9. CoMa Übung

5.6.13

Einführung Komplexitätstheorie & Turingmaschinen

Fragestellung: Welche algorith. Probleme von Rechnern effizient lösbar?

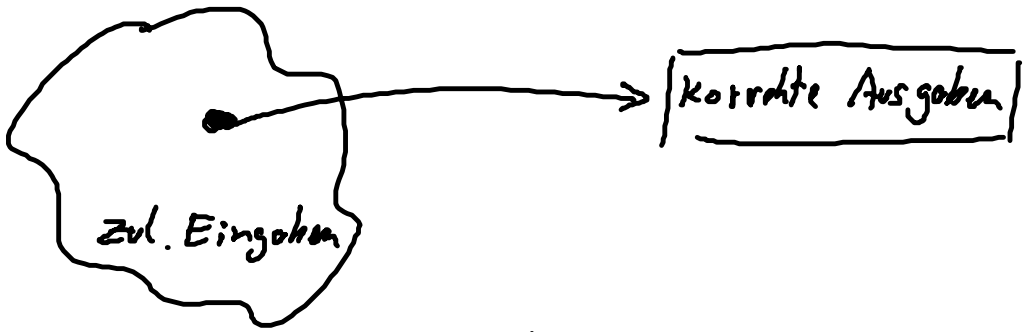
- keine Zeit auf unnötige Aufgaben verschwenden
- bestmögliche Algorithmen beweisen
- schwere Aspekte an Problemen identifizieren
- Probleme in gut lösbarer Form formulieren

zu klären:
1. was sind algorithm. Probleme?
2. Was ist ein Rechner?
3. Was bedeutet effizient?

1. Algorithmische Probleme

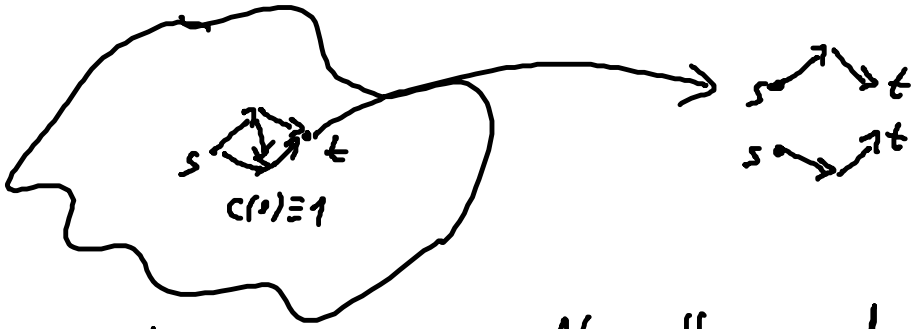
- Beschreibung der Menge der zulässigen Eingaben
- Beschr. der Funktion, die einer zulässigen Eingabe die Menge der zulässigen Ausgaben zuweist

Ein / Ausgabe jeweils endliche Folge von Zeichen eines endl. Alph.



Beispiel: kürzeste Wege Problem:

- zulässige Eingabe: Menge aller Graphen $G=(V,E)$ mit $c(e) \in \mathbb{Z}$, $s \in V$, $t \in V$
- korrekte Ausgabe: alle kürzesten s-t wege



3. Effizient: es ex. ein Algorithmus dessen Laufzeit polynomial in der Eingabegröße ist

warum polynomiell: - Hardware komplexität verdoppelt sich etwa alle 18 Monate (Moores Law)

- bei polyn. Algorithmen: es ex. $c > 1$ max. lösbare Instanz um Faktor c wächst
- bei expon. Algorithmen: max. lösbare Instanz wächst um additive Konstante

2. Rechner: benötigen Aussage über alle Rechner

Rechnermodell: - Komplex genug um alle anderen Rechner simulieren zu können
- einfach genug um Beweise zu ermöglichen

⇝ existiert so ein Modell

Churchsche Hypothese: Alle Rechnermodelle können sich gegenseitig simulieren. Menge der lösbaren Probleme nicht abhängig vom Rechnermodell.

- nicht wirklich beweisbar, wird als wahr angenommen
- simulation muss effizient sein
- ausreichend für Berechenbarkeitstheorie

erweiterte Hypothese: gegenseitige Simulation mit polyn. Mehraufwand

Einschub: Kosten für Rechenoperationen:

- Uniforme Kosten: Operationen kosten gleich viel, jeweils 1
  problem: Addition großer Zahlen gleich teuer wie die von kleinen
- logarithm Kosten: Rechenop./Vergleiche haben kosten von $\log(n)$ bei Zahl n (entspricht Kodierungslänge)
  → fairer bei Zahlen versch. länge

Rechnermodell: Turingmaschine

besteht:
- endlosem Band (enthält Zeichen aus Alph. Γ)
- Zustand aus Menge Q
- lese/schreibkopf (liest/schreibt Band, bewegt sich, ändert Zustand der Maschine)

Alphabete & Sprachen

Σ ist Alphabet $\hat{=}$ Menge von versch. Zeichen, Bsp. $\Sigma = \{0,1\}$

Kleensche Hülle Σ^* (über Alphabet Σ):
- enthält leere wort ε
- für $w \in \Sigma^*$ und $a \in \Sigma$ ist $wa \in \Sigma^*$
  wa heißt Konkatenation

positive Hülle $\Sigma^+ := \Sigma^* \setminus \{\varepsilon\}$

Sprache über Alphabet Σ: Teilmenge $L \subseteq \Sigma^*$

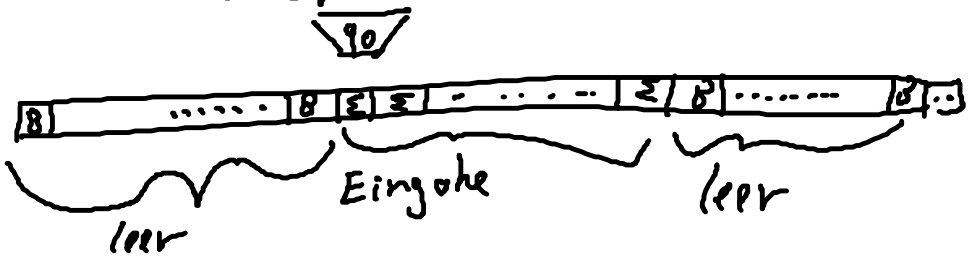
- kann leer, unendlich oder endlich sein
- Elemente $w \in L$ heissen worte
- Mengenoperationen auf sprachen möglich:
  - Konkatenation: $L_1 L_2 := \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}$
  - Potenzen: $L^i := L^{i-1} L$, $L^0 := \{\varepsilon\}$
  - Kleensche Hülle: $L^* := \bigcup_{i=0}^{\infty} L^i$

Deterministische Turingmaschine:

formal: $(Q, \Sigma, \Gamma, B, q_0, \delta, F)$

- Q: endl. Zustandsmenge
- Σ: Alphabet
- Γ: Bandalphabet, $\Gamma = \Sigma \cup \{B\}$
- B: Leerzeichen
- q_0: Startzustand, $q_0 \in Q$
- $\delta: Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, N, R\}$ (links, neutral, rechts)
- $F \subseteq Q$: Menge der akzept. Zustände

lese Zeichen + akt. Zustand → schreibe Zeichen, ändere Zustand, bewege Band

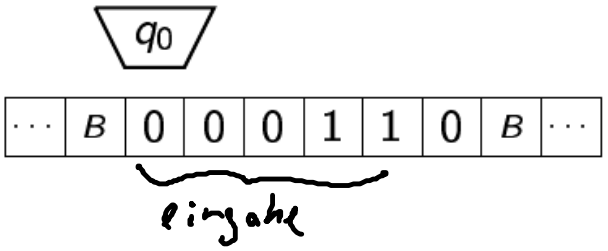


Verhalten:
- Zustand q_0 + gelesenes Zeichen c per δ abbilden
- Zustand ändern, Zeichen schreiben, Kopf bewegen
- Stopp: bei $\delta(q, c) = \delta(q, c, N)$ keine Änderung
- nach Stopp: · Rechenergebnis ablesen (auf Band)
  · Entscheidungsproblem laut q ablesen
  $q \in F \Rightarrow$ Antwort ist Ja

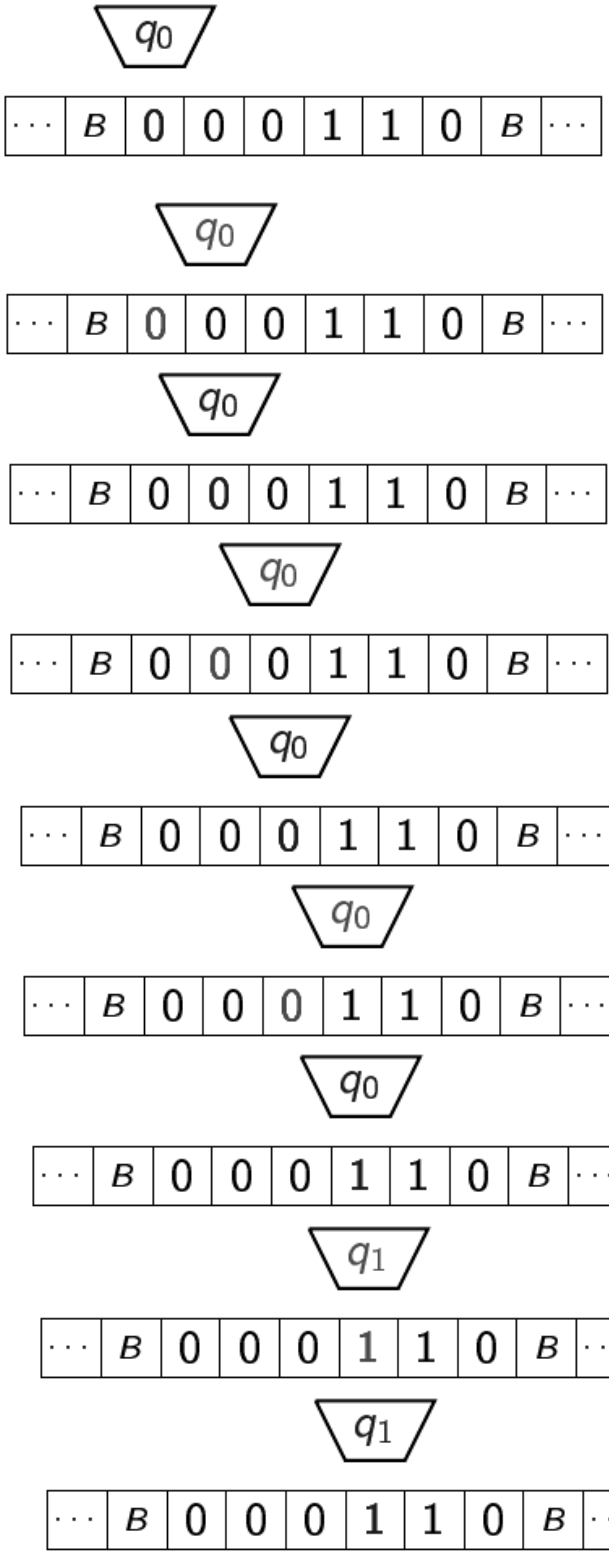
~ Entscheidungsproblem: ist Wort $w \in L$, L ist Sprache

Beispiel: Übergänge:

δ	0	1	B
q_0	$(q_0, 0, R)$	$(q_1, 1, N)$	(q_0, B, N)
q_1	$(q_1, 0, N)$	$(q_1, 1, N)$	(q_1, B, N)



● = Zustand + Zeichen lesen
○ = Zustand + Zeichen schreiben



$\delta(q_1, 1) = (q_1, 1, N)$
=> Abbruch

TM endet in q_0 => Eingabe enthält keine 1
q_1 => Eingabe enthält 1

Entwurf einer TM:

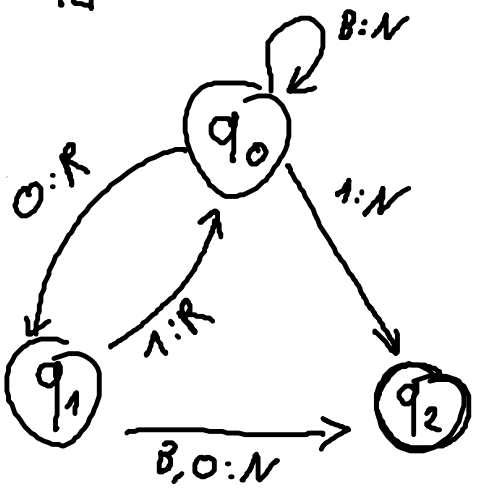
Ziel: erkenne ob Eingabewort aus Sprache $\{(01)^n \mid n \geq 0\} =: L$

bsp: 01, 0101, ε, 010101 } ok
010, 101, 1 } nicht ok

Idee: weiterer Zustand für zuletzt gelesenes Zeichen:

δ	0	1	B
q_0	$(q_1, 0, R)$	$(q_2, 1, N)$	(q_0, B, N)
q_1	$(q_2, 0, N)$	$(q_0, 1, R)$	(q_2, B, N)
q_2	—	—	—

q_0 = Start, bzw. als letztes 1 gelesen (erwarte 0)
q_1 = letztes Zeichen war 0 (erwarte 1)
q_2 = ungültiger Zustand



q_2: $w \notin L$
q_0: $w \in L$

Ablauf: Eingabe: 010

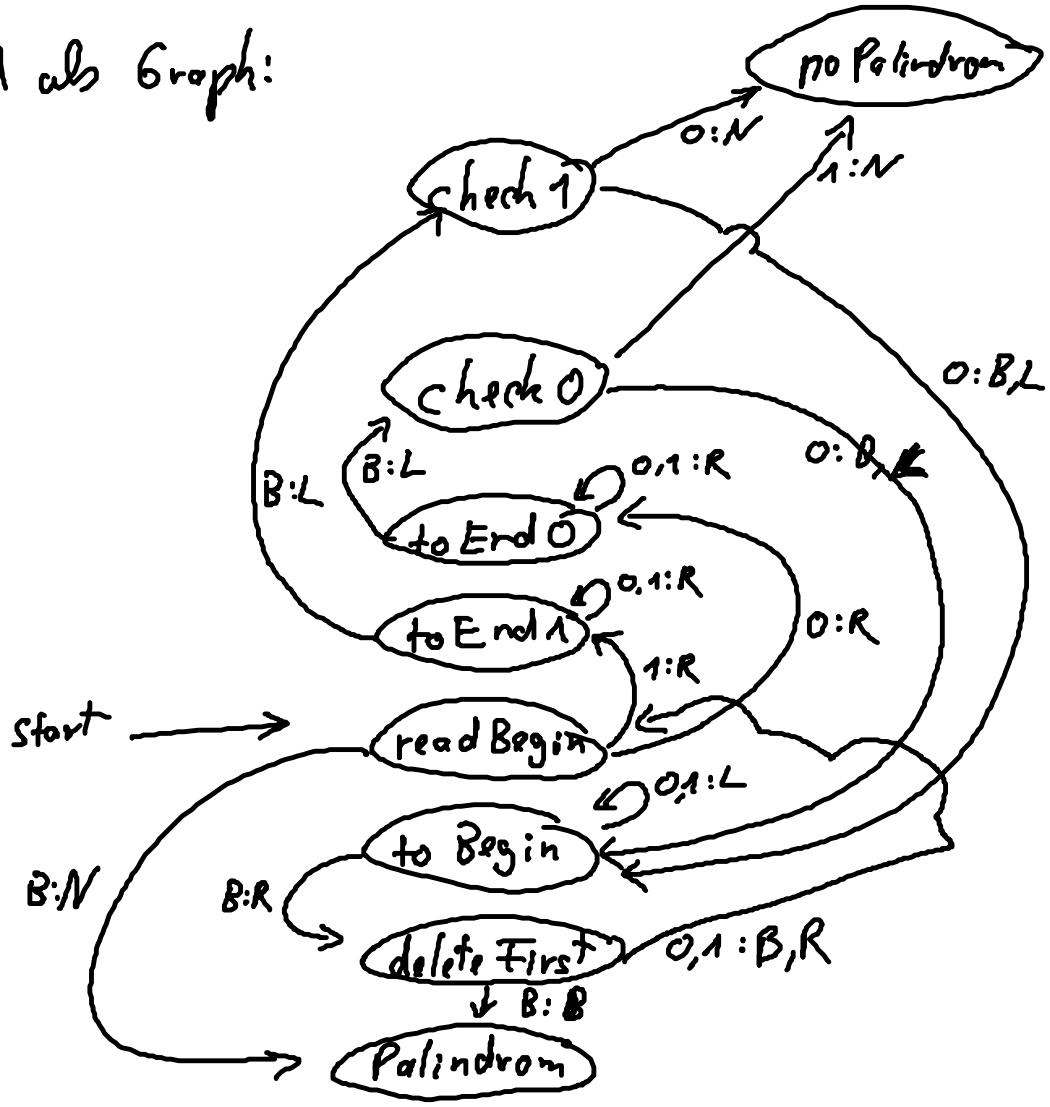
$q_0 010$ $(q_0, 0) \to (q_1, 0, R)$
$\vdash 0 q_1 10$
$\vdash 01 q_0 0$
$\vdash 010 q_1$ $(q_1, B) \to (q_2, B, N)$
$\vdash 010 q_2$ $\Rightarrow$ Abbruch, 010 ist nicht $\in L$

Entwurf einer TM zum Erkennen von Palindromen:

Idee: - Merke Zeichen am Anfang
- Vergleiche mit Ende
- Zeichen gleich? → lösche beide und gehe zu Anfang → rekursiver Start

ungleich: Abbruch

TM als Graph:



q_0 = read Begin

F = {Palindrom}