Die Struktur kontextfreier Sprachen

Was unterscheidet kontextfreie Sprachen von regulären Sprachen?

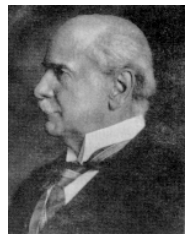
Schon gesehen:
kontextfreie Sprachen können unbeschränkt zählen, reguläre nicht.

"Typische" kontextfreie Sprachen, die nicht regulär sind:

- $\{a^n b^n \mid n \geq 0\}$
- Klammersprachen (auch gennant Dyck-Sprachen) der Form

$$S \to \underset{1}{(}S\underset{1}{)} \qquad \cdots \qquad S \to \underset{n}{(}S\underset{n}{)}$$

$$S \to SS \qquad\qquad S \to \varepsilon$$



Gibt es noch "andere Arten" von echt kontextfreien Sprachen?

Wir wollen zeigen:
Jede kontextfreie Sprache kann dargestellt werden als Schnitt einer Dyck-Sprache und einer regulären Sprache, plus "ein wenig Umbenennung".

Zum Beispiel:

$$\{a^n b^n \mid n \geq 0\} \quad = \quad h(D_1 \cap R) \quad \text{wobei}$$

- D_1 definiert über $S \to SS,\ S \to (S),\ S \to \varepsilon$
- R ist $(^*)^*$
- h benennt "$($" um in a und "$)$" in b

Um "Umbenennung" allgemeiner zu fassen, brauchen wir den Begriff eines Homomorphismus

Definition 10.17 (Homomorphismus)

Seien Σ und Γ Alphabete. Ein Homomorphismus von Σ^* nach Γ^* ist eine Abbildung $h : \Sigma^* \to \Gamma^*$ so dass $h(wv) = h(w)h(v)$ für alle $w, v \in \Sigma^*$.

Aus dieser Definition folgt unmittelbar:

1. $h(\varepsilon) = \varepsilon$
2. $h(a_1 \cdots a_n) = h(a_1) \cdots h(a_n)$, also kann man einen Homomorphismus $h : \Sigma^* \to \Gamma^*$ durch Angeben von $h(a)$ für alle $a \in \Sigma$ definieren

Homomorphismen liefern eine weitere Abschlusseigenschaft (ohne Beweis).

Theorem 10.18

Wenn L eine reguläre (bzw. kontextfreie) Sprache ist und h ein Homomorphismus, dann ist auch das homomorphe Bild $h(L) := \{h(w) \mid w \in L\}$ regulär (bzw. kontextfrei).

Theorem 10.18 (Chomsky-Schützenberger)

Jede kontextfreie Sprache L ist das homomorphe Bild des Schnittes einer Dyck-Sprache D und einer regulären Sprache R, es gibt also Homomorphismus h so dass

$$L = h(D \cap R).$$

Beweis: Sei L kontextfrei

Es gibt Grammatik $G = (N, \Sigma, P, S)$ für L in Chomsky Normalform.

Für jede Produktion $\pi \in P$, definiere

$$\pi' = \begin{cases} A \to \overset{1}{\underset{\pi}{(}} B \overset{12}{\underset{\pi\pi}{)(}} C \overset{2}{\underset{\pi}{)}} & \text{wenn } \pi = A \to BC \\ A \to \overset{1122}{\underset{\pi\pi\pi\pi}{()()}} & \text{wenn } \pi = A \to a \end{cases}$$

Setze $G' := (N, \Gamma, P', S)$ mit

$$P' = \{\pi' \mid \pi \in P\} \qquad \Gamma = \{\overset{1}{\underset{\pi}{(}}, \overset{1}{\underset{\pi}{)}}, \overset{2}{\underset{\pi}{(}}, \overset{2}{\underset{\pi}{)}} \mid \pi \in P\}.$$

Sei D_Γ die Dyck-Sprache mit den Klammern aus Γ.

Man sieht leicht: $L(G') \subseteq D_\Gamma$, aber die Umkehrung gilt nicht:

Z.B.: $\overset{1111}{\underset{\pi\pi\pi\pi}{()()}}$, $\overset{2211}{\underset{\pi\pi\pi\pi}{()()}} \notin L(G')$

$$\pi' = \begin{cases} A \to \overset{1}{\underset{\pi}{(}} B \overset{12}{\underset{\pi\pi}{)(}} C \overset{2}{\underset{\pi}{)}} & \text{wenn } \pi = A \to BC \\ A \to \overset{1122}{\underset{\pi\pi\pi\pi}{()()}} & \text{wenn } \pi = A \to a \end{cases}$$

Zusätzliche Eigenschaften, die alle Wörter in $L(G')$ erfüllen:

1. Auf jedes $\overset{1}{\underset{\pi}{)}}$ folgt $\overset{2}{\underset{\pi}{(}}$
2. Auf $\overset{2}{\underset{\pi}{)}}$ folgt nie eine öffnene Klammer.
3. Wenn $\pi = A \to BC$, dann
   - folgt auf $\overset{1}{\underset{\pi}{(}}$ immer $\overset{1}{\underset{\rho}{(}}$ mit $\rho = B \to \cdots$
   - folgt auf $\overset{2}{\underset{\pi}{(}}$ immer $\overset{1}{\underset{\sigma}{(}}$ mit $\rho = C \to \cdots$
4. Wenn $\pi = A \to a$, dann folgt auf $\overset{1}{\underset{\pi}{(}}$ immer $\overset{1}{\underset{\pi}{)}}$ und auf $\overset{2}{\underset{\pi}{(}}$ immer $\overset{2}{\underset{\pi}{)}}$

Beweis einfach per Induktion über die Anzahl Regelanwendungen.

$$\pi' = \begin{cases} A \to \overset{1}{\underset{\pi}{(}} B \overset{12}{\underset{\pi\pi}{)(}} C \overset{2}{\underset{\pi}{)}} & \text{wenn } \pi = A \to BC \\ A \to \overset{1122}{\underset{\pi\pi\pi\pi}{()()}} & \text{wenn } \pi = A \to a \end{cases}$$

Weiterhin gilt:

Für alle Wörter w mit $A \to^*_{G'} w$ gilt:

5_A. w beginnt mit $\overset{1}{\underset{\pi}{(}}$ wobei $\pi = A \to \cdots$

Diese Eigenschaften können durch reguläre Ausdrücke beschrieben werden!

Also ist für jedes $A \in N$ die folgende Sprache regulär:

$$R_A := \{w \in \Gamma^* \mid w \text{ erfüllt Eigenschaften 1-4 sowie } 5_A\}$$

Wir müssen also quasi nur noch zeigen:
L unterscheidet sich von D_Γ nur in den o.g. Eigenschaften

Lemma 10.19

Für alle $A \in N$ gilt:
$A \to_{G'} w$ gdw $w \in D_\Gamma \cap R_A$

"$\Rightarrow$" Induktion über Anzahl Regelanwendungen.

"$\Leftarrow$" Per Induktion über die Länge von w.

Sei $w \in D_\Gamma \cap R_A$.

Weil w wohlgeklammert und Eigenschaften 1 bis 5_A erfüllt, zeigt man leicht:

$$w \text{ hat die Gestalt } \overset{1}{\underset{\pi}{(}} u \overset{12}{\underset{\pi\pi}{)(}} v \overset{2}{\underset{\pi}{)}} \text{ mit } u, v \in \Gamma^* \text{ und } \pi = A \to \cdots$$

1. Fall: $\pi = A \to BC$

Dann $u \in D_\Gamma \cap R_B$ und $v \in D_\Gamma \cap R_C$ (5_B für u folgt aus 3 für w)

IV liefert $B \to^*_{G'} u$ und $C \to^*_{G'} v$

Also $A \to_{G'} \overset{1}{\underset{\pi}{(}} B \overset{12}{\underset{\pi\pi}{)(}} C \overset{2}{\underset{\pi}{)}} \to^*_{G'} \overset{1}{\underset{\pi}{(}} u \overset{12}{\underset{\pi\pi}{)(}} v \overset{2}{\underset{\pi}{)}} = w$

Lemma 10.19

Für alle $A \in N$ gilt:
$A \to_{G'} w$ gdw $w \in D_\Gamma \cap R_A$

"$\Rightarrow$" Induktion über Anzahl Regelanwendungen.

"$\Leftarrow$" Per Induktion über die Länge von w.

Sei $w \in D_\Gamma \cap R_A$.

Weil w wohlgeklammert und Eigenschaften 1 bis 5_A erfüllt, zeigt man leicht:

$$w \text{ hat die Gestalt } \underset{\pi}{(}u\underset{\pi\pi}{)(}v\underset{\pi}{)} \text{ mit } u, v \in \Gamma^* \text{ und } \pi = A \to \cdots$$

2. Fall: $\pi = A \to a$

Eigenschaft 4 liefert $u = v = \varepsilon$

Nach Konstruktion von G' gilt $A \to_{G'} \underset{\pi\pi\pi\pi}{1122}() = w$

Lemma 10.19

Für alle $A \in N$ gilt:
$A \to_{G'} w$ gdw $w \in D_\Gamma \cap R_A$

Aus Lemma 10.19 folgt insbesondere

$$L(G') = D_\Gamma \cap R_S$$

Definiere Homomorphismus h:

- $h(\overset{1}{\underset{\pi}{(}}) = h(\overset{1}{\underset{\pi}{)}}) = h(\overset{2}{\underset{\pi}{(}}) = h(\overset{2}{\underset{\pi}{)}}) = \varepsilon$ wenn $\pi = A \to BC$
- $h(\overset{1}{\underset{\pi}{)}}) = h(\overset{2}{\underset{\pi}{(}}) = h(\overset{2}{\underset{\pi}{)}}) = \varepsilon$ und $h(\overset{1}{\underset{\pi}{(}}) = a$ wenn $\pi = A \to a$

Für jedes $\pi \in P$: h angewendet auf π' ergibt π.

Also $L(G) = h(L(G')) = h(D_\Gamma \cap R_S)$

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

In Theo. Inf. II betrachten wir zwei fundamentale Themen der Informatik:

- Entscheidbarkeit / 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 ist, wieviel Zeit und Speicherplatz benötigt man mindestens?

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

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 ein möglichst einfaches Modell.

Turingmaschine:

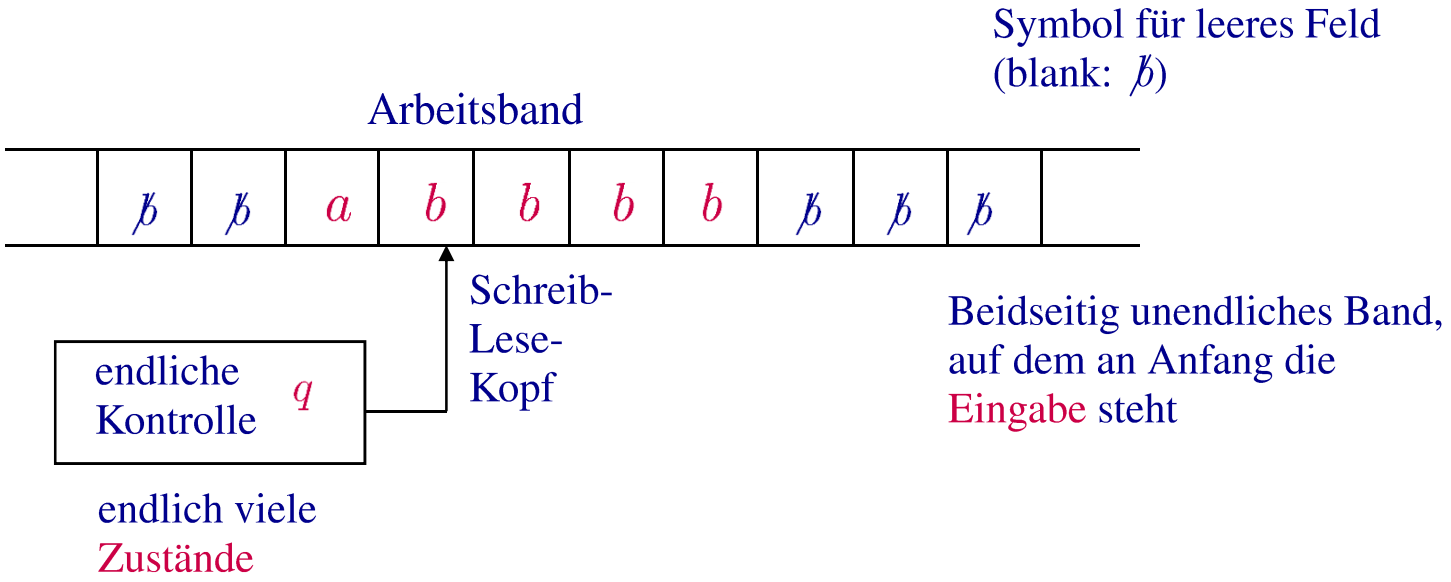
- entwickelt 1936 von Alan Turing, um Berechenbarkeit zu studieren
- endliche Kontrolle wie bei NEAs und PDAs
  + unendliches Arbeitsband (ohne Zugriffsbeschränkung von PDAs)

Interessant:

Die mit Turingmaschinen entscheidbaren Probleme sind genau die mit Java-Progammen, Lisp-Programmen, usw. entscheidbaren Probleme.

Es ist kein realistisches und mächtigeres Berechnungsmodell bekannt und wahrscheinlich gibt es auch keins. (Church-Turing These)

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.“

Unentscheidbarkeitsbeweise

Das erste Problem, von dem man üblicherweise Unentscheidbarkeit beweist, ist das Halteproblem für Turingmaschinen:

> gegeben TM M und Eingabe w für M,
> stoppt M gestartet auf w nach endlich vielen Schritten?

Unentscheidbarkeit 1937 von Alan Turing mittels Diagonalisierungsargument.

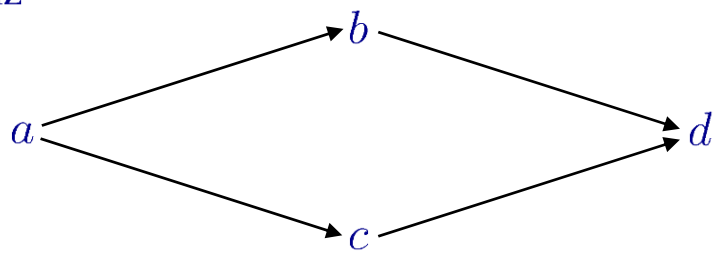
Weitere Probleme dann mittels Reduktion als unentscheidbar nachweisbar:

> wäre Problem X entscheidbar, dann auch das Halteproblem für TMs
> (oder ein anderes bereits als unentscheidbar bekanntes Problem)

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

Beispiel: Erreichbarkeitsproblem in gerichteten Graphen

Instanz



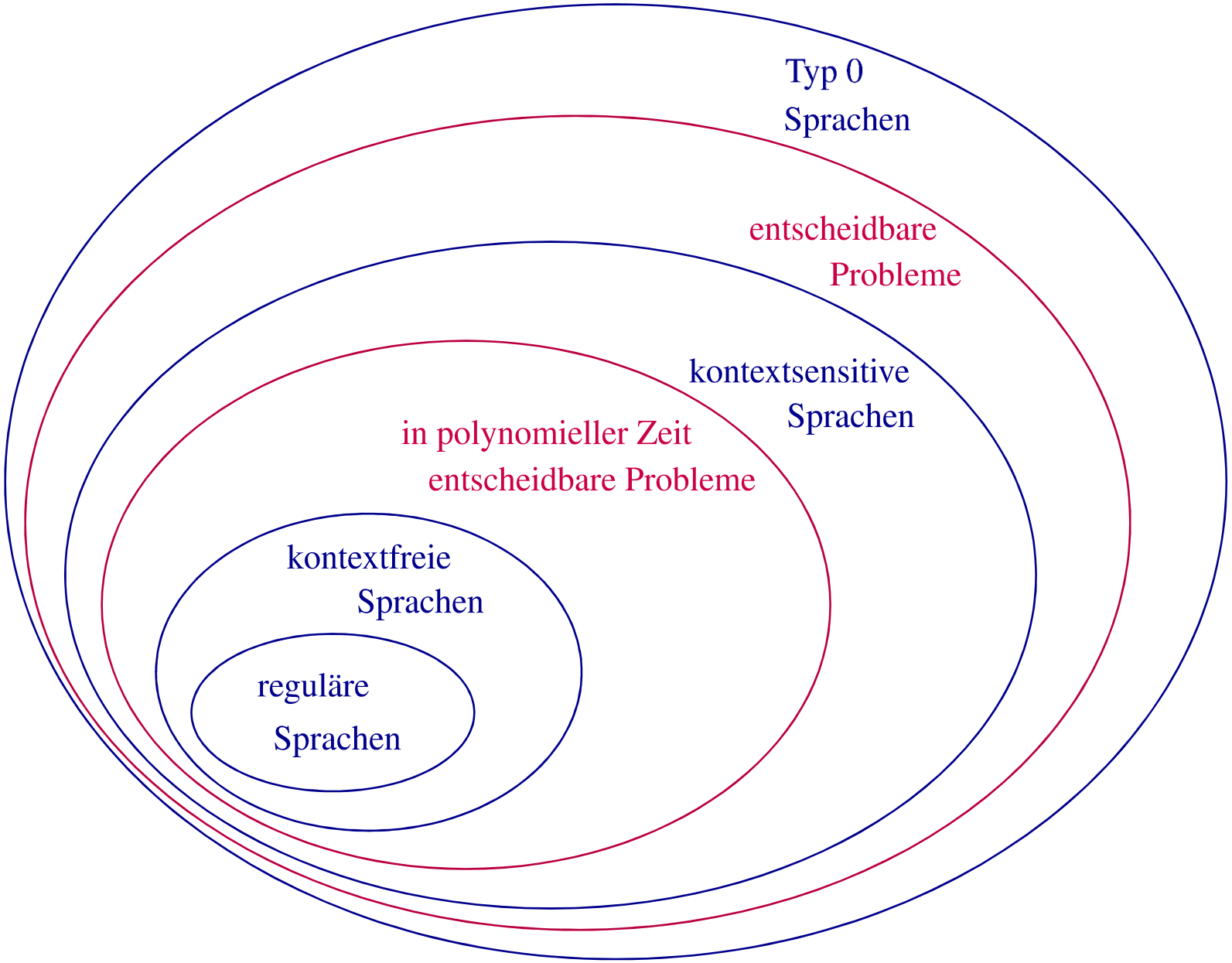
d erreichbar von a?

dargestellt als Wort

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

Graph Anfrage

Auch die Betrachtung von Entscheidbarkeit und Komplexität ist also
im Grunde nichts weiter als das Studium formaler Sprachen.


Typ 0
Sprachen
entscheidbare
Probleme
kontextsensitive
Sprachen
in polynomieller Zeit
entscheidbare Probleme
kontextfreie
Sprachen
reguläre
Sprachen

Turingmaschinen liefern auch Automatenmodelle für Typ 0 und Typ 1:

- Typ 0
  Eine Sprache ist vom Typ 0 (mit Grammatik erzeugbar) gdw. sie von einer Turingmaschine erkannt wird.

  Dabei muss die TM nicht unbedingt auf jeder Eingabe anhalten
  Das ist der Unterschied zur Entscheidbarkeit.

- 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

Komplexitätstheorie

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

Wichtige Komplexitätsklassen z.B.:

- P
  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

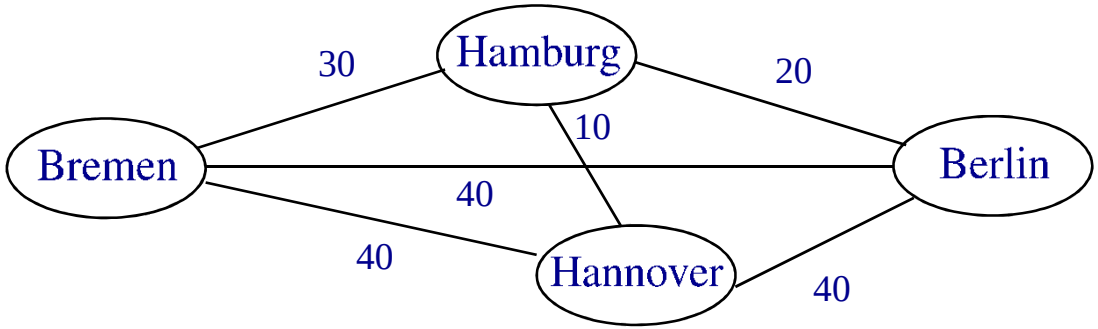
NP

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?

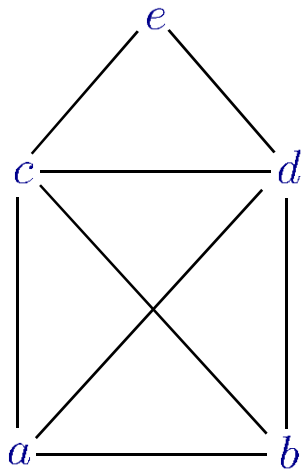
Man kann die Frage auch auf viele (sehr interessante) andere Weisen stellen!

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
- "Millenniums-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



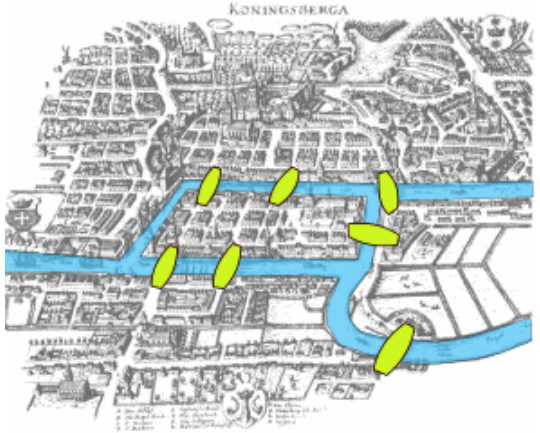
Problem 1: Eulerkreis

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

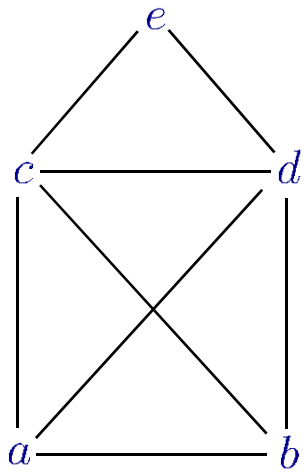
Offensichtlich deterministisch in Polyzeit prüfbar.



Historisch löste er damit das Königsberger Brückenproblem

Beispiel:

zwei verwandte Probleme auf ungerichteten Graphen



Problem 2: Hamiltonkreis

Gegeben ungerichteten Graph G, entscheide ob es

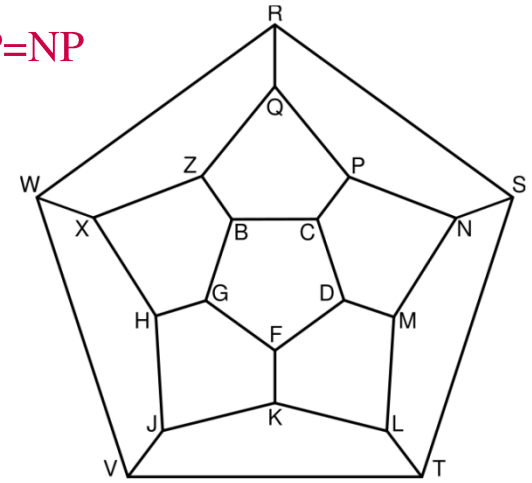
Kreis in G gibt, der jeden Knoten genau einmal besucht

Interessanter Kontrast zu Euler-Kreis:

Dieses Problem ist NP-vollständig

Es folgt insbesondere:

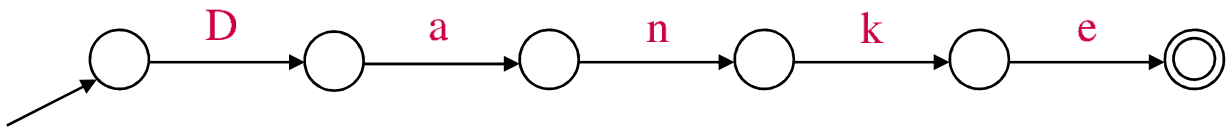
kein deterministischer Polyzeit-Algorithmus außer P=NP



Beachte:

Traveling Salesman ist auch

eine Art von Hamiltonkreis-Problem.



für Eure Aumerksamkeit!