$ entscheidbaren Sprachen:

$$TIME(f(n)) := \{A \subseteq \Sigma^* \mid \exists \text{DTM } M.\ A = L(M) \wedge \forall w \in \Sigma^*.\ time_M(w) \leq f(|w|)\}$$

6.1 Die Komplexitätsklasse P

Berechnungsmodell:

DTM = deterministische Mehrband-TM

Definition 6.1

$time_M(w)$:= Anzahl der Schritte bis die DTM $M[w]$ hält
$\in \mathbb{N} \cup \{\infty\}$

Sei $f : \mathbb{N} \to \mathbb{N}$ eine totale Funktion.
Klasse der in Zeit $f(n)$ entscheidbaren Sprachen:

$$TIME(f(n)) := \{A \subseteq \Sigma^* \mid \exists \text{DTM } M.\ A = L(M) \wedge \forall w \in \Sigma^*.\ time_M(w) \leq f(|w|)\}$$

Merke:

- n ist implizit die Länge der Eingabe

6.1 Die Komplexitätsklasse P

Berechnungsmodell:

DTM = deterministische Mehrband-TM

Definition 6.1

$time_M(w)$:= Anzahl der Schritte bis die DTM $M[w]$ hält
$\in \mathbb{N} \cup \{\infty\}$

Sei $f : \mathbb{N} \to \mathbb{N}$ eine totale Funktion.
Klasse der in Zeit $f(n)$ entscheidbaren Sprachen:

$$TIME(f(n)) := \{A \subseteq \Sigma^* \mid \exists \text{DTM } M.\ A = L(M) \wedge \forall w \in \Sigma^*.\ time_M(w) \leq f(|w|)\}$$

Merke:

- n ist implizit die Länge der Eingabe
- Die DTM entscheidet die Sprache A in $\leq f(n)$ Schritten

Wir betrachten nur Polynome mit Koeffizienten $a_k, \ldots a_0 \in \mathbb{N}$:

$$p(n) = a_k n^k + \cdots + a_1 n + a_0$$

Wir betrachten nur Polynome mit Koeffizienten $a_k, \ldots a_0 \in \mathbb{N}$:

$$p(n) = a_k n^k + \cdots + a_1 n + a_0$$

Definition 6.2

$$\mathsf{P} := \bigcup_{p\ Polynom} TIME(p(n))$$

Wir betrachten nur Polynome mit Koeffizienten $a_k, \ldots a_0 \in \mathbb{N}$:

$$p(n) = a_k n^k + \cdots + a_1 n + a_0$$

Definition 6.2

$$\mathsf{P} := \bigcup_{p\ Polynom} TIME(p(n))$$

Damit gilt auch

$$\mathsf{P} = \bigcup_{k \geq 0} TIME(O(n^k))$$

Wir betrachten nur Polynome mit Koeffizienten $a_k, \dots a_0 \in \mathbb{N}$:

$$p(n) = a_k n^k + \cdots + a_1 n + a_0$$

Definition 6.2

$$\mathsf{P} := \bigcup_{p \ Polynom} \mathit{TIME}(p(n))$$

Damit gilt auch

$$\mathsf{P} = \bigcup_{k \geq 0} \mathit{TIME}(O(n^k))$$

wobei

$$\mathit{TIME}(O(f)) := \bigcup_{g \in O(f)} \mathit{TIME}(g)$$

Beispiel 6.3

- $\{ww^R \mid w \in \Sigma^*\} \in \mathit{TIME}(O(n^2)) \subseteq \mathsf{P}$

Beispiel 6.3

- $\{ww^R \mid w \in \Sigma^*\} \in \mathit{TIME}(O(n^2)) \subseteq \mathsf{P}$
- $\{(G, w) \mid G \text{ ist CFG} \wedge w \in L(G)\} \in \mathsf{P}$

CYK
$\in TIME(O(n^3))$

Beispiel 6.3

- $\{ww^R \mid w \in \Sigma^*\} \in \mathit{TIME}(O(n^2)) \subseteq \mathsf{P}$
- $\{(G, w) \mid G \text{ ist CFG} \wedge w \in L(G)\} \in \mathsf{P}$
- $\{(G, w) \mid G \text{ ist Graph} \wedge w \text{ ist Pfad in } G\} \in \mathsf{P}$

Beispiel 6.3

- $\{ww^R \mid w \in \Sigma^*\} \in \mathit{TIME}(O(n^2)) \subseteq \mathsf{P}$
- $\{(G, w) \mid G \text{ ist CFG} \wedge w \in L(G)\} \in \mathsf{P}$
- $\{(G, w) \mid G \text{ ist Graph} \wedge w \text{ ist Pfad in } G\} \in \mathsf{P}$
- $\{bin(p) \mid p \text{ Primzahl}\} \in \mathsf{P}$

Beispiel 6.3

- $\{ww^R \mid w \in \Sigma^*\} \in \mathit{TIME}(O(n^2)) \subseteq \mathsf{P}$
- $\{(G, w) \mid G \text{ ist CFG} \wedge w \in L(G)\} \in \mathsf{P}$
- $\{(G, w) \mid G \text{ ist Graph} \wedge w \text{ ist Pfad in } G\} \in \mathsf{P}$
- $\{bin(p) \mid p \text{ Primzahl}\} \in \mathsf{P}$
- $\{(G, w) \mid G \text{ ist Graph} \wedge w \text{ ist Eulerkreis in } G\} \in \mathsf{P}$

$(a,b), (c,d), \ldots, (x,a)$

Beispiel 6.3

- $\{ww^R \mid w \in \Sigma^*\} \in \mathit{TIME}(O(n^2)) \subseteq \mathsf{P}$
- $\{(G, w) \mid G \text{ ist CFG} \wedge w \in L(G)\} \in \mathsf{P}$
- $\{(G, w) \mid G \text{ ist Graph} \wedge w \text{ ist Pfad in } G\} \in \mathsf{P}$
- $\{bin(p) \mid p \text{ Primzahl}\} \in \mathsf{P}$
- $\{(G, w) \mid G \text{ ist Graph} \wedge w \text{ ist Eulerkreis in } G\} \in \mathsf{P}$

Beweis durch Angabe eines Algorithmus mit Komplexität $O(n^k)$ für ein $k \geq 1$.

Beispiel 6.3

- $\{ww^R \mid w \in \Sigma^*\} \in \mathit{TIME}(O(n^2)) \subseteq \mathsf{P}$
- $\{(G, w) \mid G \text{ ist CFG} \wedge w \in L(G)\} \in \mathsf{P}$
- $\{(G, w) \mid G \text{ ist Graph} \wedge w \text{ ist Pfad in } G\} \in \mathsf{P}$
- $\{bin(p) \mid p \text{ Primzahl}\} \in \mathsf{P}$
- $\{(G, w) \mid G \text{ ist Graph} \wedge w \text{ ist Eulerkreis in } G\} \in \mathsf{P}$

Beweis durch Angabe eines Algorithmus mit Komplexität $O(n^k)$ für ein $k \geq 1$.

Bemerkungen

- $O(n \log n) \subset O(n^2)$

Beispiel 6.3

- $\{ww^R \mid w \in \Sigma^*\} \in \mathit{TIME}(O(n^2)) \subseteq \mathsf{P}$
- $\{(G, w) \mid G \text{ ist CFG} \wedge w \in L(G)\} \in \mathsf{P}$
- $\{(G, w) \mid G \text{ ist Graph} \wedge w \text{ ist Pfad in } G\} \in \mathsf{P}$
- $\{bin(p) \mid p \text{ Primzahl}\} \in \mathsf{P}$
- $\{(G, w) \mid G \text{ ist Graph} \wedge w \text{ ist Eulerkreis in } G\} \in \mathsf{P}$

Beweis durch Angabe eines Algorithmus mit Komplexität $O(n^k)$ für ein $k \geq 1$.

Bemerkungen

- $O(n \log n) \subset O(n^2)$
- $n^{\log n}, 2^n \notin O(n^k)$ für alle k

Beispiel 6.3

- $\{ww^R \mid w \in \Sigma^*\} \in \mathit{TIME}(O(n^2)) \subseteq \mathsf{P}$
- $\{(G, w) \mid G \text{ ist CFG} \wedge w \in L(G)\} \in \mathsf{P}$
- $\{(G, w) \mid G \text{ ist Graph} \wedge w \text{ ist Pfad in } G\} \in \mathsf{P}$
- $\{bin(p) \mid p \text{ Primzahl}\} \in \mathsf{P}$
- $\{(G, w) \mid G \text{ ist Graph} \wedge w \text{ ist Eulerkreis in } G\} \in \mathsf{P}$

Beweis durch Angabe eines Algorithmus mit Komplexität $O(n^k)$ für ein $k \geq 1$.

Bemerkungen

- $O(n \log n) \subset O(n^2)$
- $n^{\log n}, 2^n \notin O(n^k)$ für alle k

Analog zu Sprachen nennen wir eine Funktion $f : \mathbb{N} \to \mathbb{N}$ in polynomieller Zeit berechenbar, gdw es eine DTM M gibt, die f berechnet und $\mathit{time}_M(w) \leq p(|w|)$ für ein Polynom p.

- Warum P und nicht (zB) $O(n^{17})$?

- Warum P und nicht (zB) $O(n^{17})$?
  Um robust bzgl Maschinenmodell zu sein:
  1-Band DTM braucht $O(t^2)$ Schritte
  um t Schritte einer k-Band DTM zu simulieren.

- Warum P und nicht (zB) $O(n^{17})$?
  Um robust bzgl Maschinenmodell zu sein:
  1-Band DTM braucht $O(t^2)$ Schritte
  um t Schritte einer k-Band DTM zu simulieren.

  Fast alle bekannten „vernünftigen“ Maschinenmodelle lassen sich von einer DTM in polynomieller Zeit simulieren.

- Warum P und nicht (zB) $O(n^{17})$?
  Um robust bzgl Maschinenmodell zu sein:
  1-Band DTM braucht $O(t^2)$ Schritte
  um t Schritte einer k-Band DTM zu simulieren.

  Fast alle bekannten „vernünftigen“ Maschinenmodelle lassen sich von einer DTM in polynomieller Zeit simulieren.

  Offen: Quantencomputer

- Warum P und nicht (zB) $O(n^{17})$?
  Um robust bzgl Maschinenmodell zu sein:
  1-Band DTM braucht $O(t^2)$ Schritte
  um t Schritte einer k-Band DTM zu simulieren.

  Fast alle bekannten „vernünftigen“ Maschinenmodelle lassen sich von einer DTM in polynomieller Zeit simulieren.

  Offen: Quantencomputer

- Warum TM?

- Warum P und nicht (zB) $O(n^{17})$?
  Um robust bzgl Maschinenmodell zu sein:
  1-Band DTM braucht $O(t^2)$ Schritte
  um t Schritte einer k-Band DTM zu simulieren.

  Fast alle bekannten „vernünftigen“ Maschinenmodelle lassen sich von einer DTM in polynomieller Zeit simulieren.

  Offen: Quantencomputer

- Warum TM?
  Historisch. Ebenfalls möglich: (zB) WHILE.
  Aber zwei mögliche Kostenmodelle:

  Uniform x_i := x_j + n kostet 1 Zeiteinheit.

- Warum P und nicht (zB) $O(n^{17})$?
  Um robust bzgl Maschinenmodell zu sein:
  1-Band DTM braucht $O(t^2)$ Schritte
  um t Schritte einer k-Band DTM zu simulieren.

  Fast alle bekannten „vernünftigen“ Maschinenmodelle lassen sich von einer DTM in polynomieller Zeit simulieren.

  Offen: Quantencomputer

- Warum TM?
  Historisch. Ebenfalls möglich: (zB) WHILE.
  Aber zwei mögliche Kostenmodelle:

  Uniform x_i := x_j + n kostet 1 Zeiteinheit.
  Logarithmisch x_i := x_j + n kostet $\log x_j$ Zeiteinheiten.

6.2 Die Komplexitätsklasse NP

6.2 Die Komplexitätsklasse NP

Berechnungsmodell:

NTM = nichtdeterministische Mehrband-TM

- NP ist die Klasse der Sprachen, die von einer NTM in polynomieller Zeit akzeptiert werden.
- Dh: Eine Sprache A liegt in NP gdw
  es ein Polynom $p(n)$ und eine NTM M gibt, so dass

Definition 6.4

$$ntime_M(w) := \begin{cases} \text{minimale Anzahl der Schritte bis NTM } M[w] \text{ akzeptiert} & \text{falls } w \in L(M) \\ 0 & \text{falls } w \notin L(M) \end{cases}$$

Definition 6.4

$$\mathit{ntime}_M(w) := \begin{cases} \text{minimale Anzahl der Schritte} \\ \text{bis NTM } M[w] \text{ akzeptiert} & \text{falls } w \in L(M) \\ 0 & \text{falls } w \notin L(M) \end{cases}$$

Sei $f : \mathbb{N} \to \mathbb{N}$ eine totale Funktion.

$$\mathit{NTIME}(f(n)) := \{A \subseteq \Sigma^* \mid \exists \text{NTM } M.\ A = L(M) \wedge \forall w \in \Sigma^*.\ \mathit{ntime}_M(w) \leq f(|w|)\}$$

Definition 6.4

$$\mathit{ntime}_M(w) := \begin{cases} \text{minimale Anzahl der Schritte} \\ \text{bis NTM } M[w] \text{ akzeptiert} & \text{falls } w \in L(M) \\ 0 & \text{falls } w \notin L(M) \end{cases}$$

Sei $f : \mathbb{N} \to \mathbb{N}$ eine totale Funktion.

$$\mathit{NTIME}(f(n)) := \{A \subseteq \Sigma^* \mid \exists \text{NTM } M.\ A = L(M) \wedge \forall w \in \Sigma^*.\ \mathit{ntime}_M(w) \leq f(|w|)\}$$

$$\text{NP} := \bigcup_{p\ \mathit{Polynom}} \mathit{NTIME}(p(n))$$

Definition 6.4

$$\mathit{ntime}_M(w) := \begin{cases} \text{minimale Anzahl der Schritte} \\ \text{bis NTM } M[w] \text{ akzeptiert} & \text{falls } w \in L(M) \\ 0 & \text{falls } w \notin L(M) \end{cases}$$

Sei $f : \mathbb{N} \to \mathbb{N}$ eine totale Funktion.

$$\mathit{NTIME}(f(n)) := \{A \subseteq \Sigma^* \mid \exists \text{NTM } M.\ A = L(M) \wedge \forall w \in \Sigma^*.\ \mathit{ntime}_M(w) \leq f(|w|)\}$$

$$\text{NP} := \bigcup_{p\ \mathit{Polynom}} \mathit{NTIME}(p(n))$$

Bemerkungen:

- P $\subseteq$ NP

Definition 6.4

$$\mathit{ntime}_M(w) := \begin{cases} \text{minimale Anzahl der Schritte} \\ \text{bis NTM } M[w] \text{ akzeptiert} & \text{falls } w \in L(M) \\ 0 & \text{falls } w \notin L(M) \end{cases}$$

Sei $f : \mathbb{N} \to \mathbb{N}$ eine totale Funktion.

$$\mathit{NTIME}(f(n)) := \{A \subseteq \Sigma^* \mid \exists \text{NTM } M.\ A = L(M) \wedge \forall w \in \Sigma^*.\ \mathit{ntime}_M(w) \leq f(|w|)\}$$

$$\text{NP} := \bigcup_{p\ \mathit{Polynom}} \mathit{NTIME}(p(n))$$

Bemerkungen:

- P $\subseteq$ NP
- Seit etwa 1970 ist offen ob P = NP.

Definition 6.4

$$\mathit{ntime}_M(w) := \begin{cases} \text{minimale Anzahl der Schritte bis NTM } M[w] \text{ akzeptiert} & \text{falls } w \in L(M) \\ 0 & \text{falls } w \notin L(M) \end{cases}$$

Sei $f : \mathbb{N} \to \mathbb{N}$ eine totale Funktion.

$$\mathit{NTIME}(f(n)) := \{A \subseteq \Sigma^* \mid \exists \text{NTM } M.\ A = L(M) \wedge \forall w \in \Sigma^*.\ \mathit{ntime}_M(w) \leq f(|w|)\}$$

$$\text{NP} := \bigcup_{p\ \mathit{Polynom}} \mathit{NTIME}(p(n))$$

Bemerkungen:

- P $\subseteq$ NP
- Seit etwa 1970 ist offen ob P $=$ NP.

Bemerkungen zur Definition von NP:

Bemerkungen zur Definition von NP:

Akzeptierende NTM M

- braucht für $w \notin L(M)$ nicht zu halten.
- kann für $w \in L(M)$ auch beliebig lange Berechnungsfolgen haben.

Äquivalente Definition NP′ von NP:
Die NTM $M[w]$ *muss* nach maximal $p(|w|)$ Schritten halten.

Bemerkungen zur Definition von NP:

Akzeptierende NTM M

- braucht für $w \notin L(M)$ nicht zu halten.
- kann für $w \in L(M)$ auch beliebig lange Berechnungsfolgen haben.

Äquivalente Definition NP′ von NP:
Die NTM $M[w]$ *muss* nach maximal $p(|w|)$ Schritten halten.

NP′ $\subseteq$ NP: Klar.

Bemerkungen zur Definition von NP:

Akzeptierende NTM M

- braucht für $w \notin L(M)$ nicht zu halten.
- kann für $w \in L(M)$ auch beliebig lange Berechnungsfolgen haben.

Äquivalente Definition NP′ von NP:
Die NTM $M[w]$ *muss* nach maximal $p(|w|)$ Schritten halten.

NP′ $\subseteq$ NP: Klar.

NP $\subseteq$ NP′:

Bemerkungen zur Definition von NP:

Akzeptierende NTM M

- braucht für $w \notin L(M)$ nicht zu halten.
- kann für $w \in L(M)$ auch beliebig lange Berechnungsfolgen haben.

Äquivalente Definition NP′ von NP:
Die NTM $M[w]$ *muss* nach maximal $p(|w|)$ Schritten halten.

NP′ $\subseteq$ NP: Klar.

NP $\subseteq$ NP′: Falls $A \in$ NP mit Polynom p und NTM M,

Bemerkungen zur Definition von NP:

Akzeptierende NTM M

- braucht für $w \notin L(M)$ nicht zu halten.
- kann für $w \in L(M)$ auch beliebig lange Berechnungsfolgen haben.

Äquivalente Definition NP′ von NP:
Die NTM $M[w]$ *muss* nach maximal $p(|w|)$ Schritten halten.

NP′ $\subseteq$ NP: Klar.

NP $\subseteq$ NP′: Falls $A \in$ NP mit Polynom p und NTM M,
so kann man NTM M' konstruieren mit $L(M') = A$,

Bemerkungen zur Definition von NP:

Akzeptierende NTM M

- braucht für $w \notin L(M)$ nicht zu halten.
- kann für $w \in L(M)$ auch beliebig lange Berechnungsfolgen haben.

Äquivalente Definition NP′ von NP:
Die NTM $M[w]$ *muss* nach maximal $p(|w|)$ Schritten halten.

NP′ $\subseteq$ NP: Klar.

NP $\subseteq$ NP′: Falls $A \in$ NP mit Polynom p und NTM M,
so kann man NTM M' konstruieren mit $L(M') = A$,
so dass $M'[w]$ immer innerhalb von polynomieller Zeit hält. $p(|w|)$

Bemerkungen zur Definition von NP:

Akzeptierende NTM M

- braucht für $w \notin L(M)$ nicht zu halten.
- kann für $w \in L(M)$ auch beliebig lange Berechnunsgfolgen haben.

Äquivalente Definition NP′ von NP:
Die NTM $M[w]$ *muss* nach maximal $p(|w|)$ Schritten halten.

NP′ $\subseteq$ NP: Klar.

NP $\subseteq$ NP′: Falls $A \in$ NP mit Polynom p und NTM M,
so kann man NTM M' konstruieren mit $L(M') = A$,
so dass $M'[w]$ immer innerhalb von polynomieller Zeit hält.

1. Eingabe w

Bemerkungen zur Definition von NP:

Akzeptierende NTM M

- braucht für $w \notin L(M)$ nicht zu halten.
- kann für $w \in L(M)$ auch beliebig lange Berechnunsgfolgen haben.

Äquivalente Definition NP′ von NP:
Die NTM $M[w]$ *muss* nach maximal $p(|w|)$ Schritten halten.

NP′ $\subseteq$ NP: Klar.

NP $\subseteq$ NP′: Falls $A \in$ NP mit Polynom p und NTM M,
so kann man NTM M' konstruieren mit $L(M') = A$,
so dass $M'[w]$ immer innerhalb von polynomieller Zeit hält.

1. Eingabe w
2. Schreibe $p(|w|)$ auf ein getrenntes Band („timer“)
3. Simuliere $M[w]$, aber dekrementiere timer nach jedem Schritt.

Bemerkungen zur Definition von NP:

Akzeptierende NTM M

- braucht für $w \notin L(M)$ nicht zu halten.
- kann für $w \in L(M)$ auch beliebig lange Berechnunsgfolgen haben.

Äquivalente Definition NP′ von NP:
Die NTM $M[w]$ *muss* nach maximal $p(|w|)$ Schritten halten.

NP′ $\subseteq$ NP: Klar.

NP $\subseteq$ NP′: Falls $A \in$ NP mit Polynom p und NTM M,
so kann man NTM M' konstruieren mit $L(M') = A$,
so dass $M'[w]$ immer innerhalb von polynomieller Zeit hält.

1. Eingabe w
2. Schreibe $p(|w|)$ auf ein getrenntes Band („timer“)
3. Simuliere $M[w]$, aber dekrementiere timer nach jedem Schritt.
4. Wird timer=0, ohne dass M gehalten hat, halte in einem nicht-Endzustand („timeout“)

Beispiel 6.5

- Ein Euler-Kreis ist ein geschlossener Pfad in einem (ungerichteten) Graphen, der jede Kante genau einmal enthält.

Bemerkungen zur Definition von NP:

Beispiel 6.5

- Ein Euler-Kreis ist ein geschlossener Pfad in einem (ungerichteten) Graphen, der jede Kante genau einmal enthält.

Beispiel 6.5

- Ein Euler-Kreis ist ein geschlossener Pfad in einem (ungerichteten) Graphen, der jede Kante genau einmal enthält.
- Satz: Ein Graph hat einen Euler-Kreis gdw er zusammenhängend ist, und jeder Knoten geraden Grad hat.

Beispiel 6.5

- Ein Euler-Kreis ist ein geschlossener Pfad in einem (ungerichteten) Graphen, der jede Kante genau einmal enthält.
- Satz: Ein Graph hat einen Euler-Kreis gdw er zusammenhängend ist, und jeder Knoten geraden Grad hat.
- Beide Eigenschaften sind in polynomieller Zeit von einer DTM überprüfbar.

Beispiel 6.5

- Ein Euler-Kreis ist ein geschlossener Pfad in einem (ungerichteten) Graphen, der jede Kante genau einmal enthält.
- Satz: Ein Graph hat einen Euler-Kreis gdw er zusammenhängend ist, und jeder Knoten geraden Grad hat.
- Beide Eigenschaften sind in polynomieller Zeit von einer DTM überprüfbar.
- $\Longrightarrow \{G \mid G \text{ hat Euler-Kreis}\} \in \mathrm{P}$

Beispiel 6.6

- Ein Hamilton-Kreis ist ein geschlossener Pfad in einem (ungerichteten) Graphen, der jeden Knoten genau einmal enthält.

Beispiel 6.6

- Ein Hamilton-Kreis ist ein geschlossener Pfad in einem (ungerichteten) Graphen, der jeden Knoten genau einmal enthält.
- HAMILTON $:= \{G \mid G \text{ hat Hamilton-Kreis}\} \in \mathrm{NP}$ mit folgendem Algorithmus der Art „Rate und prüfe“:

Beispiel 6.6

- Ein Hamilton-Kreis ist ein geschlossener Pfad in einem (ungerichteten) Graphen, der jeden Knoten genau einmal enthält.
- HAMILTON $:= \{G \mid G \text{ hat Hamilton-Kreis}\} \in \mathrm{NP}$ mit folgendem Algorithmus der Art „Rate und prüfe“:
  1. Rate eine Permutation der Knoten des Graphen.

Beispiel 6.6

- Ein Hamilton-Kreis ist ein geschlossener Pfad in einem (ungerichteten) Graphen, der jeden Knoten genau einmal enthält.
- HAMILTON $:= \{G \mid G \text{ hat Hamilton-Kreis}\} \in$ NP mit folgendem Algorithmus der Art „Rate und prüfe“:
  1. Rate eine Permutation der Knoten des Graphen.
  2. Prüfe, ob diese Permutation ein Hamilton-Kreis ist.

Beispiel 6.6

- Ein Hamilton-Kreis ist ein geschlossener Pfad in einem (ungerichteten) Graphen, der jeden Knoten genau einmal enthält.
- HAMILTON $:= \{G \mid G \text{ hat Hamilton-Kreis}\} \in$ NP mit folgendem Algorithmus der Art „Rate und prüfe“:
  1. Rate eine Permutation der Knoten des Graphen.
  2. Prüfe, ob diese Permutation ein Hamilton-Kreis ist.

Das Raten ist in polynomieller Zeit von einer NTM machbar.

329

Erinnerung: Viele Probleme sind von der Art dass

- schwer ist, zu entscheiden, ob sie lösbar sind,

Erinnerung: Viele Probleme sind von der Art dass

- schwer ist, zu entscheiden, ob sie lösbar sind,
- leicht ist, zu entscheiden, ob ein Lösungsvorschlag eine Lösung ist.

330

Erinnerung: Viele Probleme sind von der Art dass

- schwer ist, zu entscheiden, ob sie lösbar sind,
- leicht ist, zu entscheiden, ob ein Lösungsvorschlag eine Lösung ist.

Formalisierung:

Definition 6.7

Sei M eine DTM mit $L(M) \subseteq \{w\#c \mid w \in \Sigma^*, c \in \Delta^*\}$.

- Falls $w\#c \in L(M)$, so heißt c Zertifikat für w.

Erinnerung: Viele Probleme sind von der Art dass

- schwer ist, zu entscheiden, ob sie lösbar sind,
- leicht ist, zu entscheiden, ob ein Lösungsvorschlag eine Lösung ist.

Formalisierung:

Definition 6.7

Sei M eine DTM mit $L(M) \subseteq \{w\#c \mid w \in \Sigma^*, c \in \Delta^*\}$.

- Falls $w\#c \in L(M)$, so heißt c Zertifikat für w.
- M ist ein polynomiell beschränkter Verifikator für die Sprache $\{w \in \Sigma^* \mid \exists c \in \Delta^*.\ w\#c \in L(M)\}$

Erinnerung: Viele Probleme sind von der Art dass

- schwer ist, zu entscheiden, ob sie lösbar sind,
- leicht ist, zu entscheiden, ob ein Lösungsvorschlag eine Lösung ist.

Formalisierung:

Definition 6.7

Sei M eine DTM mit $L(M) \subseteq \{w\#c \mid w \in \Sigma^*, c \in \Delta^*\}$.

- Falls $w\#c \in L(M)$, so heißt c Zertifikat für w.
- M ist ein polynomiell beschränkter Verifikator für die Sprache $\{w \in \Sigma^* \mid \exists c \in \Delta^*.\ w\#c \in L(M)\}$
  falls es ein Polynom p gibt, so dass $\mathit{time}_M(w\#c) \leq p(|w|)$.

Erinnerung: Viele Probleme sind von der Art dass

- schwer ist, zu entscheiden, ob sie lösbar sind,
- leicht ist, zu entscheiden, ob ein Lösungsvorschlag eine Lösung ist.

Formalisierung:

Definition 6.7

Sei M eine DTM mit $L(M) \subseteq \{w\#c \mid w \in \Sigma^*, c \in \Delta^*\}$.

- Falls $w\#c \in L(M)$, so heißt c Zertifikat für w.
- M ist ein polynomiell beschränkter Verifikator für die Sprache $\{w \in \Sigma^* \mid \exists c \in \Delta^*.\ w\#c \in L(M)\}$
  falls es ein Polynom p gibt, so dass $\mathit{time}_M(w\#c) \leq p(|w|)$.

NB:
In Zeit $p(n)$ kann M maximal die ersten $p(n)$ Zeichen von c lesen.

Erinnerung: Viele Probleme sind von der Art dass

- schwer ist, zu entscheiden, ob sie lösbar sind,
- leicht ist, zu entscheiden, ob ein Lösungsvorschlag eine Lösung ist.

Formalisierung:

Definition 6.7
Sei M eine DTM mit $L(M) \subseteq \{w\#c \mid w \in \Sigma^*, c \in \Delta^*\}$.

- Falls $w\#c \in L(M)$, so heißt c Zertifikat für w.
- M ist ein polynomiell beschränkter Verifikator für die Sprache $\{w \in \Sigma^* \mid \exists c \in \Delta^*.\ w\#c \in L(M)\}$
  falls es ein Polynom p gibt, so dass $\mathit{time}_M(w\#c) \leq p(|w|)$.

NB:
In Zeit $p(n)$ kann M maximal die ersten $p(n)$ Zeichen von c lesen.
Daher genügt für w ein Zertifikat der Länge $\leq p(|w|)$.

Erinnerung: Viele Probleme sind von der Art dass

- schwer ist, zu entscheiden, ob sie lösbar sind,
- leicht ist, zu entscheiden, ob ein Lösungsvorschlag eine Lösung ist.

Formalisierung:

Definition 6.7
Sei M eine DTM mit $L(M) \subseteq \{w\#c \mid w \in \Sigma^*, c \in \Delta^*\}$.

- Falls $w\#c \in L(M)$, so heißt c Zertifikat für w.
- M ist ein polynomiell beschränkter Verifikator für die Sprache $\{w \in \Sigma^* \mid \exists c \in \Delta^*.\ w\#c \in L(M)\}$
  falls es ein Polynom p gibt, so dass $\mathit{time}_M(w\#c) \leq p(|w|)$.

NB:
In Zeit $p(n)$ kann M maximal die ersten $p(n)$ Zeichen von c lesen.
Daher genügt für w ein Zertifikat der Länge $\leq p(|w|)$.

Beispiel 6.8 (HAMILTON)

Zertifikat: Knotenpermutation. In polynomieller Zeit verifizierbar.

Satz 6.9
$A \in NP$ gdw es einen polynomiell beschränkten Verifikator für A gibt.

Erinnerung: Viele Probleme sind von der Art dass

- schwer ist, zu entscheiden, ob sie lösbar sind,
- leicht ist, zu entscheiden, ob ein Lösungsvorschlag eine Lösung ist.

Formalisierung:

Definition 6.7
Sei M eine DTM mit $L(M) \subseteq \{w\#c \mid w \in \Sigma^*, c \in \Delta^*\}$.

- Falls $w\#c \in L(M)$, so heißt c Zertifikat für w.
- M ist ein polynomiell beschränkter Verifikator für die Sprache $\{w \in \Sigma^* \mid \exists c \in \Delta^*.\ w\#c \in L(M)\}$
  falls es ein Polynom p gibt, so dass $\mathit{time}_M(w\#c) \leq p(|w|)$.

NB:
In Zeit $p(n)$ kann M maximal die ersten $p(n)$ Zeichen von c lesen.

Bemerkungen zur Definition von NP:

Akzeptierende NTM M

- braucht für $w \notin L(M)$ nicht zu halten.
- kann für $w \in L(M)$ auch beliebig lange Berechnungsfolgen haben.

Erinnerung: Viele Probleme sind von der Art dass

- schwer ist, zu entscheiden, ob sie lösbar sind,
- leicht ist, zu entscheiden, ob ein Lösungsvorschlag eine Lösung ist.

Formalisierung:

Definition 6.7

Sei M eine DTM mit $L(M) \subseteq \{w\#c \mid w \in \Sigma^*, c \in \Delta^*\}$.

- Falls $w\#c \in L(M)$, so heißt c Zertifikat für w.
- M ist ein polynomiell beschränkter Verifikator für die Sprache $\{w \in \Sigma^* \mid \exists c \in \Delta^*.\ w\#c \in L(M)\}$
  falls es ein Polynom p gibt, so dass $\mathit{time}_M(w\#c) \leq p(|w|)$.

NB:
In Zeit $p(n)$ kann M maximal die ersten $p(n)$ Zeichen von c lesen.
Daher genügt für w ein Zertifikat der Länge $\leq p(|w|)$.

Beispiel 6.8 (HAMILTON)

Zertifikat: Knotenpermutation. In polynomieller Zeit verifizierbar.

Satz 6.9

$A \in$ NP gdw es einen polynomiell beschränkten Verifikator für A gibt.

Satz 6.9

$A \in$ NP gdw es einen polynomiell beschränkten Verifikator für A gibt.

Beweis:

„$\Rightarrow$“:

Satz 6.9
$A \in$ NP gdw es einen polynomiell beschränkten Verifikator für A gibt.

Beweis:
„$\Rightarrow$“:
Sei $A \in$ NP.

Satz 6.9
$A \in$ NP gdw es einen polynomiell beschränkten Verifikator für A gibt.

Beweis:
„$\Rightarrow$“:
Sei $A \in$ NP. Dh es gibt NTM N, die A in Zeit $p(n)$ akzeptiert.
Ein Zertifikat für $w \in A$ ist die Folge der benutzten Transitionen $\delta(q, a) \ni (q', a', d)$ einer akzeptierenden Berechnungsfolge von $N[w]$

Satz 6.9
$A \in$ NP gdw es einen polynomiell beschränkten Verifikator für A gibt.

Beweis:
„$\Rightarrow$“:
Sei $A \in$ NP. Dh es gibt NTM N, die A in Zeit $p(n)$ akzeptiert.
Ein Zertifikat für $w \in A$ ist die Folge der benutzten Transitionen $\delta(q, a) \ni (q', a', d)$ einer akzeptierenden Berechnungsfolge von $N[w]$

Satz 6.9
$A \in$ NP gdw es einen polynomiell beschränkten Verifikator für A gibt.

Beweis:
„$\Rightarrow$“:
Sei $A \in$ NP. Dh es gibt NTM N, die A in Zeit $p(n)$ akzeptiert.
Ein Zertifikat für $w \in A$ ist die Folge der benutzten Transitionen $\delta(q, a) \ni (q', a', d)$ einer akzeptierenden Berechnungsfolge von $N[w]$ mit $\leq p(n)$ Schritten.

Satz 6.9

$A \in$ NP gdw es einen polynomiell beschränkten Verifikator für A gibt.

Beweis:

„$\Rightarrow$“:
Sei $A \in$ NP. Dh es gibt NTM N, die A in Zeit $p(n)$ akzeptiert.
Ein Zertifikat für $w \in A$ ist die Folge der benutzten Transitionen $\delta(q, a) \ni (q', a', d)$ einer akzeptierenden Berechnunsgfolge von $N[w]$ mit $\leq p(n)$ Schritten.
Ein polynomiell beschränkter Verfikator für $L(N)$:

Satz 6.9

$A \in$ NP gdw es einen polynomiell beschränkten Verifikator für A gibt.

Beweis:

„$\Rightarrow$“:
Sei $A \in$ NP. Dh es gibt NTM N, die A in Zeit $p(n)$ akzeptiert.
Ein Zertifikat für $w \in A$ ist die Folge der benutzten Transitionen $\delta(q, a) \ni (q', a', d)$ einer akzeptierenden Berechnunsgfolge von $N[w]$ mit $\leq p(n)$ Schritten.
Ein polynomiell beschränkter Verfikator für $L(N)$:

1. Eingabe $w\#c$
2. Simuliere $N[w]$, gesteuert durch die Transitionen in c.
3. Überprüfe dabei, ob die Transition in c jeweils zu N und zur augenblicklichen Konfiguration von N passt.

Satz 6.9

$A \in$ NP gdw es einen polynomiell beschränkten Verifikator für A gibt.

Beweis:

„$\Rightarrow$“:
Sei $A \in$ NP. Dh es gibt NTM N, die A in Zeit $p(n)$ akzeptiert.
Ein Zertifikat für $w \in A$ ist die Folge der benutzten Transitionen $\delta(q, a) \ni (q', a', d)$ einer akzeptierenden Berechnunsgfolge von $N[w]$ mit $\leq p(n)$ Schritten.
Ein polynomiell beschränkter Verfikator für $L(N)$:

1. Eingabe $w\#c$
2. Simuliere $N[w]$, gesteuert durch die Transitionen in c.
3. Überprüfe dabei, ob die Transition in c jeweils zu N und zur augenblicklichen Konfiguration von N passt.
4. Akzeptiere, falls c in einen Endzustand führt.

Beweis (Forts.):

„$\Leftarrow$“:
Sei M ein polynomiell (durch p) beschränkter Verifikator für A.

Beweis (Forts.):

„⇐“:
Sei M ein polynomiell (durch p) beschränkter Verifikator für A.
Wir bauen eine polynomiell beschränkte NTM N mit $L(M) = A$:

Beweis (Forts.):

„⇐“:
Sei M ein polynomiell (durch p) beschränkter Verifikator für A.
Wir bauen eine polynomiell beschränkte NTM N mit $L(M) = A$:

1. Eingabe w.

Beweis (Forts.):

„⇐“:
Sei M ein polynomiell (durch p) beschränkter Verifikator für A.
Wir bauen eine polynomiell beschränkte NTM N mit $L(M) = A$:

1. Eingabe w.
2. Schreibe hinter w zuerst $\#$ und dann ein nichtdeterministisch gewähltes Wort $c \in \Delta^*$, $|c| = p(|w|)$:
   for $i := 1, \ldots, p(|w|)$ **do**
   schreibe ein Zeichen aus Δ und gehe nach rechts

Beweis (Forts.):

„⇐“:
Sei M ein polynomiell (durch p) beschränkter Verifikator für A.
Wir bauen eine polynomiell beschränkte NTM N mit $L(M) = A$:

1. Eingabe w.
2. Schreibe hinter w zuerst $\#$ und dann ein nichtdeterministisch gewähltes Wort $c \in \Delta^*$, $|c| = p(|w|)$:
   for $i := 1, \ldots, p(|w|)$ **do**
   schreibe ein Zeichen aus Δ und gehe nach rechts
3. Führe M aus (mit Eingabe $w\#c$).

Beweis (Forts.):

„⇐“:
Sei M ein polynomiell (durch p) beschränkter Verifikator für A.
Wir bauen eine polynomiell beschränkte NTM N mit $L(M) = A$:

1. Eingabe w.
2. Schreibe hinter w zuerst $\#$ und dann ein nichtdeterministisch gewähltes Wort $c \in \Delta^*$, $|c| = p(|w|)$:
   for $i := 1, \ldots, p(|w|)$ **do**
   schreibe ein Zeichen aus Δ und gehe nach rechts
3. Führe M aus (mit Eingabe $w\#c$).

Nach Annahme gilt $\mathit{time}_M(w\#c) \leq p(|w|)$.

Beweis (Forts.):

„⇐“:
Sei M ein polynomiell (durch p) beschränkter Verifikator für A.
Wir bauen eine polynomiell beschränkte NTM N mit $L(M) = A$:

1. Eingabe w.
2. Schreibe hinter w zuerst $\#$ und dann ein nichtdeterministisch gewähltes Wort $c \in \Delta^*$, $|c| = p(|w|)$:
   for $i := 1, \ldots, p(|w|)$ **do**
   schreibe ein Zeichen aus Δ und gehe nach rechts
3. Führe M aus (mit Eingabe $w\#c$).

Nach Annahme gilt $\mathit{time}_M(w\#c) \leq p(|w|)$.
Damit braucht $N[w]$ $O(p(|w|))$ Schritte. □

332

Fazit:

- P sind die Sprachen, bei denen $w \in L$ schnell entschieden werden kann.
- NP sind die Sprachen, bei denen ein Zertifkat für $w \in L$ schnell verifiziert/überprüft werden kann.

Fazit:

- P sind die Sprachen, bei denen $w \in L$ schnell entschieden werden kann.
- NP sind die Sprachen, bei denen ein Zertifkat für $w \in L$ schnell verifiziert/überprüft werden kann.

Intuition:
Es ist leichter, eine Lösung zu verifizieren als zu finden.

333

6.3 NP-Vollständigkeit