Theorie der Informatik

IV.2. Nichtdeterminismus

Malte Helmert Christian Tschudin

Universität Basel

6. Mai 2013

Überblick: Vorlesung

Vorlesungsteile

I. ~~Logik~~
II. ~~Automatentheorie und formale Sprachen~~
III. ~~Berechenbarkeitstheorie~~
IV. Komplexitätstheorie ⟵ Sie sind hier!

Überblick Komplexitätstheorie

Überblick über diesen Vorlesungsteil:

IV. Komplexitätstheorie
IV.1. ~~Motivation und Einführung~~
IV.2. Nichtdeterminismus ⟵ hier!
IV.3. P, NP und polynomielle Reduktionen
IV.4. Satz von Cook und Levin
IV.5. einige NP-vollständige Probleme

IV.2.1 Entscheidungsprobleme

Allgemeine algorithmische Probleme

Inhalt der Komplexitätstheorie ist die Untersuchung der Schwierigkeit algorithmischer Probleme.

Beispiel (algorithmische Probleme)

- Sortieren einer Zahlenfolge
- Finden von kürzesten Pfaden in einem Graphen
- Finden von Zyklen in einem Graphen, die alle Knoten beinhalten

Lösungen für solche Probleme können viele Formen haben.

Beispiel (Lösungen für algorithmische Probleme)

- eine sortierte Zahlenfolge
- ein Pfad
- ein Zyklus

Entscheidungsprobleme

Um die Untersuchung zu vereinfachen, beschränkt man sich in der Komplexitätstheorie meist auf Entscheidungsprobleme: Probleme, bei denen die „Lösung“ eine Antwort Ja oder Nein ist.

Beispiel (Entscheidungsprobleme)

- Ist die gegebene Folge sortiert?
- Gibt es einen Pfad von u nach v mit Kosten höchstens K?
- Gibt es einen Zyklus im Graphen, der alle Knoten beinhaltet?

Idee: Finde zu einem interessierenden allgemeinen Problem $\mathcal{P}$ ein zugehöriges Entscheidungsproblem $\mathcal{E}$, so dass gilt:

Ein effizienter Algorithmus für $\mathcal{P}$ existiert genau dann, wenn ein effizienter Algorithmus für $\mathcal{E}$ existiert.

Allgemeine vs. Entscheidungsprobleme (1)

Beispiel (kürzeste Pfade)

$\mathcal{P}$: Optimierungsproblem ShortestPathOpt

- Eingabe: gerichteter, gewichteter Graph $G = \langle V, E, w \rangle$ mit positiven Kantengewichten $w : E \to \mathbb{N}_1$, Knoten $u \in V$, $v \in V$.
- Ausgabe: ein Pfad von u nach v mit minimalen Kosten bzw. Meldung, dass kein Pfad existiert

$\mathcal{E}$: Entscheidungsproblem ShortestPathDec

- Eingabe: gerichteter, gewichteter Graph $G = \langle V, E, w \rangle$ mit positiven Kantengewichten $w : E \to \mathbb{N}_1$, Knoten $u \in V$, $v \in V$, Kostenschranke $K \in \mathbb{N}_0$
- Frage: Gibt es einen Pfad von u nach v mit Kosten $\leq K$?

Allgemeine vs. Entscheidungsprobleme (2)

Satz (ShortestPathOpt ∼ ShortestPathDec)

ShortestPathOpt *und* ShortestPathDec
können aufeinander zurückgeführt werden.

Aus einem polynomiellen Algorithmus für das eine Problem kann man einen polynomiellen Algorithmus für das jeweils andere Problem konstruieren.

Beweis.

⇝ Tafel □

Hamiltonkreis

Definition (Hamiltonkreis)

Sei $G = \langle V, E \rangle$ ein (gerichteter oder ungerichteter) Graph.

Ein Hamiltonkreis in G ist eine Folge von Knoten $\pi = v_0, \ldots, v_n \in V$ mit folgenden Eigenschaften:

- π ist ein Pfad: für alle $0 \leq i < n$ führt eine Kante von v_i nach v_{i+1}
- π ist ein Kreis: $v_0 = v_n$
- π ist einfach: $v_i \neq v_j$ für alle $i \neq j$ mit $i, j < n$
- π ist hamiltonsch: alle Knoten von V sind in π enthalten

Allgemeine vs. Entscheidungsprobleme (3)

Beispiel (Hamiltonkreise in gerichteten Graphen)

$\mathcal{P}$: Suchproblem DirHamiltonCycleGen

- Eingabe: gerichteter Graph $G = \langle V, E \rangle$
- Ausgabe: ein Hamiltonkreis in G bzw. Meldung, dass keiner existiert

$\mathcal{E}$: Entscheidungsproblem DirHamiltonCycle

- Eingabe: gerichteter Graph $G = \langle V, E \rangle$
- Frage: Enthält G einen Hamiltonkreis?

Allgemeine vs. Entscheidungsprobleme (4)

Satz (DirHamiltonCycleGen ∼ DirHamiltonCycle)

DirHamiltonCycleGen *und* DirHamiltonCycle
können aufeinander zurückgeführt werden.

Aus einem polynomiellen Algorithmus für das eine Problem kann man einen polynomiellen Algorithmus für das jeweils andere Problem konstruieren.

Beweis.

⇝ Tafel □

IV.2.2 Nichtdeterminismus

Nichtdeterminismus (1)

- Um die Komplexitätstheorie zu entwickeln, benötigen wir ein neues algorithmisches Konzept: Nichtdeterminismus.
- Alle Bestandteile deterministischer Algorithmen sind auch in nichtdeterministischen Algorithmen erlaubt: **IF**, **WHILE**, etc.
- Zusätzlich gibt es eine nichtdeterministische Zuweisung:

  GUESS $x_i \in \{0, 1\}$

  Hierbei ist x_i eine Programmvariable.

Nichtdeterminismus (2)

- Bedeutung von **GUESS**:
  x_i wird entweder der Wert 0 oder der Wert 1 zugewiesen
- Damit ist nicht mehr eindeutig festgelegt,
  wie sich das Programm für eine gegebene Eingabe verhält:
  es gibt mehrere Möglichkeiten, was passieren kann.
- Das Programm berechnet den Wert x,
  wenn mindestens ein Ausführungspfad existiert,
  bei dem der Wert x berechnet wird.
- Das Programm berechnet keinen Wert
  (d. h. das Ergebnis ist undefiniert),
  wenn kein Ausführungspfad terminiert.

Nichtdeterminismus (3)

- Im Allgemeinen ist möglich, dass bei derselben Eingabe verschiedene Werte berechnet werden können.
- Das ist in Ordnung, sofern alle diese Werte korrekte Lösungen sind, z. B. mehrere unterschiedliche Hamiltonkreise bei DirHamiltonCycleGen.

Nichtdeterminismus (4)

- Wir verwenden im Folgenden im Pseudo-Code auch Anweisungen wie

  GUESS $x \in \{0, 1, 2, \ldots, 2^k - 1\}$

  oder

  GUESS $x \in S$

  für eine Menge S.
- Solche Anweisungen sind Kurzschreibweisen und lassen sich in k bzw. $\lceil \log_2 |S| \rceil$ „atomare“ **GUESS**-Anweisungen zerlegen.

Nichtdeterminismus für das Akzeptieren von Sprachen

Wir betrachten im Folgenden nichtdeterministische Verfahren für das Akzeptieren von Sprachen (= Entscheidungsproblemen).

- Algorithmus akzeptiert eine Eingabe oder lehnt sie ab:
  - Anweisung **ACCEPT** akzeptiert und beendet Programm
  - Anweisung **REJECT** lehnt ab und beendet Programm
- Bei nichtdeterministischen Programmen gilt Eingabe als akzeptiert, wenn mindestens ein Berechnungspfad akzeptiert.

⇝ Asymmetrie zwischen Akzeptieren und Verwerfen (vgl. Konzept der Semi-Entscheidbarkeit)

Beispiel: nichtdeterministischer Algorithmus (1)

Beispiel (DirHamiltonCycle)

Eingabe: gerichteter Graph $G = \langle V, E \rangle$

$start$:= ein beliebiger Knoten aus V
$current := start$
$remaining := V \setminus \{start\}$
WHILE $remaining \neq \emptyset$:
    GUESS $next \in remaining$
    IF $\langle current, next \rangle \notin E$:
        REJECT
    $remaining := remaining \setminus \{next\}$
    $current := next$
IF $\langle current, start \rangle \in E$:
    ACCEPT

Beispiel: nichtdeterministischer Algorithmus (2)

- bei geeigneten Datenstrukturen löst der Algorithmus das Problem in $O(n \log n)$ Programmschritten
- Wie viele Schritte würde ein deterministischer Algorithmus benötigen?

Raten und Prüfen

- Das DirHamiltonCycle-Beispiel illustriert ein allgemeines Design-Prinzip für nichtdeterministische Algorithmen:

  Raten und Prüfen („guess and check“)
- Im Allgemeinen können nichtdeterministische Algorithmen ein Problem lösen, indem sie zuerst eine „Lösung“ raten und dann überprüfen, dass diese tatsächlich eine Lösung ist.
- Wenn Lösungen zu einem Problem effizient überprüfbar sind, dann kann das Problem auch effizient gelöst werden, sofern Nichtdeterminismus verwendet werden darf.

IV.2.3 Nichtdeterministische Turingmaschinen

Deterministische und nichtdeterministische TMs

- Wir führen nun Nichtdeterminismus für Turingmaschinen ein.
- Gleichzeitig wiederholen wir die Funktionsweise von TMs.
- Im Folgenden unterscheiden wir zwischen
  - deterministischen Turingmaschinen (DTMs), die wir schon kennen, und
  - nichtdeterministischen Turingmaschinen (NTMs).
- Wenn wir von Turingmaschinen (TMs) sprechen, meinen wir im Folgenden beide Varianten.

Deterministische Turingmaschinen: Definition

Wiederholung:

Definition (Deterministische Turingmaschine)

Eine Deterministische Turingmaschine (DTM) ist ein 7-Tupel $M = \langle Z, \Sigma, \Gamma, \delta, z_0, \square, E \rangle$ mit:

- Z endliche, nicht-leere Menge von Zuständen
- $\Sigma \neq \emptyset$ endliches Eingabealphabet
- $\Gamma \supset \Sigma$ endliches Bandalphabet
- $\delta : (Z \setminus E) \times \Gamma \to Z \times \Gamma \times \{\mathrm{L}, \mathrm{R}, \mathrm{N}\}$ Übergangsfunktion
- $z_0 \in Z$ Startzustand
- $\square \in \Gamma \setminus \Sigma$ Blank-Zeichen
- $E \subseteq Z$ Endzustände

Nichtdeterministische Turingmaschinen: Definition

Definition (nichtdeterministische Turingmaschine)

Eine nichtdeterministische Turingmaschine (NTM) ist ein 7-Tupel $M = \langle Z, \Sigma, \Gamma, \delta, z_0, \square, E \rangle$ mit:

- Z endliche, nicht-leere Menge von Zuständen
- $\Sigma \neq \emptyset$ endliches Eingabealphabet
- $\Gamma \supset \Sigma$ endliches Bandalphabet
- $\delta \subseteq ((Z \setminus E) \times \Gamma) \times (Z \times \Gamma \times \{L, R, N\})$ Übergangsrelation
- $z_0 \in Z$ Startzustand
- $\square \in \Gamma \setminus \Sigma$ Blank-Zeichen
- $E \subseteq Z$ Endzustände

Unterschied zu DTM: für gegebenen Zustand und gelesenes Symbol mehrere verschiedene Übergänge möglich

DTMs sind ein Spezialfall von NTMs

Beobachtung: DTMs sind NTMs

Deterministische Turingmaschinen sind ein Spezialfall von NTMs.

Die DTMs sind diejenigen NTMs, bei denen für alle Paare $p = \langle s, a \rangle \in (Z \setminus E) \times \Gamma$ genau ein Tripel $t = \langle s', a', y \rangle$ existiert, so dass $\langle p, t \rangle \in \delta$.

- $\rightsquigarrow$ Es reicht, Begriffe wie Berechnungsschritte etc. für NTMs zu definieren.
- $\rightsquigarrow$ Definition für DTMs folgt automatisch.

NTMs: Konfigurationen und Berechnungen

Wie arbeiten nichtdeterministische Turingmaschinen?

- Konfiguration: $\alpha z \beta$ mit $\alpha \in \Gamma^*$, $z \in Z$, $\beta \in \Gamma^+$
- Berechnungsschritt: $c \vdash c'$, wenn aus Konfiguration c in einem Berechnungsschritt Konfiguration c' entstehen kann (siehe nächste Folie)
- Berechnung mit n Schritten: $c \vdash^n c'$ (mit $n \in \mathbb{N}_0$), wenn aus Konfiguration c in n Schritten Konfiguration c' entstehen kann ($c = c_0 \vdash c_1 \vdash c_2 \vdash \cdots \vdash c_{n-1} \vdash c_n = c'$)
- Berechnung: $c \vdash^* c'$, wenn $c \vdash^n c'$ für irgendein $n \in \mathbb{N}_0$

NTMs: Berechnungsschritte

Definition (Berechnungsschritt einer NTM)

Sei $M = \langle Z, \Sigma, \Gamma, \delta, z_0, \square, E \rangle$ eine NTM.

Die folgenden Regeln definieren, wann Konfiguration c von M in Konfiguration c' von M übergehen kann, geschrieben $c \vdash c'$.

Es gelte $\langle \langle z, a \rangle, \langle z', a', y \rangle \rangle \in \delta$.

$$
\begin{array}{rl}
\alpha z a \beta \vdash \alpha z' a' \beta & \text{wenn } y = N \\
\alpha z a \beta \vdash \alpha a' z' \beta & \text{wenn } y = R, |\beta| \geq 1 \\
\alpha z a \vdash \alpha a' z' \square & \text{wenn } y = R \\
\alpha b z a \beta \vdash \alpha z' b a' \beta & \text{wenn } y = L \\
z a \beta \vdash z' \square a' \beta & \text{wenn } y = L
\end{array}
$$

Dabei seien $a, a', b \in \Gamma$, $\alpha, \beta \in \Gamma^*$, $z \in Z \setminus E$, $z' \in Z$.

Berechnungen von NTMs: Vorbemerkungen

- Wir könnten jetzt einen Berechnungsbegriff für NTMs analog zu DTMs einführen.
- Da wir uns hier auf Entscheidungsprobleme beschränken, benötigen wir aber keinen allgemeinen Berechnungsbegriff.
- Es reicht ein Akzeptanzbegriff:
  - eine gegebene Eingabe $w \in \Sigma^*$ wird entweder akzeptiert (d. h. eine Berechnung endet in einem Endzustand)
  - oder nicht akzeptiert (keine solche Berechnung existiert).

Berechnungen von NTMs: Akzeptanz

Definition (Akzeptanz von Wörtern in Zeit n)

Sei $M = \langle Z, \Sigma, \Gamma, \delta, z_0, \Box, E \rangle$ eine NTM.

Ein Wort $w \in \Sigma^*$ wird von M in Zeit n akzeptiert, wenn gilt:

$$c_0 \vdash^k c_e \text{ für eine Zahl } k \in \mathbb{N}_0 \text{ mit } k \leq n,$$

wobei c_0 die Startkonfiguration für w ist ($c_0 = z_0 w$ ausser für den Spezialfall $w = \epsilon$, wo $c_0 = z_0 \Box$) und c_e eine Endkonfiguration ist (d. h. eine Konfiguration, bei der die TM in einem Zustand $z_e \in E$ ist).

Anmerkung: Es reicht also aus, wenn eine akzeptierende Berechnung der Länge höchstens n existiert!

IV.2.4 Ausblick

NTMs vs. DTMs

- NTMs sind sehr mächtig, da sie den „richtigen“ Berechnungsschritt „raten“ können.
- Oder anders interpretiert: sie durchlaufen viele mögliche Berechnungen „parallel“, und es reicht, wenn eine davon erfolgreich ist.
- Können sie Probleme effizient (in polynomieller Zeit) lösen, die DTMs nicht effizient lösen können?
- Das ist die grosse Frage!