

Grundlagen Theoretischer Informatik 2

WiSe 2011/12 in Trier

Henning Fernau

Universität Trier

fernau@uni-trier.de

Grundlagen Theoretischer Informatik 2 Gesamtübersicht

- Organisatorisches; Einführung
- Ersetzungsverfahren: Grammatiken und Automaten
- Rekursionstheorie-Einführung
- **Komplexitätstheorie-Einführung**

Rekursions- vs. Komplexitätstheorie

Rekursionstheorie ist die Theorie des prinzipiell Machbaren auf Computern. Ob eine praktische Implementierung eines Verfahrens möglich ist, interessiert hier nicht.

So lässt sich die Ackermann-Funktion sehr kurz notieren und auch rasch implementieren, “benutzbar” ist sie deshalb lange nicht.

Hinweis: Spezialvorlesung Rekursions- und Lerntheorie im Master-Programm

Komplexitätstheorie ist die Theorie des Machbaren mit beschränktem Aufwand. Es scheint sinnvoll, den Ressourcenverbrauch in Abhängigkeit von der Eingabe zu “messen”.

Hinweis: Spezialvorlesung Komplexitätstheorie im Master-Programm

Neues Ziel: **Aufwand der Lösung von Problemen untersuchen**

Wichtigste Ressourcen: Rechenzeit und Speicherplatz der Lösungsalgorithmen

Obere Schranken für ein Problem:

- Untersuche *einen* Lösungsalgorithmus und
- schätze Bedarf an Rechenzeit und Speicherplatz ab

Untere Schranken für ein Problem (**schwierig. . .**):

- Gültig für *alle* Lösungsalgorithmen.

Hier nur: Zeitkomplexität von Berechnungsproblemen

Oft **Trade-off** zwischen Zeitbedarf und Speicherplatzbedarf bei Problemen:

- schnelle Algorithmen mit hohem Speicherplatzbedarf oder
- langsame Algorithmen auf wenig Speicherplatz.

Für eine deterministische Mehrband-Turingmaschine M sei

$$\text{time}_M(w) = \text{Schrittzahl von } M \text{ bei Eingabe von } w$$

(*Schrittzahl*: Zahl Übergänge von Startkonfig. bis Berechnungsende)

Sei $f : \mathbb{N} \rightarrow \mathbb{N}$ eine totale Zahlenfunktion.

- Die Klasse $\text{FTIME}(f)$ besteht aus allen totalen Wortfunktionen $g : E^* \rightarrow E^*$ über bel. Alphabet E derart, dass es eine deterministische Mehrband-Turingmaschine M mit Eingabealphabet E gibt, die die Funktion g berechnet und bei Eingabe eines Wortes w nach höchstens $f(|w|)$ Schritten anhält, d.h. $\text{time}_M(w) \leq f(|w|)$ für jedes Eingabewort w erfüllt.
- Die Klasse $\text{TIME}(f)$ besteht aus allen Sprachen L über irgendeinem Alphabet E derart, dass es eine deterministische Mehrband-Turingmaschine M mit Eingabealphabet E gibt, die $L(M) = L$ und $\text{time}_M(w) \leq f(|w|)$ für jedes Eingabewort w erfüllt.

Zeitmessung in der Diskussion

(1) Zeitbedarf wird in Abhängigkeit von Wortlänge $|w|$ gemessen, nicht von w selbst.

(2) Mehrbandmaschinen liefern realistischere Zeitkomplexitäten als Einbandmaschinen.

Erinnerung: Arbeitet Mehrbandmaschine in Zeit $O(f(n))$, so gibt es äquivalente Einbandmaschine in Zeit $O(f^2(n))$. (Spurenkonstruktion)

(3) M muss in allen Fällen anhalten.

Bei $w \notin L(M)$ also keine Endlosschleifen möglich,
d.h. M muß dann in einem Nichtendzustand steckenbleiben.

(4) Komplexität von Zahlenfunktionen $h : \mathbb{N} \rightarrow \mathbb{N}$

über korrespondierende Wortfunktion $g : E^* \rightarrow E^*$,
die für jede Zahl n das Wort $\text{bin}(n)$ auf $\text{bin}(h(n))$ abbildet

(5) Bezeichnung: **Bitkomplexität**: Jeder Übergang ändert Konfiguration der TM 'um konstant viele Bits'

(6) Alternativ: **uniforme Komplexität** (vorherrschend in der Algorithmik)

Hierbei: Zählung der 'elementaren' Rechenoperationen,
die nötig sind, um $f(n)$ aus n zu berechnen.

Beispiel:

Berechne Zahl 2^{2^n} wie folgt:

Starte mit $x := 2$ und führe n -fach $x := x^2$ aus

- Uniforme Komplexität: n
- Bit-Komplexität mindestens 2^n , da 2^{2^n} bereits Länge 2^n hat!
- Uniforme Komplexität nicht immer angemessen.
- Turingmaschinenmodell / Bitkomplexität i.d.R. passender!

Weitere Diskussionen: Vorlesung Rechnerarithmetik im Master-Programm

Funktionen- und Sprachklassen

- FP sei die Menge aller in Polynomzeit berechenbaren Wortfunktionen:

$$FP = \bigcup_{p \text{ Polynom}} FTIME(p)$$

- P sei die Menge aller in Polynomzeit lösbaren Probleme:

$$P = \bigcup_{p \text{ Polynom}} TIME(p)$$

Analog: Klasse EXP der in exponentieller Zeit lösbaren Probleme,
mit Zeitschranken $2^{c \cdot n}$

Nichtdeterminismus

Für eine nichtdeterministische Turingmaschine M sei

$$\text{ntime}_M(w) = \max\{ \text{Schrittzahl irgendeiner Rechnung von } M \\ \text{bei Eingabe von } w \quad \}$$

- Sei $f : \mathbb{N} \rightarrow \mathbb{N}$ eine totale Zahlenfunktion.

Die Klasse $\text{NTIME}(f)$ besteht aus allen Sprachen L über irgendeinem Alphabet E derart, dass es eine nichtdeterministische Mehrband-Turingmaschine M gibt mit $L(M) = L$ und $\text{ntime}_M(w) \leq f(|w|)$ für alle Eingaben w .

- $\text{NP} = \bigcup_{p \text{ Polynom}} \text{NTIME}(p)$

Nichtdeterminismus—Anmerkungen

TM M nichtdeterministisch \rightsquigarrow

- i.d.R. zu einer Eingabe w viele mögliche Berechnungen, oft unendlich lang
- $w \in L(M) \iff$ es gibt (mindestens) eine (endliche) akzeptierende Berechnung auf w
- $\text{NTIME}(f)$: *jede* Berechnung hält nach maximal $f(|w|)$ Schritten
Keine unendlichen Berechnungen erlaubt, M hält *stets!*
- Halt in Endzustand: Eingabewort w akzeptiert.
- Halt (Steckenbleiben) in Nicht-Endzustand: w nicht akzeptiert.

Guess and Check

- “Nichtdeterministische Algorithmen” erscheint zunächst als eigenümliches Konzept.
- Wie kann man solche Algorithmen entwerfen?
- Alternative Kennzeichnung von NP durch 2-Phasen-Algorithmus:
- Phase 1: Rate (polynomiell viele) Bits
- Phase 2: Führe deterministische Berechnung aus mit Hilfe der geratenen Bits.
Antworte JA, falls dies die Berechnung liefert.
- NEIN wird hier NIE geliefert, das wäre implizit durch Schrittmitzählen zu erledigen.

Guess and Check: Ein Beispiel

Erfüllbarkeitsproblem der Aussagenlogik (SAT)

(formaler auf der nächsten Folie)

Gibt es erfüllende Belegung der Booleschen Variablen, z.B. für den Ausdruck

$$(x_1 \wedge \neg x_2) \vee (x_1 \wedge x_3) \vee (\neg x_2 \wedge x_3)?$$

Guess and Check:

1. Rate n -Bitvektor (für die vorkommenden n Variablen x_1, \dots, x_n).
2. Verifiziere durch “Ausrechnen”, ob “richtig geraten” wurde.

(Kontextfreie) Sprache \mathcal{F} der aussagenlogischen Formeln:

- Jede Variable x_i ist eine Formel (mit $i \in \mathbb{N}$), setze $V := \{x_i \mid i \in \mathbb{N}\}$.
- Sind F, G Formeln, so auch $\neg F$, $(F \wedge G)$ und $(F \vee G)$.

Formeln können *ausgewertet* werden, indem Variablen boolesche Werte zugewiesen werden:

- Eine *Belegung* ϕ ist eine Abbildung $\phi : V \rightarrow \{0, 1\}$; Deutung 0 “falsch”, 1 “wahr”.
- ϕ kann auf Formeln erweitert werden durch

$$\begin{aligned}\phi(\neg F) &= 1 - \phi(F) \\ \phi((F \wedge G)) &= \min\{\phi(F), \phi(G)\} \\ \phi((F \vee G)) &= \max\{\phi(F), \phi(G)\}\end{aligned}$$

Übung: Gib kfG an, die \mathcal{F}_n (über festem Variablenalphabet $V_n = \{x_1, \dots, x_n\}$) beschreibt.

Alternative Darstellung der Auswertung aussagenlogischer Formeln und ihres Aufbaus

Ein Boolescher Ausdruck (BA) besteht aus Booleschen Variablen aus V , Klammern, sowie den logischen Operatoren \wedge , \vee und \neg . Ist eine Belegung $\phi : V \rightarrow \{0, 1\}$ vorgegeben, so können BAs wie folgt (rekursiv) ausgewertet werden:

1. Variablen selbst sind BAs; ihr Wert ergibt sich unmittelbar aus ϕ .
2. Sind E_1 und E_2 BAs, so ist $(E_1 \wedge E_2)$ ein BA, der zu 1 auswertet, wenn sowohl E_1 als auch E_2 den Wert 1 liefern.
3. Sind E_1 und E_2 BAs, so ist $(E_1 \vee E_2)$ ein BA, der zu 1 auswertet, wenn E_1 oder auch E_2 den Wert 1 liefert.
4. Ist E ein BA, so auch $\neg E$; dieser liefert 1 gdw. E zu 0 auswertet.

Diese Darstellung erleichtert vermutlich die Übungsaufgabe.

(Kontextfreie) Sprache \mathcal{F} der aussagenlogischen Formeln:

Umgang mit beliebig vielen Variablen:

Kodiere Variablen x_i binär durch $x \text{ bin}(i)$, dann gilt:

- Für jede aussagenlogische Formel F ist $\text{code}(F)$ Wort über

$$E = \{0, 1, x, (,), \neg, \wedge, \vee\}.$$

- Hat F Länge m und n Variablen, dann gilt $|\text{code}(F)| \leq m \log n$.

Erfüllbarkeitsproblem der Aussagenlogik

$\text{SAT} := \{\text{code}(F) \text{ aus } E^* \mid F \text{ erfüllbare Formel der Aussagenlogik}\}$

(Eine Formel F heie *erfüllbar*, wenn es Belegung ϕ gibt mit $\phi(F) = 1$.)

In noch zu beschreibendem Sinne ist SAT zentral für NP.

P versus NP

Nach Definition der Turingmaschinen sofort $P \subseteq NP$, offen jedoch:

P-NP-Problem: $P \stackrel{?}{=} NP$

- berühmtestes Problem der theoretischen Informatik.
- Lösung wertvoll, Preisgeld 1 Million US-\$
vgl. <http://www.claymath.org/millennium>
- Grund: viele *praktisch relevante Probleme* liegen in NP, für die keine brauchbaren Algorithmen bekannt sind (d.h. unbekannt ist, ob sie in P liegen)

Spezielle große Problemklasse: *NP-vollständige Probleme*

Liegt auch nur ein NP-vollständiges Problem auch in P, so ist $P = NP$.

Liegt auch nur ein NP-vollständiges Problem *nicht* in P, so ist $P \neq NP$.

Arbeitshypothese (z.B. für Kryptographie): Eher $P \neq NP$ als $P = NP$...

Wie teuer ist es, Nichtdeterminismus zu eliminieren?

Satz: Die Klasse NP ist enthalten in $\bigcup_p \text{Polynom TIME}(2^{p(n)})$.

Beweisskizze: Betrachte $L \in \text{NP}$.

- M : nichtdeterministische Turingmaschine mit $L(M) = L$.
- M arbeite auf Eingabe w in Zeit $\leq q(|w|)$ für Polynom q .
- c sei Maximalzahl möglicher Nachfolgekonfigurationen einer Konfiguration.

Betrachte Berechnungsbaum B_w von M auf w .

- Knoten in B_w entsprechen Konfig. von M .
- Kante (K, K') in B_w Übergang $K \vdash_M K'$.
- Jeder Knoten in B_w hat Grad $\leq c$.
- B_w hat Tiefe $\leq q(|w|) \rightsquigarrow$
 B_w hat höchstens $c^{q(|w|)}$ Knoten.

Deterministischer Algorithmus für L :

- Durchsuche B_w , ob akzeptierende Konfiguration enthalten ist.
- Wenn ja, akzeptiere w .
- Wenn nein, verwerfe w .

Da $\leq c^{q(|w|)}$ Knoten: Zeitaufwand $2^{p(|w|)}$ für Polynom p .
Damit

$$L \in \text{TIME}(2^{p(n)})$$

Vergleich LOOP-Berechenbarkeit

Erinnerung: Auch LOOP-berechenbare Funktionen sind stets total.

Satz: Ist $f : \mathbb{N} \rightarrow \mathbb{N}$ eine LOOP-berechenbare Funktion, so ist jede Sprache L aus $\text{TIME}(f)$ LOOP-berechenbar.

Beweisskizze: Sei L aus $\text{TIME}(f)$.

- Betrachte Turingmaschine M mit $L = L(M)$.
- M arbeite in Zeit $f(|w|)$ bei Eingaben w .

Konstruiere aus M neue Turingmaschine M' mit:

- M' berechnet aus einer Eingabe $n \in \mathbb{N}$ (genauer: aus $u \in \{0, 1\}^*$ mit $n = \text{bin}(u)$) zunächst w mit $w = v(n) = v(\text{bin}(u))$.
- Danach wendet M' die Maschine M auf w an.
- Akzeptiert M , so gibt M' eine 1 aus, sonst eine 0.

Vergleich LOOP-Berechenbarkeit (Forts.)

- M' berechnet die charakteristische Funktion von $\nu^{-1}(L)$.
- Achtung: Rechenzeit von M' wird gemessen in $|u|$, d.h. in $\log_2 n$.
- M' arbeitet in Zeit

$$f'(|u|) := f(|w|) + (\text{Zeit zur Umwandlung } n \mapsto w).$$

- Da f LOOP-berechenbar ist, gibt es LOOP-berechenbares g mit $f'(|u|) \leq g(n)$.

Beispiele:

- Alle Polynome p sind LOOP-berechenbar.
- Alle Funktionen der Form $2^{p(n)}$ (mit Polynom p) sind LOOP-berechenbar.

Damit sind alle Mengen in P oder NP auch LOOP-berechenbar!

Ein angepasster Reduktionsbegriff

Seien A und B Sprachen über einem Alphabet E .

Dann heißt A auf B *polynomial reduzierbar*,
wenn es eine in Polynomzeit berechenbare Funktion $f : E^* \rightarrow E^*$ gibt,
so dass für alle $w \in E^*$ gilt:

$$w \in A \iff f(w) \in B$$

Schreibweise: $A \leq_p B$

\rightsquigarrow Dies ist der aus der Rekursionstheorie bekannte Reduktionsbegriff,
angereichert um Polynomzeit-Effizienz.

Abschluss nach unten

- Falls $A \leq_p B$ und $B \in P$, so ist auch $A \in P$.
- Falls $A \leq_p B$ und $B \in NP$, so ist auch $A \in NP$.

Sei $A \leq_p B \in P$ mit Reduktionsfunktion f :

- M_B sei Turingmaschine M_B mit $L(M_B) = B$
- M_B arbeite in Zeit q für Polynom q
- M_f berechne f
- M_f arbeite in Zeit p für Polynom p

Zeitbedarf von M : $\text{time}_M(w) \leq p(|w|) + q(|f(w)|)$.

Mit $|f(w)| \leq p(|w|) + |w|$ also

$$\text{time}_M(w) \leq p(|w|) + q(|f(w)|) \leq p(|w|) + q(p(|w|) + |w|)$$

d.h. polynomial in $|w|$.

Betrachte folgende Maschine M :

- Aus Eingabe w berechnet M zuerst $v := f(w)$ durch Simulation von M_f
- M testet, ob $v \in B$ durch Simulation von M_B
- M akzeptiert w genau dann, wenn v dabei von M_B akzeptiert wird

Damit sofort $L(M) = A$

Eigenschaften von \leq_p

Satz: Die Relation \leq_p ist transitiv und reflexiv, aber nicht antisymmetrisch.

Beweis: Transitivität, d.h. z.z.: aus $A \leq_p B \leq_p C$ folgt $A \leq_p C$.

Das sieht man wie im Beweis auf der vorigen Folie ein.

Berechne nämlich Komposition der Reduktionsfunktionen.

Reflexivität, d.h.: $A \leq_p A$ trivial.

Nicht antisymmetrisch: d.h. $(A \leq_p B \wedge B \leq_p A) \rightarrow A = B$ gilt nicht.

Daher sinnvoller Äquivalenzbegriff: $A \equiv_p B$ gdw. $A \leq_p B$ und $B \leq_p A$.

Wir werden nämlich sehen:

Viele Probleme sind mit SAT äquivalent, aber nicht damit identisch.

Hart und weich...

- Eine Sprache L heißt *NP-hart*, wenn *alle* Sprachen aus NP auf sie polynomial reduzierbar sind.
- Eine Sprache L heißt *NP-vollständig*, wenn sie aus NP ist und NP-hart ist.

Damit: NP-vollständige Probleme sind schwierigste Probleme in NP!

Einer für alle...

Satz: L sei NP-vollständig. Dann folgt: $L \in P \iff P = NP$.

Beweis von ' \Leftarrow ': trivial

Beweis von ' \Rightarrow ': Sei $L \in P$ und NP-vollständig.

Betrachte beliebiges $L' \in NP$, damit $L' \leq_p L$.

Mit Lemma über den "Abschluss nach unten" folgt also $L' \in P$.

Damit gilt $NP \subseteq P$; Inklusion $P \subseteq NP$ gilt nach Definition.

Beispiele für Reduktionen 1

Eine aussagenlogische Formel ist in *Literal-Normalform*, wenn jede in ihr vorkommende Negation unmittelbar vor einer Variablen steht.

Unter einem *Literal* versteht man auch eine Variable oder ihr negiertes Vorkommen. Bei der Literal-NF kommen Negationen nur innerhalb von Literalen vor.

LIT – SAT: SAT, eingeschränkt auf Formeln in Literal-NF.

Lemma: $SAT \leq_p \text{LIT} - SAT$.

Erinnerung: *De Morgans Gesetze:*

(1) $\overline{\neg(x \vee y)} \equiv (\neg x \wedge \neg y)$ und (2) $\overline{\neg(x \wedge y)} \equiv (\neg x \vee \neg y)$.

Wiederholte Anwendung dieser Gesetze (von “links nach rechts”) führt schließlich auf Literal-NF.

Formaler Beweis?

Polynomielle Laufzeit?

Beispiele für Reduktionen 2

Klausel: Disjunktion von Literalen

konjunktive Normalform (KNF): Konjunktion von Klauseln

KNF – SAT: SAT, eingeschränkt auf Formeln in KNF.

Bsp.: $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$ ist in KNF.

Lemma: $\text{LIT} - \text{SAT} \leq_p \text{KNF} - \text{SAT}$

Erinnerung: Sind V_1 und V_2 Variablenmengen mit $V_1 \subseteq V_2$, so heißt eine Belegung $\phi_2 : V_2 \rightarrow \{0, 1\}$ *Erweiterung* der Belegung $\phi_1 : V_1 \rightarrow \{0, 1\}$ falls $\phi_1(x) = \phi_2(x)$ für alle $x \in V_1$. Dann heißt ϕ_1 auch *Einschränkung* von ϕ_2 auf V_1 .

Beispiele für Reduktionen 2

Lemma: $\text{LIT} - \text{SAT} \leq_p \text{KNF} - \text{SAT}$

Beweis: (Induktion über den rekursiven Aufbau von BAs)

1. BAs, die nur aus Literalen bestehen, sind bereits in KNF. (trivialer Induktionsanfang)
2. Ist BA E von der Form $(E_1 \wedge E_2)$, so gibt es nach IV KNF-Ausdrücke E'_1 und E'_2 , die äquivalent zu E_1 bzw. E_2 sind. $\sim \rightarrow E'_1 \wedge E'_2$ ist äquivalent zu $E_1 \wedge E_2$ und in KNF.
3. Ist BA E von der Form $(E_1 \vee E_2)$, so gibt es nach IV KNF-Ausdrücke E'_1 und E'_2 , die äquivalent zu E_1 bzw. E_2 sind. Hierbei sei

$$E'_j = (C_{j,1} \wedge \dots \wedge C_{j,r_j}).$$

Sei y eine neue Variable und setze $E' := \tilde{E}_1 \wedge \tilde{E}_2$ mit

$$\tilde{E}_j = \tilde{C}_{j,1} \wedge \dots \wedge \tilde{C}_{j,r_j}$$

wobei $\tilde{C}_{1,i} := (y \vee C_{1,i})$ und $\tilde{C}_{2,i} := (\neg y \vee C_{2,i})$.

E' ist offenbar in KNF. Logische Äquivalenz ist auch klar ?!

Erfüllende Belegung ϕ für LIT-SAT BA E über Variablenmenge V würde sich durch Einschränkung der erfüllenden Belegung ϕ' für äquivalenten KNF-SAT BA E' über $V' \supseteq V$ ergeben.

Beobachte: Polynomielle Rechenzeit der entsprechenden Prozedur!

Angewendete Transitivität: $SAT \leq_p KNF - SAT$

Beispiel: Betrachte

$$\neg(\neg(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2))$$

De Morgan überführt in LIT-SAT:

$$E = \underbrace{(x_1 \vee x_2)}{=:E_1} \vee \underbrace{(x_1 \wedge \bar{x}_3)}{=:E_2}$$

E_1 wird sodann transformiert in

$$E'_1 = (x_1 \vee y_1) \wedge (x_2 \vee \neg y_1),$$

während $E'_2 = E_2$.

$$\rightsquigarrow E' = (x_1 \vee y_1 \vee y_2) \wedge (x_2 \vee \neg y_1 \vee y_2) \wedge (x_1 \vee \neg y_2) \wedge (\neg x_3 \vee \neg y_2)$$

$x_1 = 0$, $x_2 = 1$ und $x_3 = 1$ ist eine erfüllende Belegung ϕ für E .

Mit $y_1 = 1$ und $y_2 = 0$, kann man ϕ erweitern, um E' zu erfüllen.

3-SAT (schwache Def.); Beispiele für Reduktionen 3

Ein BA in KNF heißt in (schwacher) **3-SAT-NF**, falls jede Klausel höchstens drei Literale umfasst.

Hieraus ergibt sich das Entscheidungsproblem 3 – SAT.

Lemma: $\text{KNF – SAT} \leq_p \text{3 – SAT}$

WHILE E contains a “large” clause with more than 3 literals DO
 Pick some “large” clause $C = l_1 \vee \dots \vee l_k$.
 Choose a “new” variable y .
 Replace C in E by
 1. $C_{\text{short}} = l_1 \vee l_2 \vee y$ and
 2. $C_{\text{tail}} = \neg y \vee l_3 \vee \dots \vee l_k$
OD

Beachte: C_{tail} ist kürzer als C!

Formalisieren Sie dieses Argument zur Übung! Warum Pol.-Zeit-Reduktion?
Wie verhalten sich die Belegungen “zueinander”?

3-SAT (starke Def.); Beispiele für Reduktionen 4

Ein BA in KNF heißt in (starker) **3-SAT-NF**, falls jede Klausel genau drei Literale umfasst.

Hieraus ergibt sich das Entscheidungsproblem 3 – SAT'.

Lemma: $3 - \text{SAT} \leq_p 3 - \text{SAT}'$

WHILE E contains a “small” clause with less than 3 literals DO
 Pick some “small” clause C.
 Choose a “new” variable y.
 Replace C in E by
 1. $C_1 = C \vee y$ and
 2. $C_2 = C \vee \neg y$
OD

Beachte: C_1 und C_2 sind kürzer als C!

Formalisieren Sie dieses Argument zur Übung! Warum Pol.-Zeit-Reduktion?