Ludwig-Maximilians-Universität München

WS 2015/2016

Martin Hofmann, Ulrich Schöpp

Komplexitätstheorie

Vorlesungsmitschrieb von

Philipp Moers
<p.moers@campus.lmu.de>
<soziflip@gmail.com>

Last updated: 19. Oktober 2015, 10:05

Zusammenfassung

Die Komplexitätstheorie beschäftigt sich mit der Klassifikation von Algorithmen und Berechnungsproblemen nach ihrem Ressourcenverbrauch, z.B. Rechenzeit oder benötigtem Speicherplatz. Probleme mit gleichartigem Ressourcenverbrauch werden zu Komplexitätsklassen zusammengefasst. Die bekanntesten Komplexitätsklassen sind sicherlich P und NP, die die in polynomieller Zeit deterministisch bzw. nicht-deterministisch lösbaren Probleme umfassen.

P und NP sind jedoch nur zwei Beispiele von Komplexitätsklassen. Andere Klassen ergeben sich etwa bei der Untersuchung der effizienten Parallelisierbarkeit von Problemen, der Lösbarkeit durch zufallsgesteuerte oder interaktive Algorithmen, der approximativen Lösung von Problemen, um nur einige Beispiele zu nennen.

Anmerkung

Dies ist ein inoffizieller Vorlesungsmitschrieb. Als solcher erhebt er keinen Anspruch auf (NP-) Vollständigkeit oder Korrektheit. Nutzung, Anmerkungen und Korrekturen sind jedoch durchaus erwünscht!

Vorlesungs-Website: `http://www.tcs.ifi.lmu.de/lehre/ws-2015-16/kompl`

Inhaltsverzeichnis

1. Einführung

1.1. Motivation

Theoretische Informatik, Berechenbarkeit und insbesondere Komplexitätstheorie ist der Informatiker-Shit schlechthin. Let's do it!

1.2. Literatur

Die Vorlesung basiert hauptsächlich auf folgendem Buch:

- Bovet, Crescenzi. Introduction to the Theory of Complexity. Prentice Hall. New York. 1994.

Weiterhin ist folgende Literatur gegeben:

- C. Papadimitriou. Computational Complexity. Addison-Wesley. Reading. 1995.
- I. Wegener. Komplexitätstheorie: Grenzen der Effizienz von Algorithmen. Springer. 2003.
- S. Arora und B. Barak. Complexity Theory: A Modern Approach.

Zur Motivation:

- Heribert Vollmer. Was leistet die Komplexitätstheorie für die Praxis? Informatik Spektrum 22 Heft 5, 1999.

- Stephen Cook: The Importance of the P versus NP Question. Journal of the ACM (Vol. 50 No. 1)

2. Turingmaschinen, Berechenbarkeit und Komplexität

Vorlesung vom 12.10.15

2.1. Turingmaschinen

Definition

Eine **Turingmaschine** T mit k Bändern ist ein 5-Tupel

$$T = (Q, \Sigma, I, q_0, F)$$

- Q ist eine endliche Menge von Zuständen
- Σ ist eine endliche Menge von Bandsymbolen, $\square \in \Sigma$
- I ist eine Menge von Quintupeln der Form (q, s, s', m, q') mit $q, q' \in Q$ und $s, s' \in \Sigma^k$ und $m \in \{L, R, S\}^k$

- $q_0 \in Q$ Startzustand
- $F \subseteq Q$ Endzustände

$\square$ ist das Leerzeichen oder **Blanksymbol** .

T heißt **deterministisch** genau dann, wenn für jedes $q \in Q$ und $s \in \Sigma^k$ genau ein Quintupel der Form $(q, s, \_, \_, \_) \in I$ existiert. Sonst heißt T **nichtdeterministisch** .

Eine Turingmaschine heißt **Akzeptormaschine** genau dann, wenn zwei Zustände $q_A, q_R \in F$ speziell markiert sind. q_A signalisiert Akzeptanz, q_R signalisiert Verwerfen der Eingabe.

Eine Turingmaschine heißt **Transducermaschine** genau dann, wenn ein zusätzliches Band ausgezeichnet ist (das Ausgabeband).

Beispiel

Akzeptormaschine T für Sprache $L = \{0^n 1^n | n \geq 0\}$ wobei $\Sigma = \{0, 1\}, Q = \{q_0, \dots q_4\}$

T wird deterministisch sein. $T = (Q, \Sigma, I, q_0, F), q_A = q_1, q_R = q_2, F = \{q_1, q_2\}, k = 2$

q	s_1	s_2	s_1'	s_2'	m_1	m_2	q'
q_0	□	□	□	□	S	S	q_1
q_0	0	□	0	0	R	R	q_3
q_0	1	□	1	□	S	S	q_2
q_3	□	□	_	_	_	_	q_2
q_3	0	□	0	0	R	R	q_3
q_3	1	□	1	□	S	L	q_4
q_4	0	0	_	_	_	_	q_2
q_4	1	0	1	0	R	L	q_4
q_4	0	□	□	□	S	S	q_1
_	_	_	_	_	_	_	q_2

Die **globale Konfiguration** (oder der **Zustand**) einer Turingmaschine beinhaltet die Beschriftung aller Bänder, den internen Zustand ($\in Q$) und die Positionen aller k Lese-/Schreibköpfe. Globale Konfigurationen können als endliche Wörter über einem geeigneten Alphabet (z.B. $\{0,1\}$) codiert werden.

Eine Turingmaschine **akzeptiert** eine Eingabe genau dann, wenn eine Berechnungsfolge ausgehend von dieser Eingabe existiert und in einem Zustand aus F endet.

Eine Turingmaschine **akzeptiert** eine Sprache $L \subseteq (\Sigma \setminus \{\square\})^*$ falls gilt:

$$\text{Die Turingmaschine akzeptiert } w \Leftrightarrow w \in L$$

Eine Turingmaschine **entscheidet** eine Sprache $L \subseteq (\Sigma \setminus \{\square\})^*$ genau dann, wenn sie sie akzeptiert und eine/die Berechnung in q_A endet.