Zusammenfassung von Theoretische Informatik I

Behandelte Themen

- Sprachklassen (Chomsky Hierarchie): regulär = erkennbar = rechtslinear, deterministisch kontextfrei, kontextfrei, kontextsensitiv, Typ 0
- Automatenmodelle zur Beschreibung von Sprachen::
  NEAs, DEAs, ε-NEAs, Wort-NEAs, Kellerautomaten, det. Kellerautomaten
- Andere Mechanismen, um Sprachen endlich zu beschreiben:
  reguläre Ausdrücke, verschiedene Arten von Grammatiken
- Eigenschaften von Sprachfamilien:
  Abschlusseigenschaften, Entscheidbarkeit und Komplexität von Problemen
- Konstruktionen und Beweistechniken:
  Potenzmengenkonstruktion, Produktautomat, Quotientenautomat, Nerode-Rechtskongruenz, verschiedene Pumping Lemmas, Normalformen von Grammatiken, etc.

Zusammenfassung Abschlusseigenschaften

	$L_1 \cap L_2$	$L_1 \cup L_2$	$\overline{L}$	$L_1 \cdot L_2$	L^*
Typ 0	✓	✓	✗	✓	✓
kontextsensitiv	✓	✓	✓	✓	✓
kontextfrei	✗	✓	✗	✓	✓
det. kontextfrei	✗	✗	✓	✗	✗
regulär	✓	✓	✓	✓	✓

Zusammenfassung Entscheidungsprobleme

	Wortproblem	Leerheitsprob..	Äquivalenzprob.
Typ 0 Grammatik	unentsch.	unentsch.	unenscht.
Typ 1 Grammatik	nicht polyzeit	unentsch.	unentsch.
Typ 2 Grammatik / PDA	polyzeit	polyzeit	unentsch.
det. PDA	linearzeit	polyzeit	entsch. [2001]
NEA / reg. Ausdruck / Typ 3 Grammatik	linearzeit	linearzeit	nicht polyzeit
DEA	linearzeit	linearzeit	polyzeit

Ausblick auf Theoretische Informatik II

Themen von Theoretische Informatik II:

- Entscheidbarkeit und Berechenbarkeit
  Welche Probleme sind algorithmisch entscheidbar und welche nicht?

  Beispiele für unentscheidbare Probleme:
  Äquivalenzproblem für PDAs, Wortproblem für Typ 0 Grammatiken

- Komplexität
  Wenn ein Problem entscheidbar / berechenbar ist, wieviel Zeit/Speicherplatz benötigt man mindestens?

  Beispiel:
  Das Äquivalenzproblem für NEAs kann man (wahrscheinlich) nicht in polynomieller Zeit entscheiden

- Sprachen und Grammatiken der Typen 0 und 1

Entscheidbarkeit / Berechenbarkeit

Wir haben die Entscheidbarkeit zahlreicher Probleme nachgewiesen, z.B.

- Wortproblem für kontextfreie Grammatiken;
- Äquivalenzproblem für DEAs.

Methode:

- Angabe eines Algorithmus in Pseudocode (z.B. CYK-Algorithmus)
- Beschreibung des Verfahrens, so dass Implementierung möglich wäre

  Berechnen von $\sim_{\mathcal{A}}$ über Folge $\sim_0, \sim_1, \ldots$ für beide Automaten

  Konstruktion der Quotientenautomaten

  Test auf Isomorphie

Genug Information für Implementierung in konkreter Programmiersprache!

Aber wie beweist man, dass für ein Problem kein Algorithmus existiert?

Dazu muss man zunächst die Frage beantworten:

Was ist ein Algorithmus?

Mögliche Antworten:

- Programmiersprachen: C, Pascal, Java, Lisp, Prolog, Assembler, etc.
- Mathematische Formalismen: Turingmaschine, Registermaschine (RAM), WHILE Programme, μ-berechenbare Funktionen, λ-Kalkül, Abstract State Machines (ASMs), etc.

Interessant: alle diese Modelle sind gleich mächtig.

Wir wählen möglichst einfaches Modell, um Beweise zu erleichtern:

Turingmaschine:

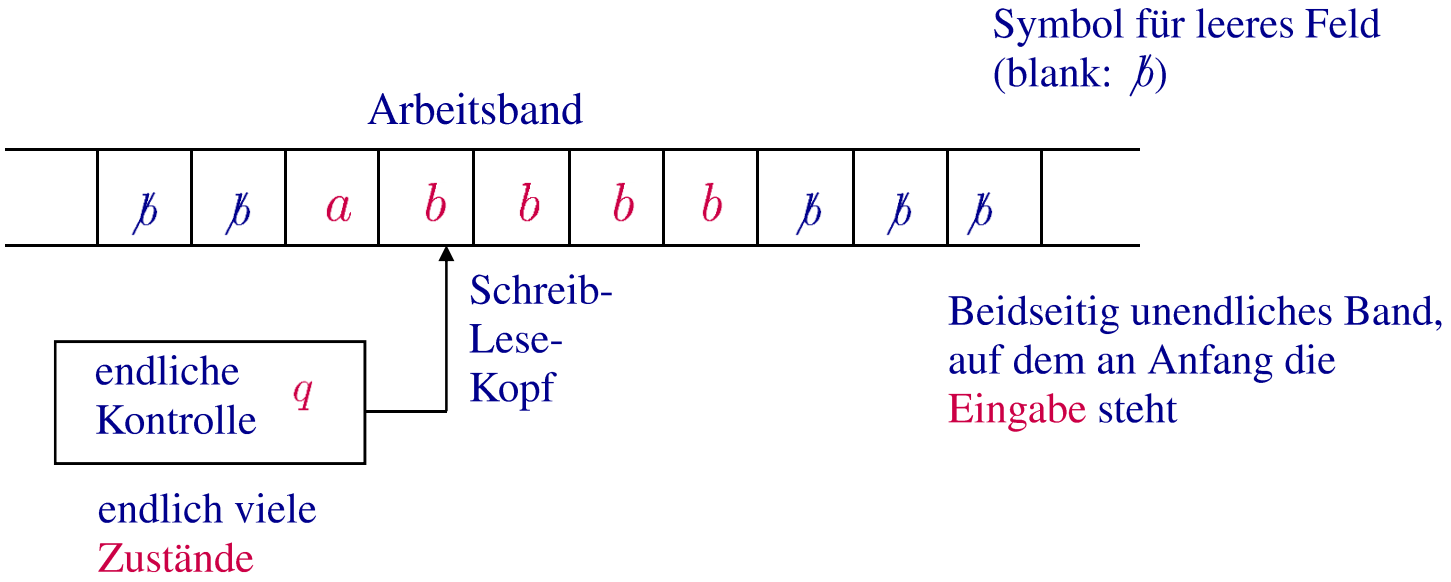
- entwickelt 1936 von Alan Turing, um Berechenbarkeit zu studieren
- endliche Kontrolle wie bei NEAs und PDAs
  + unendliches Arbeitsband (ohne Zugriffsbeschränkung von PDAs)

Church-Turing These:

> Die (intuitiv) berechenbaren Funktionen sind genau die mit Turingmaschinen (und damit auch mit Java-Programmen etc.) berechenbaren Funktionen.

These, kein Theorem, da „intuitiv berechenbar“ kein formaler Begriff ist.

Turing-Maschine



- Kopf in jedem Schritt um max. ein Feld nach links oder rechts.
- Akzeptieren/Verwerfen über Endzustände

Papadimitriou:

„It is amazing how little we need to have everything.“

Definition Turingmaschine

Eine deterministische Turingmaschine (DTM) über Eingabealphabet Σ hat die Form $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, \Delta, F)$, wobei

- Q endliche Zustandsmenge ist,
- Σ das Eingabealphabet ist,
- Γ das Arbeitsalphabet ist mit $\Sigma \subseteq \Gamma$, $\not b \in \Gamma \setminus \Sigma$,
- $q_0 \in Q$ der Anfangszustand ist,
- $F \subseteq Q$ die Endzustandsmenge ist und
- $\delta : (Q \times \Gamma) \to (Q \times \Gamma \times \{R, L, N\})$ die Übergangsfunktion ist.

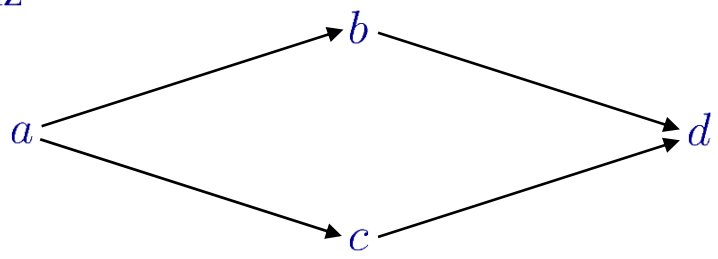
Es gibt auch wieder eine nicht-deterministische Version.

Zur formalen Definition von Entscheidbarkeit betrachten wir Entscheidungsprobleme als formale Sprachen:

- Probleminstanz wird als Wort w dargestellt.
- $w \in L$ gdw. w eine „ja-Instanz“ ist.

Beispiel: Erreichbarkeitsproblem in gerichteten Graphen

Instanz



d erreichbar von a?

dargestellt als Wort

00/01#00/10#01/11#10/11##00/11

Graph Anfrage

Turingmaschinen können formale Sprachen erkennen:

$$L(M) = \{w \in \Sigma^* \mid M \text{ gestartet auf } w \text{ hält in Endzustand}\}$$

Beachte: wenn $w \notin L(M)$, dann entweder

- M gestartet auf w hält an, aber nicht in Endzustand oder
- M gestartet auf w hält niemals an.

Ein Algorithmus, der ein Problem entscheidet, soll natürlich immer anhalten

Definition Entscheidbarkeit

Ein Problem $L \subseteq \Sigma^*$ heisst entscheidbar gdw. es eine Turingmaschine M gibt, die auf allen Eingaben anhält und mit $L(M) = L$.

Zusammenhang zur Chomsky-Hierarchie:

- Typ 0

  Eine Sprache ist vom Typ 0 (mit Grammatik erzeugbar) gdw. sie von einer Turingmaschine erkannt wird (TM muss nicht anhalten!).

  TMs können also verwendet werden, um Eigenschaften von Typ 0-Sprachen zu studieren (Abschluss etc).

- Typ 1

  Eine Sprache ist vom Typ 1 (mit kontextsensitiver Grammatik erzeugbar) gdw. sie von einem linear beschränkten Automaten erkannt wird.

  Linear beschränkter Automat:
  Turingmaschine, die nur den von der Eingabe belegten Teil des Bandes nutzen darf

Theorem

Die Sprache $L = \{a^n b^n c^n \mid n \geq 0\}$ wird von LBA erkannt.

LBA, der L erkennt, geht wie folgt vor:

- ersetze erstes a durch a', erstes b durch b' und erstes c durch c'; prüfe dabei, ob Eingabe von der Form $a^*b^*c^*$, verwerfe wenn nicht
- laufe zurück zum zweiten a, ersetze es durch a', das zweite b durch b' und das zweite c durch c'
- dies wird solange wiederholt, bis kein a oder kein b oder kein c mehr gefunden wird; bei fehlendem b oder c: verwerfe (zu viele a's)
- bei fehlendem a prüfe, ob das Band nur noch Symbole a', b', c' enthält, verwerfe wenn nicht (zu viele b's oder c's)
- Akzeptiere

Komplexitätstheorie

Klassifikation von Entscheidungsproblemen in Komplexitätsklassen gemäss Ressourcen, die zum Entscheiden des Problems benötigt werden.

Wichtige Komplexitätsklassen z.B.:

- P oder PTime
  Menge der Probleme, die mit polynomial zeitbeschränkter deterministischer Turingmaschine entschieden werden können
- NP
  Menge der Probleme, die mit polynomial zeitbeschränkter nicht-deterministischer Turingmaschine entschieden werden können
- PSpace
  Menge der Probleme, die mit polynomial platzbeschränkter Turingmaschine entschieden werden können

Besonders interessant ist der Zusammenhang von P und NP:

P

wird i.d.R. als Menge der effizient lösbaren Probleme betrachtet

also: polynomielle Laufzeit = akzeptable Laufzeit

z.B.
Wortproblem DEAs und NEAs, Sortierprobleme, Arithmetische Operationen, etc

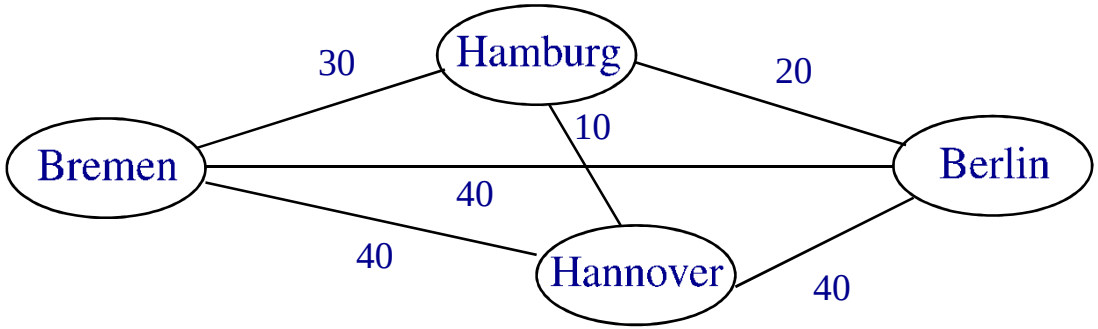
NP

für viele wichtige kombinatorische Probleme gibt es sehr einfache nicht-deterministische Polyzeit-Algorithmen, aber deterministische sind nicht bekannt

z.B. viele Logikprobleme, graphentheoretische Probleme, mengentheoretische Probleme, Routing Probleme, Scheduling Probleme, Datenbankproblem, etc pp

Beispiel: Das Travelling Salesman Problem

Eingabe: Ein Routennetz mit Kosten, z.B.:



und eine Kostengrenze x, z.B. 120

Frage: Kann man von Bremen aus eine Rundreise organisieren, die alle Städte einschließt und Gesamtkosten ≤ 120 hat?

Ist ein typisches "schwieriges" Problem in NP:

- hat stark kombinatorischen Charakter
- ist offensichtlich in NP
- es ist unbekannt, ob das Problem in P ist

Die bekannteste offene Frage der Theoretischen Informatik:

Ist P = NP (unwahrscheinlich)

oder

P $\neq$ NP (wie allgemein vermutet) ???

Mit anderen Worten:
deterministische Polyzeit-TMs = nicht-deterministische Polyzeit-TMs?

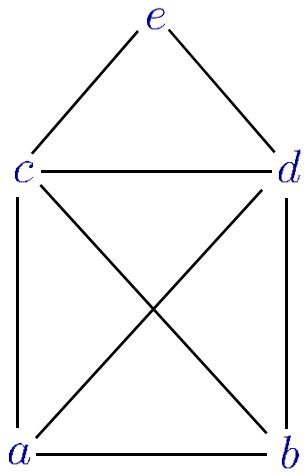
Man kann die Frage auch auf viele (sehr interessante) andere Weisen stellen!

Dies ist eine extrem prominente Frage, auch über die Informatik hinaus:

- das Clay Mathematics Institute hat es in seine Liste der 7 wichtigsten ungelösten Probleme der Mathematik aufgenommen
- "Milenniums-Problem", Preisgeld: US$1.000.000
- es gibt immer mal wieder Behauptungen, das Problem wäre gelöst, die es manchmal bis in die Medien schaffen (z.B. Deolalikar 2011)

Beispiel:

zwei verwandte Probleme auf ungerichteten Graphen



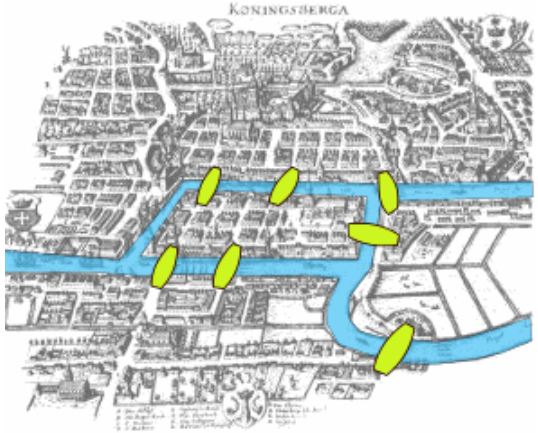
Problem 1: Eulerkreis

Gegeben ungerichteten Graph G, entscheide ob es Kreis in G gibt, der jede Kante genau einmal besucht

Euler:

Eulerkreis existiert gdw. jeder Knoten geraden Grad hat

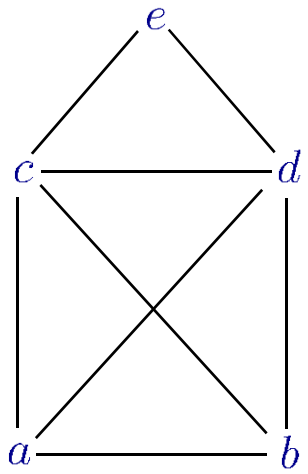
Offensichtlich deterministisch in Polyzeit prüfbar.



Historisch löste er damit das Königsberger Brückenproblem

Beispiel:

zwei verwandte Probleme auf ungerichteten Graphen



Problem 2: Hamiltonkreis

Gegeben ungerichteten Graph G, entscheide ob es

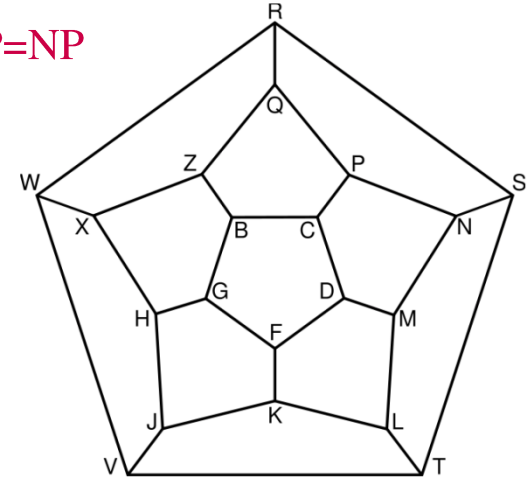
Kreis in G gibt, der jeden Knoten genau einmal besucht

Interessanter Kontrast zu Euler-Kreis:

Dieses Problem ist NP-vollständig

Es folgt insbesondere:

kein deterministischer Polyzeit-Algorithmus außer P=NP



Beachte:

Traveling Salesman ist auch

eine Art von Hamiltonkreis-Problem.