Komplexitätstheorie

SoSe 2019

Jean Christoph Jung, Thomas Schneider

Kapitel 2: Turingmaschinen

Homepage der Vorlesung: http://tinyurl.com/ss19-kt

Einleitung

Wir möchten oft Aussagen über alle Algorithmen treffen
(z. B. es gibt keinen, der ein bestimmtes Problem in Polyzeit löst).

Um in diesem Kontext formale Beweise führen zu können,
müssen wir festlegen, was ein Algorithmus ist.

Prinzipiell gibt es viele Alternativen:

- Reale Modelle: z. B. Java-Programme oder C-Programme
- Mathematische Modelle, inspiriert durch Hardware:

  z. B. Registermaschinen (RAMs)
- Rein mathematische Modelle: z. B. Turingmaschinen

Da wir einfache Beweise wollen, wählen wir ein Modell,
das so einfach wie möglich ist: **Turingmaschinen**

Einleitung

Turingmaschinen...

- wurden in den 1930er Jahren von Alan Turing entwickelt
- sind sehr einfach aufgebaut
  (Papadimitriou: „it is amazing how little we need to have everything“)
- sind kein realistisches Modell für reale Computer,
  leisten jedoch interessanterweise genau dasselbe

Church-Turing-These

Die durch Turingmaschinen berechenbaren Probleme sind genau die im intuitiven Sinn berechenbaren Probleme.

„These“, denn diese Aussage lässt sich nicht beweisen
(„im intuitiven Sinn berechenbar“ kann man nicht formal fassen)

Einleitung

Die Church-Turing-These rechtfertigt noch nicht die Verwendung von Turingmaschinen in der Komplexitätstheorie!

Erweiterte Church-Turing-These

Problem kann mit Zeitverbrauch t in natürlichem und generellen Berechnungsmodell gelöst werden gdw. es mit Zeitverbrauch $p(t)$ von Turingmaschinen gelöst werden kann, für ein Polynom p.

Zeitverbrauch wird gemessen durch Anzahl elementarer Schritte (was das ist, hängt vom Berechnungsmodell ab)

Wenn „effizient berechenbar“ = „in polynomieller Zeit lösbar“:
Problem ist entweder in allen Modellen effizient berechenbar oder in keinem.

Der Grad des Polynoms kann sich aber u. U. unterscheiden.

Kapitel 2

Turingmaschinen

Turingmaschine

- ist zu jeder Zeit in einem von endlich vielen Zuständen
- Arbeitet auf einseitig unendlichem Arbeitsband, das aus Kette von linear geordneten Zellen besteht
- Jede Zelle enthält eines von endlich vielen Symbolen
- Zu Anfang enthält das Band ganz links das Sondersymbol $\triangleright$ gefolgt von der Eingabe, gefolgt von unendl. oft Sondersymbol $\perp$
- Die Maschine hat einen Schreib-/Lesekopf, der zu Anfang auf dem ersten (linken) Symbol der Eingabe steht
- In jedem Schritt kann die Maschine das Symbol der aktuellen Zelle lesen und schreiben, sowie den Kopf um eine Position verschieben
- Arbeitet feste Verarbeitungsvorschrift ab

T2.1

Turingmaschinen

Definition Turingmaschine

(Deterministische) Turingmaschine (kurz: (D)TM) ist Tupel

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$$

mit

- Q endlicher Menge von Zuständen
- Σ endliches Eingabealphabet
- Γ endliches Bandalphabet mit $\Sigma \subseteq \Gamma$ und $\{\bot, \rhd\} \subseteq \Gamma \setminus \Sigma$
- $\delta : ((Q \setminus \{q_{\text{acc}}, q_{\text{rej}}\}) \times \Gamma) \to (Q \times \Gamma \times \{L, R, N\})$ Übergangsfunktion
- $q_0 \in Q$ Startzustand
- q_{acc} akzeptierender Endzustand
- q_{rej} verwerfender Endzustand

Turingmaschinen: Beispiel

Beispiel

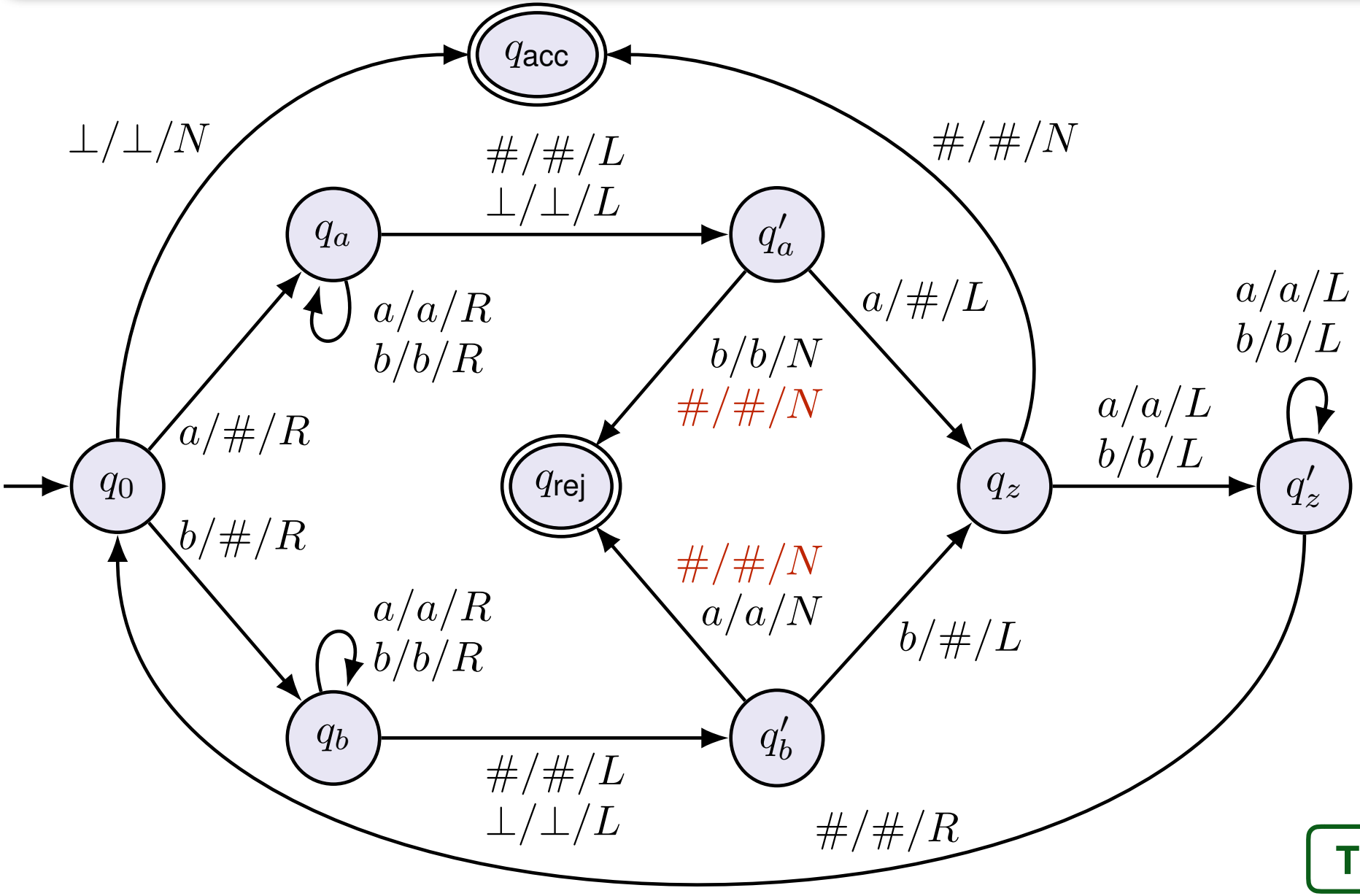
TM M, die die Eingabe $w \in \{a, b\}^*$ akzeptiert
gdw. $w = u\overleftarrow{u}$ mit $u \in \{a, b\}^*$ und $\overleftarrow{u} = u$ rückwärts gelesen

Idee

- Ersetze Symbol unter Kopf durch # und „merke“ (im Zustand), ob a oder b überschrieben wurde.
- Laufe nach rechts bis $\perp$ oder # und gehe 1 Schritt nach links.
- Vergleiche Symbol unter Kopf mit „gemerktem“ Symbol.
  Wenn unterschiedlich, dann verwirf.
- Überschreibe aktuelles Symbol mit #.
- Laufe ganz nach links.
- Laufe nach rechts bis zum ersten Nicht-#-Symbol.
  Ist dieses ein $\perp$, dann akzeptiere. Sonst beginne von vorn.

T2.2

Turingmaschinen: Beispiel



T2.3

Alle fehlenden Übergänge wie z. B. $\delta(q'_z, \perp)$ werden auf $(q_{\text{rej}}, \perp, N)$ gesetzt.

Turingmaschinen

Konvention: Wenn $\delta(q, a) = (q', a', B)$, dann verlangen wir, dass

- wenn $a = \triangleright$, dann $a' = \triangleright$ und $B = R$;
- wenn $a \neq \triangleright$, dann $a' \neq \triangleright$.

Also: $\triangleright$ bleibt stets am linken Bandende und steht sonst nirgendwo.

Definition Konfiguration

Konfiguration einer TM M hat Form uqv, wobei

- $u \in \Gamma^*$ Bandbeschriftung links des Kopfes,
- $q \in Q$ der momentane Zustand,
- $v \in \Gamma^*$ Bandbeschriftung ab (inkl.) Kopf nach rechts
- $uv \in \{\triangleright\} \cdot (\Gamma \setminus \{\triangleright\})^*$

Die *Länge* $|\alpha|$ einer Konfiguration $\alpha = uqv$ ist $|uv|$.

T2.4

Turingmaschinen

Mehr Begriffe:

- Für Eingabe $w \in \Sigma^*$ ist $\triangleright q_0 w$ die *Startkonfiguration*.
- β ist *Folgekonfiguration* von α,
  wenn β aus α in einem Schritt hervorgeht.

  Wir schreiben $\alpha \vdash_M \beta$ (formale Def. als Übung).

  Folgekonfigurationen von DTMs sind eindeutig!
- Konfiguration uqv ist *akzeptierend*, wenn $q = q_{\text{acc}}$,
  und *verwerfend*, wenn $q = q_{\text{rej}}$.
- *Berechnung* auf Eingabe w ist Folge von Konfigurationen $\alpha_0, \alpha_1, \ldots$
  (endl. oder unendl.) mit α_0 Startkonfig. für w und $\alpha_i \vdash_M \alpha_{i+1}$ für $i \geq 0$.
- Endl. Berechnung $\alpha_0, \ldots, \alpha_k$ ist *akzeptierend*, wenn α_k akzeptierend,
  und *verwerfend*, wenn α_k verwerfend.

T2.5

Turingmaschinen

Mehr Begriffe:

- DTM *akzeptiert* Eingabe w wenn die (maximale) Berechnung auf w endlich und akzeptierend ist, sonst *verwirft* M die Eingabe w
- Von TM M *erkannte* Sprache ist

$$L(M) := \{w \in \Sigma^* \mid M \text{ akzeptiert } w\}$$

Anm.: Es gibt zwei Wege, auf denen DTM Eingabe verwerfen kann:

- Berechnung endet in verwerfendem Zustand
- Berechnung ist unendlich (TM terminiert nicht)

Kapitel 2

Entscheidungsprobleme

Eingabe für TM ist (endl.) Wort aus Σ^*.

Andere Eingaben (Graphen, Zahlen, etc) müssen kodiert werden.

Definition Problem

(Entscheidungs)problem ist Teilmenge $L \subseteq \Sigma^*$, für ein endliches Alphabet Σ.

Idee: das Problem ist die Menge aller „Ja-Instanzen“, z. B.:

- GAP ist die Menge aller Tripel (G, v, v') mit $G = (V, E)$ Graph und $v, v' \in V$ (geeignet kodiert), so dass v' von v erreichbar in G
- CLIQUE ist die Menge aller Paare (G, k) so dass G eine k-Clique hat

Wir geben die Kodierung in der Regel nicht explizit an.

Entscheidungsprobleme

Wir setzen „Turingmaschine“ mit „Algorithmus“ gleich

Wir interessieren uns für Algorithmen, die auf jeder Eingabe anhalten (denn nur solche Algorithmen lösen ein Entscheidungsproblem).

Definition Entscheidungsverfahren

TM M *entscheidet* Problem L: M terminiert auf jeder Eingabe und $L(M) = L$. Dann ist M *Entscheidungsverfahren* für L.

Also: wenn M Entscheidungsverfahren für L ist, dann

- akzeptiert M jedes $w \in \Sigma$ mit $w \in L$ durch halten in q_{acc};
- verwirft M jedes $w \in \Sigma$ mit $w \notin L$ durch halten in q_{rej}.

Kapitel 2

Mehrband-TM

(Deterministische) Mehrband-TM hat mehrere Bänder (endlich viele):

- wie Einband-TM ist sie zu jedem Zeitpunkt in einem Zustand (nicht ein Zustand pro Band!)
- Es gibt einen Schreib-/Lesekopf pro Band
- in einem Schritt werden alle Bänder gleichzeitig gelesen und geschrieben
- Die Köpfe bewegen sich unabhängig voneinander (z.B.: einer nach links, ein anderer nach rechts)
- Die Eingabe ist auf dem ersten Band *(Eingabeband);* alle anderen Bänder sind initial mit $\triangleright\perp\perp\perp\cdots$ beschriftet

T2.6

Mehrband-TM

Formal, bei k Bändern:

- Übergangsfunktion hat jetzt die Form

$$((Q \setminus \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\}) \times \Gamma^k) \to (Q \times \Gamma^k \times \{L, R, N\}^k)$$

- bei $\delta(q, a_1, \ldots, a_k) = (q', a'_1, \ldots, a'_k, B_1, \ldots, B_k)$ ist
  a_i Symbol, das auf i-tem Band gelesen wird,
  a'_i Symbol, das auf i-tem Band geschrieben wird,
  B_i Bewegung des i-ten Kopfes
- Konfiguration hat Form $(q, u_1, v_1, \ldots, u_k, v_k)$, mit
  u_i Beschriftung Band i links des Kopfes,
  v_i Beschriftung i-tes Band ab (inkl.) Kopfposition
- Akzeptanz ist wie für Einband DTMs definiert.

T2.7

Kapitel 2

Komplexitätsklassen

Komplexitätsklasse ist Klasse von Problemen, die mit bestimmter „Menge“ einer bestimmten Ressource gelöst werden können.

Je nach betrachteter Ressource:

- Zeitkomplexitätsklassen
- Platzkomplexitätsklassen

„Menge“ z. B.:

- polynomiell viel
- exponentiell viel

Wesentliche Fragestellung der Komplexitätstheorie:

Wie ist der Zusammenhang zwischen verschiedenen K.-Klassen?

Zeitkomplexitätsklassen

Definition zeitbeschränkte Funktion, DTime

Für DTM M und $w \in \Sigma^*$ schreiben wir $\text{time}_M(w) = n$,
wenn M auf w nach n Schritten hält.

Sei $t : \mathbb{N} \to \mathbb{N}$ monoton wachsende Funktion mit $t(n) \geq n$.
M ist *t-zeitbeschränkt,* wenn $\text{time}_M(w) \leq t(n)$ für alle w der Länge n.

Wir definieren *Zeitkomplexitätsklasse* abhängig von Bandzahl:

$$\text{DTime}_k(t) := \{L \subseteq \Sigma^* \mid \text{es gibt } \mathcal{O}(t)\text{-zeitbeschränkte } k\text{-Band-DTM } M \text{ mit } L(M) = L\}$$

Zum Beispiel:

$\text{DTime}_1(n^2)$: Menge der Probleme, die in quadratischer Zeit von 1-Band-TM entschieden werden können.

Zeitkomplexitätsklassen

Beispiel:

Betrachte die 1-Band-TM M mit $L(M) = \{u\overleftarrow{u} \mid u \in \{a,b\}^*\}$ aus T2.2.

M ist $2n^2$-zeitbeschränkt:

- Bei Eingabe der Länge n werden n Symbole verglichen.
- Jeder Vergleich erfordert das zweimalige Überqueren der Eingabe, also $\mathcal{O}(2n)$ Schritte.

$\rightsquigarrow$ Insgesamt $\mathcal{O}(2n^2)$ Schritte

Also ist $L(M) \in \text{DTime}_1(2n^2)$ (und damit z. B. auch $\in \text{DTime}_1(2^n)$)

Relevante Funktionen

Wichtige Funktionen t zur Definition von Zeitkomplexitätsklassen z.B.:

- n (linear), n^2 (quadratisch), n^3 (kubisch) und andere Polynome
- 2^n, 2^{n^2}, 2^{n^3} und andere Exponentialfunktionen
- $n^{\log(n)}$, $2^{\log(n)^2}$ und andere quasipolynomielle Funktionen
- $2^{\sqrt{n}}$ und andere subexponentielle Funktionen

Um einen Eindruck von deren Wachstum zu bekommen:

Wolfram Alpha is your friend. :)

Mehrband-TM

Viele Bänder vs. wenige:

Bandreduktion / Satz von Hennie und Stearns (1966)

Für alle $k \geq 1$ und t gilt:

1. $\text{DTime}_k(t) \subseteq \text{DTime}_1(t^2)$
2. Wenn es $\varepsilon > 0$ gibt mit $t(n) \geq (1+\varepsilon)n$,
   dann gilt $\text{DTime}_k(t) \subseteq \text{DTime}_2(t \cdot \log(t))$.

Beweisskizze für 1.: (siehe Theor. Inf. 2)

- Repräsentiere k Bänder als $2k$ „Spuren“ auf einem Band:
  k Spuren für Bandbeschriftung, k Spuren für Kopfmarker
- Simuliere jeden Schritt der k-Band TM
  durch zweimaliges Ablaufen des Bandes.

$\rightsquigarrow$ insgesamt $\mathcal{O}(t(n)^2)$ Schritte

Mehrband-TM

Viele Bänder vs. wenige:

Bandreduktion / Satz von Hennie und Stearns (1966)

Für alle $k \geq 1$ und t gilt:

1. $\text{DTime}_k(t) \subseteq \text{DTime}_1(t^2)$
2. Wenn es $\varepsilon > 0$ gibt mit $t(n) \geq (1+\varepsilon)n$,
   dann gilt $\text{DTime}_k(t) \subseteq \text{DTime}_2(t \cdot \log(t))$.

Beweisskizze für 2.:

- Um quadratische Zeit zu vermeiden:
  Hin- und Herlaufen über das gesamte Band umgehen.
- Dazu geschickt die Spureninhalte rotieren, so dass zu jeder Zeit
  die Köpfe der Maschine „nah“ am „alten Bandende“ sind (log. Abstand).
- Beim Rotieren braucht man 2. Band als „Zwischenspeicher“.

Mehrband-TM

Daher ist folgende Definition natürlich:

Definition DTime

Für $t : \mathbb{N} \to \mathbb{N}$ monoton wachsend mit $t(n) \geq n$ definieren wir:

$$\text{DTime}(t) := \bigcup_{k \geq 1} \text{DTime}_k(t)$$

Im Folgenden

- … sind alle weiteren deterministischen Zeitkomplexitätsklassen mittels DTime definiert
- … verwenden wir also beliebige (endliche) Bandzahl je nach Bedarf

Kapitel 2

Universelle Turingmaschinen

Manchmal braucht man **universelle Turingmaschinen,**
die alle anderen DTMs simulieren können.

Sie sind sozusagen Interpreter für DTMs, implementiert als DTM.

Vorbetrachtung: jede DTM lässt sich als ein Wort kodieren.
(siehe Theor. Inf. 2)

Universelle Turingmaschinen

Theorem

Es gibt eine *universelle DTM* U, d. h. U erfüllt folgende Eigenschaft.

Für alle DTMs M und Eingaben w für M gilt:

$$U \text{ akzeptiert } \text{code}(M)w \quad \text{gdw.} \quad M \text{ akzeptiert } w$$

Beweisidee: (siehe auch Theor. Inf. 2)

- Erzeuge zunächst $\text{code}(M)\text{code}(q_0 w)$.
- Wenn M bei Eingabe w die Berechnung $q_0 w = K_1 \vdash_M K_2 \vdash_M \cdots$ ausführt, erzeuge sukzessive

  $$\text{code}(M)\text{code}(K_1),\ \text{code}(M)\text{code}(K_2),\ \text{code}(M)\text{code}(K_3),\ \ldots$$

- Jeder einzelne Schritt wird durch mehrmaliges Hin- und Herlaufen implementiert.

Übersicht Vorlesung

Kapitel 1: Einführung

Kapitel 2: Turingmaschinen

Kapitel 3: P vs. NP (Grundlagen)

Kapitel 4: Platzkomplexität (Grundlagen)

Kapitel 5: Orakel (Grundlagen)