In der Praxis genügt es nicht zu wissen, daß eine Funktion berechenbar ist.

Man interessiert sich auch dafür, wie groß der Aufwand zur Berechnung ist.

Aufwand/Komplexität:

- Rechenzeit: entspricht bei TM der Anzahl der Schritte bis zum Halten
- Speicherplatz: entspricht bei TM der Anzahl der benutzten Felder

Beides soll abgeschätzt werden als Funktion in der Größe der Eingabe.

Verschiedene Fragestellungen:

- Komplexität eines speziellen Algorithmus/Programms:
  Wieviel Aufwand benötigt dieser Algorithmus?

  Algorithmen-theorie
- Komplexität eines Entscheidungsproblems:
  Wieviel Aufwand benötigt der „beste“ Algorithmus im „schlimmsten“ Fall?

  Komplexitäts-theorie

§ 18. Komplexitätsklassen

Komplexitätsklasse ist Klasse von Entscheidungsproblemen, die mit einer bestimmten ‘Menge’ einer Ressource gelöst werden können

Es gibt Zeitkomplexitätsklassen und Platzkomplexitätsklassen

Besonders wichtig ist der Unterschied zwischen polynomiellem und exponentiellem Ressourcenverbrauch:

Eingabegröße	1	2	3	4	5	6	7	8		20	128
n^2	1	4	9	16	25	36	49	64		400	16.384
2^n	2	4	8	16	32	64	128	256		1.048.576	RIESIG!

Bei der Definition von Komplexitätsklassen macht die Unterscheidung deterministisch / nicht-deterministisch einen erheblichen Unterschied!

Definition 18.1 ($f(n)$-zeitbeschränkt, $f(n)$-platzbeschränkt)

Es sei $f : \mathbb{N} \to \mathbb{N}$ eine Funktion und $\mathcal{A}$ eine DTM über Σ.

1. $\mathcal{A}$ heißt $f(n)$-zeitbeschränkt, falls
   - für alle Eingaben $w \in \Sigma^*$ der Länge n
   - die Maschine $\mathcal{A}$ nach $\leq f(n)$ Schritten anhält.
2. $\mathcal{A}$ heißt $f(n)$-platzbeschränkt, falls
   - für alle Eingaben $w \in \Sigma^*$ der Länge n
   - die Maschine $\mathcal{A} \leq f(n)$ Felder des Bandes benutzt.

Übertragung dieser Begriffe auf NTM:

Aufwandsbeschränkung für alle bei der Eingabe möglichen Berechnungen

Zeitbeschränkte TMs terminieren stets, platzbeschränkte nicht unbedingt.

Folgendes Schema verwenden wir zur Definition von Komplexitätsklassen

Definition 18.2

Sei $f : \mathbb{N} \to \mathbb{N}$ eine Funktion. Definiere

$$\mathsf{DTIME}(f(n)) := \{L \mid \text{es gibt eine } f(n)\text{-zeitbeschr. DTM } \mathcal{A} \text{ mit } L(\mathcal{A}) = L\}$$

$$\mathsf{NTIME}(f(n)) := \{L \mid \text{es gibt eine } f(n)\text{-zeitbeschr. NTM } \mathcal{A} \text{ mit } L(\mathcal{A}) = L\}$$

$$\mathsf{DSPACE}(f(n)) := \{L \mid \text{es gibt eine } f(n)\text{-platzbeschr. DTM } \mathcal{A} \text{ mit } L(\mathcal{A}) = L\}$$

$$\mathsf{NSPACE}(f(n)) := \{L \mid \text{es gibt eine } f(n)\text{-platzbeschr. NTM } \mathcal{A} \text{ mit } L(\mathcal{A}) = L\}$$

Interessante Klassen ergeben sich durch die Wahl geeigneter Funktionen $f(n)$.

Elementare Zusammenhänge zwischen Komplexitätsklassen:

Satz 18.3

1. $\mathsf{DTIME}(f(n)) \subseteq \mathsf{DSPACE}(f(n)) \subseteq \mathsf{NSPACE}(f(n))$
2. $\mathsf{DTIME}(f(n)) \subseteq \mathsf{NTIME}(f(n))$
3. $\mathsf{NSPACE}(f(n)) \subseteq \mathsf{DTIME}(2^{\mathcal{O}(f(n))})$

Beweis Punkt 1:

Offenbar kann man in k Schritten höchstens k Felder benutzen.
Jede DTM ist auch eine NTM.

Beweis Punkt 2:

Jede DTM ist auch eine NTM.

Beweisskizze Punkt 3: $\mathsf{NSPACE}(f(n)) \subseteq \mathsf{DTIME}(2^{\mathcal{O}(f(n))})$

Sei $\mathcal{A}$ eine $f(n)$-platzbeschränkte NTM.

Die Berechnungen von $\mathcal{A}$ auf Eingabe w der Länge n kann man als endlichen Konfigurationsgraph $G_{\mathcal{A},n} = (V_{\mathcal{A},n}, E_{\mathcal{A},n})$ darstellen:

- Knoten $V_{\mathcal{A},n}$ = Konfigurationen von $\mathcal{A}$ der Länge höchstens $f(n)$
- $(k, k') \in E_{\mathcal{A},n}$ gdw k' von k aus in einem Schritt erreicht werden kann

Offensichtlich gilt:
$\mathcal{A}$ akzeptiert w gdw. in $G_{\mathcal{A},n}$ von Knoten q_0w aus ein Knoten k erreichbar ist, so dass k akzeptierende Stoppkonfiguration.

Arbeitsweise von DTM $\mathcal{A}'$:

- Bei Eingabe w der Länge n, konstruiere Konfigurationsgraph $G_{\mathcal{A},n}$
- überprüfe, ob von q_0w aus akzeptierende Stoppkonfiguration erreichbar

Beweisskizze Punkt 3: $\mathsf{NSPACE}(f(n)) \subseteq \mathsf{DTIME}(2^{\mathcal{O}(f(n))})$

Analyse des Zeitbedarfs von $\mathcal{A}'$:

Anzahl der Konfigurationen von $\mathcal{A}$ der Länge $\leq f(n)$ ist beschränkt durch

$$\text{akonf}_{\mathcal{A}}(n) := \underbrace{|Q|}_{\#\text{ Zustände}} \cdot \underbrace{f(n)}_{\#\text{ Kopfpositionen}} \cdot \underbrace{|\Gamma|^{f(n)}}_{\#\text{ Bandbeschriftungen}}$$

Es gilt:

- $|V_{\mathcal{A},n}| = \text{akonf}_{\mathcal{A}}(n) \in 2^{\mathcal{O}(f(n))}$ und damit auch $|E_{\mathcal{A},n}| \in 2^{\mathcal{O}(f(n))}$
- Der Konfigurationsgraph $G_{\mathcal{A},n}$ kann in Zeit $2^{\mathcal{O}(f(n))}$ erzeugt werden
- Das Erreichbarkeitsproblem in gerichteten Graphen ist in Linearzeit lösbar, z.B. durch Breitensuche (siehe Bücher zu Algorithmentheorie)

Fundamentale Komplexitätsklassen:

Definition 18.4 (P, NP, PSpace, NPSpace, ExpTime)

$$\mathsf{P} := \bigcup_{p \text{ Polynom in } n} \mathsf{DTIME}(p(n))$$

$$\mathsf{NP} := \bigcup_{p \text{ Polynom in } n} \mathsf{NTIME}(p(n))$$

$$\mathsf{PSpace} := \bigcup_{p \text{ Polynom in } n} \mathsf{DSPACE}(p(n))$$

$$\mathsf{NPSpace} := \bigcup_{p \text{ Polynom in } n} \mathsf{NSPACE}(p(n))$$

$$\mathsf{ExpTime} := \bigcup_{p \text{ Polynom in } n} \mathsf{DTIME}(2^{p(n)})$$

Es folgt aus Satz 18.3:

Korollar 18.5

$$\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSpace} \subseteq \mathsf{NPSpace} \subseteq \mathsf{ExpTime}$$

Besonders wichtig sind die Komplexitätsklassen P und NP:

- Die Probleme in P werden im allgemeinen als die effizient lösbaren Probleme angesehen (engl. “tractable”, d.h. machbar)
- Im Gegensatz dazu nimmt man an, dass NP viele Probleme enthält, die nicht effizient lösbar sind (engl “intractable”).
- Die in Polynomialzeit lösbaren Probleme (Klasse P) stimmen in allen bekannten deterministischen Berechnungsmodellen überein (Erweiterte Church-Turing-These)

Die Komplexitätsklasse NP ist sehr wichtig da sie

1. sehr viele natürliche Probleme der Informatik enthält, von denen
2. angenommen wird, dass sie nicht in P sind.

Wir betrachten zwei Beispiele.

Definition 18.6 (SAT)

SAT ist die Menge der erfüllbaren aussagenlogischen Formeln über den Operatoren $\neg, \wedge, \vee$ („SAT“ für „satisfiable“).

Streng genommen geht es natürlich um eine geeignete Kodierung dieses Problems als formale Sprache.

Zum Beispiel muss man die (potentiell) unendlich vielen Aussagenvariablen $x_1, x_2, \ldots$ mittels eines endlichen Alphabets darstellen.

Lemma 18.7

$\mathsf{SAT} \in \mathsf{NP}$

Beweis:

Die NTM, die SAT in polynomieller Zeit erkennt, arbeitet wie folgt:

1. Teste, ob die Eingabe ein aussagenlogische Formel ist.

   Dies ist eine Instanz des Wortproblems für kontextfreie Grammatiken, das in Polynomalzeit lösbar ist (CYK-Algorithmus).

2. Die Variablen in der Eingabeformel φ seien $x_1, \ldots, x_n$.

   Schreibe nicht-deterministisch ein beliebiges Wort $u \in \{0,1\}$ der Länge n hinter die Eingabe auf das Band

3. Betrachte u als Belegung: $x_i \mapsto 1$ gdw. das i-te Symbol von u ist 1.

   Überprüfe deterministisch in Polyzeit, ob Belegung die Formel φ erfüllt Akzeptiere, wenn das der Fall ist; verwerfe sonst

Definition 18.8 (Clique)

Eine Clique in einem ungerichteten Graph (V, E) ist eine nichtleere Knotenmenge $C \subseteq V$ so dass der durch C induzierte Subgraph vollständig ist, also:

$$\{v, v'\} \in E \text{ für alle } v, v' \in C$$

Das Problem CLIQUE ist die Menge aller Paare (G, k) mit G Graph und $k > 0$, so dass G eine Clique der Größe k enthält.

Lemma 18.9

$\mathsf{CLIQUE} \in \mathsf{NP}$

Beweis: Polyzeit NTM für CLIQUE arbeitet wie folgt:

1. Schreibe nicht-deterministisch Knotenmenge C der Grösse k aufs Band
2. Überprüfe deterministisch und in Polyzeit, ob C eine Clique ist.

Algorithmen dieser Art formuliert man üblicherweise mittels der Metapher des Ratens, zum Beispiel für SAT:

1. Bei Eingabe von Formel φ mit Variablen $x_1, \ldots, x_n$, rate Wahrheitsbelegung B für $x_1, \ldots, x_n$

2. Überprüfe deterministisch und in Polyzeit, ob B die Formel φ erfüllt.

Die NTM akzeptiert die Eingabe, wenn es möglich ist, so zu raten, dass die Berechnung erfolgreich ist (in akzeptierender Stoppkonfiguration endet)

Beachte:

- eine polyzeitbeschränkte NTM kann nur ein polynomiell großes Wort / Objekt raten
- man muss deterministisch in Polyzeit prüfen können, ob das geratene Wort "wie gewünscht" ist.

Bei den Platzkomplexitätsklassen PSpace und NPSpace gibt es keinen derartigen Unterschied:

Theorem 18.10 (Savitch's Theorem)

$\mathsf{PSpace} = \mathsf{NPSpace}$.

Beweis beruht auf einer cleveren Simulation von NTMs mittels DTMs.

Savitch's Theorem hat folgende Konsequenzen:

- Die Komplexitätsklasse NPSpace wird in der Regel nicht explizit betrachtet
- Wenn man sich für die Klasse PSpace interessiert, so darf man o.B.d.A nicht-deterministische TMs verwenden (oft praktisch!)

Betrachten wir nocheinmal

Korollar 18.5

$$\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSpace} \subseteq \mathsf{ExpTime}$$

Mit einem verfeinerten Diagonalisierungsargument kann man zeigen, dass

$$\mathsf{P} \neq \mathsf{ExpTime}$$

Es muss also mindestens eine der Inklusionen in Korollar 18.5 echt sein!

Leider weiß man bis heute nicht mehr darüber.

Insbesondere wird das P vs NP Problem als das wichtigste offene Problem der Informatik angesehen, ob also gilt

$\mathsf{P} = \mathsf{NP}$ (glaubt kaum jemand) oder $\mathsf{P} \neq \mathsf{NP}$ (glauben alle)

§ 19. NP-vollständige Probleme

Wegen $\mathsf{P} \subseteq \mathsf{NP}$ enhält die Klasse NP auch einfach lösbare Probleme, d.h. nicht alle Probleme in NP sind schwer.

Wie kann man die schweren Probleme in NP identifizieren, wenn nicht mal bekannt ist, ob $\mathsf{P} = \mathsf{NP}$?

Man verwendet die sogenannten NP-vollständigen Probleme. Jedes solche Problem L gehört zu den "schwersten Problemen" in NP in folgendem Sinne:

> für jedes Problem $L' \in \mathsf{NP}$ gilt: das Lösen von L' erfordert nur polynomiell mehr Zeitaufwand als das Lösen von L.

Insbesondere folgt dann für jedes NP-vollständige Problem L:
wenn $L \in \mathsf{P}$, dann gilt $\mathsf{P} = \mathsf{NP}$ (was sehr unwahrscheinlich ist!)

Um "polynomiellen Mehraufwand" zu formalisieren, verfeinern wir in geeigneter Weise den Begriff einer Reduktion

Definition 19.1 (Polynomialzeitreduktion, $\leq_p$)

1. Eine Reduktion f von $L \subseteq \Sigma^*$ auf $L' \subseteq \Sigma^*$ heisst Polynomialzeitreduktion, wenn es ein Polynom $p(n)$ und eine $p(n)$-zeitbeschränkte DTM gibt, die f berechnet

2. Wenn es eine Polynomialzeitreduktion von L auf L' gibt, dann schreiben wir $L \leq_p L'$

Die meisten wichtigen Eigenschaften von Reduktionen gelten auch für Polynomialzeitreduktionen, insbesondere:

Lemma 19.2

Wenn $L_1 \leq_p L_2$ und $L_2 \leq_p L_3$, dann auch $L_1 \leq_p L_3$.

Definition 19.3 (NP-hart, NP-vollständig)

1. Problem L heißt NP-hart, falls für alle $L' \in \mathsf{NP}$ gilt: $L' \leq_p L$, d.h. mindestens so schwer zu lösen wie jedes NP-Problem.

2. L heißt NP-vollständig, falls $L \in \mathsf{NP}$ und L NP-hart

Aus folgendem Grund ist "L NP-vollständig" guter Ersatz für "$L \notin \mathsf{P}$"

solange $\mathsf{P} \neq \mathsf{NP}$ nicht bewiesen ist

Lemma 19.4

Wenn L NP-vollständig, dann gilt:

$$L \in \mathsf{P} \quad \text{impliziert} \quad \mathsf{P} = \mathsf{NP}$$

Es ist für NP-vollständige Probleme also höchst unwahrscheinlich, in P zu sein.

Lemma 19.4

Wenn L NP-vollständig, dann gilt:

$$L \in \mathsf{P} \quad \text{impliziert} \quad \mathsf{P} = \mathsf{NP}$$

Beweis: Sei L NP-vollständig und in P; zu zeigen: $\mathsf{NP} \subseteq \mathsf{P}$

Es gibt Polynom $p(n)$ und $p(n)$-zeitbeschränkte DTM $\mathcal{A}$ mit $L(\mathcal{A}) = L$.

Sei $L' \in \mathsf{NP}$. Es gilt $L' \leq_p L$, d.h. es gibt Reduktion f von L' auf L,
die in Zeit $q(n)$ berechenbar ist, $q(n)$ Polynom.

DTM für L' berechnet bei Eingabe w zunächst $f(w)$,
führt dann $\mathcal{A}$ auf $f(w)$ aus.

Zeitaufwand:

- $q(|w|)$ zum Berechnen von $f(w)$; es gilt: $|f(w)| \leq q(|w|) + |w|$
- $p(q(|w|) + |w|)$ für den Lauf von $\mathcal{A}$ auf Eingabe $f(w)$.

Aufwand insgesamt $q(|w|) + p(q(|w|) + |w|)$, also polynomiell.

Satz 19.5 (Cook/Levin)

SAT ist NP-vollständig.

Da $\mathsf{SAT} \in \mathsf{NP}$ bereits gezeigt, bleibt zu beweisen:

SAT ist NP-hart, also: $\forall L \in \mathsf{NP} \Rightarrow L \leq_p \mathsf{SAT}$

Sei $L \in \mathsf{NP}$.

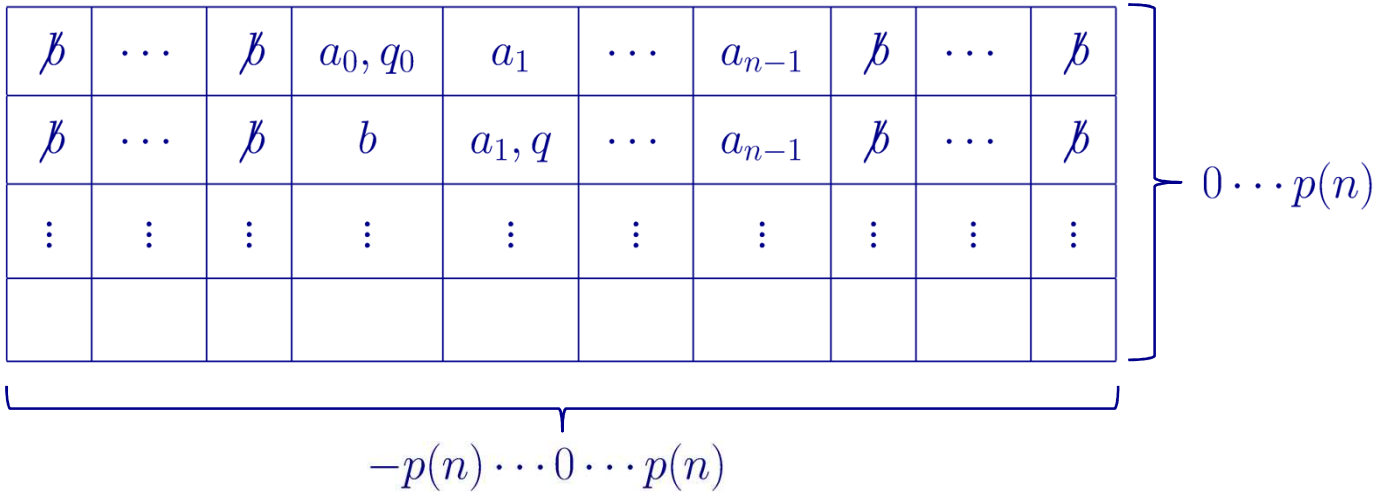
Dann gibt es $p(n)$-zeitbeschränkte NTM $\mathcal{A}$, die L erkennt, mit $p(n)$ Polynom

Mit anderen Worten: L ist das Wortproblem von $\mathcal{A}$

Ziel: gegeben Eingabe w für $\mathcal{A}$, finde AL-Formel φ_w so dass

1. w wird von $\mathcal{A}$ akzeptiert gdw. φ_w ist erfüllbar
2. φ_w in polynomieller Zeit konstruiert werden kann

Berechnungen von $\mathcal{A}$ auf Eingabe $w = a_0 \cdots a_{n-1}$ als Matrix darstellbar:



Die Matrix lässt sich mit polynomiell vielen Aussagenvariablen darstellen:

- $B_{a,i,t}$: zum Zeitpunkt t ist Zelle i mit a beschriftet
- $K_{i,t}$: zum Zeitpunkt t ist der Kopf über Zelle i
- $Z_{q,t}$: zum Zeitpunkt t ist q der aktuelle Zustand

$$0 \leq t \leq p(n)$$
$$-p(n) \leq i \leq p(n)$$
$$a \in \Gamma, q \in Q$$

Idee: φ_w verwendet die Variablen $B_{a,i,t}$, $K_{i,t}$, $Z_{q,t}$,

Belegungen, die φ_w erfüllen $\approx$ akzeptierenden Berechnungen von $\mathcal{A}$ auf w

Berechnung beginnt mit Startkonfiguration für $w = a_0 \cdots a_{n-1}$:

$$\psi_{\text{ini}} := Z_{q_0,0} \wedge K_{0,0} \wedge \bigwedge_{i<n} B_{a_i,i,0} \wedge \bigwedge_{n\le i\le p(n)} B_{\not b,i,0} \wedge \bigwedge_{-p(n)\le i<0} B_{\not b,i,0}.$$

Übergänge am Beispiel von $(p, a, b, r, q), (p, a, b'', \ell, q')$:

$$\bigwedge_{t\le p(n)} \bigwedge_{-p(n)\le i\le p(n)} B_{a,i,t} \wedge K_{i,t} \wedge Z_{p,t} \to \Big(B_{b,i,t+1} \wedge K_{i+1,t+1} \wedge Z_{q,t+1} \vee B_{b',i,t+1} \wedge K_{i-1,t+1} \wedge Z_{q',t+1} \Big)$$

Sei ψ_{move} die Konjunktion aller Übergangsformeln.

Zellen, die nicht unter dem Kopf sind, ändern sich nicht:

$$\psi_{\text{keep}} := \bigwedge_{t<p(n)} \bigwedge_{-p(n)\le i\le p(n)} \bigwedge_{a\in\Gamma} \Big((\neg K_{i,t} \wedge B_{a,i,t}) \to B_{a,i,t+1} \Big)$$

Die Eingabe wird akzeptiert:

$$\psi_{\text{acc}} := \bigwedge_{q_f \in Q \setminus F \text{ Stoppzustand}} \bigwedge_{t \leq p(n)} \neg Z_{q_f,t}$$

Wir nehmen hier wieder o.B.d.A. an, dass stoppen nur vom Zustand, aber nicht vom gelesenen Symbol abhängt.

Bandbeschriftung, Kopfposition, Zustand sind eindeutig und definiert:

$$\psi_{\text{aux}} := \bigwedge_{t,q,q',q \neq q'} \neg(Z_{q,t} \wedge Z_{q',t}) \wedge \bigwedge_{t,i,a,a',a \neq a'} \neg(B_{a,i,t} \wedge B_{a',i,t}) \wedge \bigwedge_{t,i,j,i \neq j} \neg(K_{i,t} \wedge K_{j,t})$$

$$\bigwedge_{t \leq p(n)} \bigvee Q \wedge \bigwedge_{t \leq p(n)} \bigvee_{-p(n) \leq i \leq p(n)} K_{i,t} \wedge \bigwedge_{t \leq p(n)} \bigwedge_{-p(n) \leq i \leq p(n)} \bigvee \Gamma$$

Wir setzen nun

$$\varphi_w := \psi_{\text{ini}} \wedge \psi_{\text{move}} \wedge \psi_{\text{keep}} \wedge \psi_{\text{acc}} \wedge \psi_{\text{aux}}.$$

Zu zeigen: φ_w erfüllbar gdw. $\mathcal{A}$ akzeptiert w.

Im folgenden nur Beweisskizze

"$\Leftarrow$" Sei $k_0 \vdash_{\mathcal{A}} k_1 \vdash_{\mathcal{A}} \cdots \vdash_{\mathcal{A}} k_m$ akzeptierende Berechnung von $\mathcal{A}$ auf w.

Erzeuge daraus Belegung für die Variablen $B_{a,i,t}$, $K_{i,t}$, $Z_{q,t}$, z.B.:

$B_{a,i,t} \mapsto 1$ wenn die i-te Zelle in k_t mit a beschriftet ist.

Man sieht leicht, dass die Belegung alle Konjunkte von φ_w erfüllt.

"$\Rightarrow$" Aus Belegung der Variablen $B_{a,i,t}$, $K_{i,t}$, $Z_{q,t}$, die φ_w erfüllt, liest man Konfigurationsfolge $k_0 \vdash_{\mathcal{A}} k_1 \vdash_{\mathcal{A}} \cdots \vdash_{\mathcal{A}} k_{p(n)}$ ab, z.B.:

die i-te Zelle wird in k_t mit a beschriftet wenn $B_{a,i,t} \mapsto 1$

Da φ_w erfüllt, erhält man akzeptierende Berechnung von $\mathcal{A}$ auf w

Weitere NP-vollständige Probleme erhält man durch polynomielle Reduktion

Satz 19.5

1. Ist $L_2 \in \mathsf{NP}$ und gilt $L_1 \leq_p L_2$, so ist auch L_1 in NP.
2. Ist L_1 NP-hart und gilt $L_1 \leq_p L_2$, so ist auch L_2 NP-hart.

Beweis:

(1) $L_2 \in \mathsf{NP} \quad \Rightarrow \quad$ es gibt polyzeitbeschränkte NTM $\mathcal{A}$, die L_2 erkennt

$L_1 \leq_p L_2 \quad \Rightarrow \quad$ es gibt polyzeitberechenbare Funktion f mit
$w \in L_1$ gdw $f(w) \in L_2$.

Die polyzeitbeschränkte NTM für L_1 arbeitet wie folgt:

- Bei Eingabe w berechnet sie zunächst $f(w)$.
- Dann wendet sie $\mathcal{A}$ auf $f(w)$ an.

(2) Sei L_1 NP-hart und $L_1 \leq_p L_2$; zu zeigen: L_2 NP-hart.

Wähle beliebiges $L \in \mathsf{NP}$. Zu zeigen: $L \leq_p L_2$.

Gesucht: polyzeitberechenbare Reduktion f von L auf L_2, also

$$w \in L \quad \text{gdw} \quad f(w) \in L_2$$

Diese erhält man wie folgt:

- Da L_1 NP-hart ist, gibt es eine Polyzeit-Reduktion g von L auf L_1

$$w \in L \quad \text{gdw} \quad g(w) \in L_1$$

- Wegen $L_1 \leq_p L_2$ gibt es eine Polyzeit-Reduktion h von L_1 auf L_2:

$$u \in L_1 \quad \text{gdw} \quad h(u) \in L_2$$

Dann ist $f(w) := h(g(w))$ wie gewünscht:

$$w \in L \text{ gdw } g(w) \in L_1 \text{ gdw } h(g(w)) \in L_2.$$

Definition 19.6 (3SAT)

- Eine 3-Klausel ist von der Form

$$\ell_1 \vee \ell_2 \vee \ell_3,$$

  wobei ℓ_i ein Literal ist: Variable oder eine negierte Variable.

- Eine 3-Formel ist eine endliche Konjunktion von 3-Klauseln.
- 3SAT ist das folgende Problem:

  Gegeben: eine 3-Formel φ

  Frage: ist φ erfüllbar?

Satz 19.8

3SAT ist NP-vollständig.

Beweis:

(1) 3SAT $\in$ NP folgt unmittelbar aus SAT $\in$ NP,

da jede 3SAT-Instanz eine aussagenlogische Formel ist.

(2) 3SAT ist NP-hart: wir zeigen SAT $\leq_p$ 3SAT

Es sei φ eine beliebige aussagenlogische Formel.

Wir geben ein polynomielles Verfahren an, das φ in eine 3-Formel ψ umwandelt, so dass gilt:

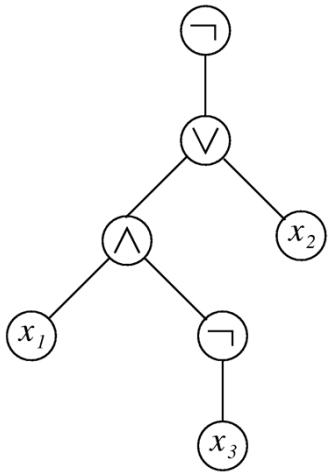
φ erfüllbar gdw ψ erfüllbar.

Die Umformung erfolgt in mehreren Schritten, die wir am Beispiel der Formel

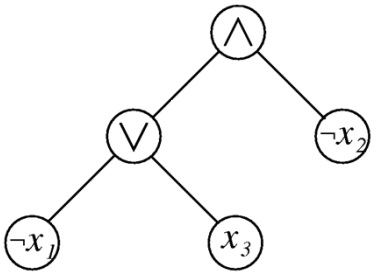
$$\neg((x_1 \wedge \neg x_3) \vee x_2)$$

veranschaulichen.

Wir stellen diese Formel als Baum dar:



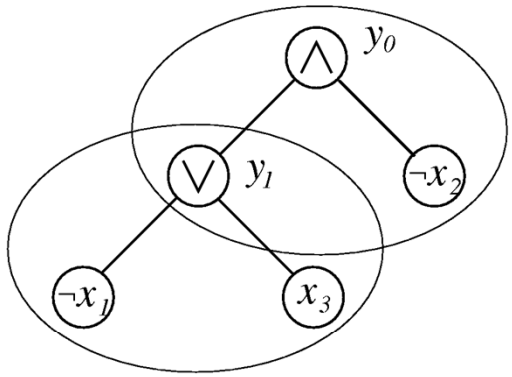
1. Schritt: Wende de Morgan an, um die Negationszeichen zu den Blättern zu schieben.

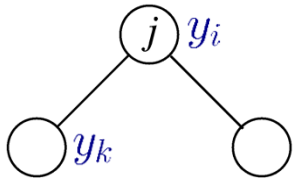


2. Schritt:
Ordne jedem inneren Knoten neue Variable aus $\{y_0, y_1, \ldots\}$ zu, wobei die Wurzel y_0 erhält.

Jede Variable y_i repräsentiert die Teilformel, an Wurzel sie steht

3. Schritt: Fasse jede Verzweigung (gedanklich) zu einer Dreiergruppe zusammen:



Jeder Verzweigung der Form 

mit $j \in \{\wedge, \vee\}$ ordnen wir eine Formel der folgenden Form zu:

$$(y_i \Leftrightarrow (y_k \ j \ y_\ell))$$

Diese Formeln werden nun konjunktiv mit y_0 verknüpft, was die Formel φ_1 liefert.

Im Beispiel ist φ_1 die Formel

$$y_0 \wedge (y_0 \Leftrightarrow (y_1 \wedge \neg x_2)) \wedge (y_1 \Leftrightarrow (\neg x_1 \vee x_3))$$

Behauptung: φ und φ_1 sind erfüllbarkeitsäquivalent.

4. Schritt: Jedes Konjunkt von φ_1 wird separat in Konjunktion von 3-Klauseln umgeformt:

$$
\begin{aligned}
a \Leftrightarrow (b \vee c) \quad &\rightsquigarrow \quad (\neg a \vee (b \vee c)) \wedge (\neg(b \vee c) \vee a) \\
&\rightsquigarrow \quad (\neg a \vee b \vee c) \wedge (\neg b \vee a) \wedge (\neg c \vee a) \\
&\rightsquigarrow \quad (\neg a \vee b \vee c) \wedge (\neg b \vee a \vee a) \wedge (\neg c \vee a \vee a) \\
a \Leftrightarrow (b \wedge c) \quad &\rightsquigarrow \quad (\neg a \vee (b \wedge c)) \wedge (\neg(b \wedge c) \vee a) \\
&\rightsquigarrow \quad (\neg a \vee b) \wedge (\neg a \vee c) \wedge (\neg b \vee \neg c \vee a) \\
&\rightsquigarrow \quad (\neg a \vee b \vee b) \wedge (\neg a \vee c \vee c) \wedge (\neg b \vee \neg c \vee a)
\end{aligned}
$$

Insgesamt erhält man eine 3-Formel, die äquivalent zu φ_1 ist,
und damit wie gewünscht erfüllbarkeitsäquivalent zu φ.

Jeder Schritt kann offensichtlich in polynomieller Zeit durchgeführt werden.

3SAT ist oft sehr nützlich, um mittels Reduktion die NP-Härte anderer Probleme nachzuweisen. Wir betrachten zwei Beispiele.

Zur Erinnerung:

Definition 18.8 (Clique)

Das Problem CLIQUE ist:

Gegeben: Ungerichteter Graph G und $k > 0$

Frage: Enthält G eine Clique mit k Knoten?

Satz 19.11

CLIQUE ist NP-vollständig.

Bereits bewiesen: CLIQUE ist in NP

(2) CLIQUE ist NP-hart: Reduktion von 3SAT auf CLIQUE.

Sei also

$$\varphi = \underbrace{(\ell_{11} \vee \ell_{12} \vee \ell_{13})}_{K_1} \wedge \cdots \wedge \underbrace{(\ell_{m1} \vee \ell_{m2} \vee \ell_{m3})}_{K_m}$$

mit $\ell_{ij} \in \{x_1, \ldots, x_n\} \cup \{\neg x_1, \ldots, \neg x_n\}$.

Der Graph $G = (V, E)$ und die Cliquengröße k werden wie folgt definiert:

- $V := \{\langle i, p\rangle \mid 1 \leq i \leq m \text{ und } 1 \leq p \leq 3\}$
- $E := \{\, \{\langle i, p\rangle, \langle i', p'\rangle\} \mid i \neq i' \text{ und } \ell_{ip} \neq \overline{\ell}_{i'p'}\}$, wobei

$$\overline{\ell} = \begin{cases} \neg x & \text{falls} \quad \ell = x \\ x & \text{falls} \quad \ell = \neg x \end{cases}$$

- $k = m$

Definition 19.8 (Mengenüberdeckung)

Gegeben: Ein Mengensystem über einer endlichen Grundmenge M, d.h.

$$T_1, \ldots, T_k \subseteq M$$

und eine Zahl $n > 0$.

Frage: Gibt es eine Auswahl von n Mengen, die ganz M überdecken, d.h.

$\{i_1, \ldots, i_n\} \subseteq \{1, \ldots, k\}$ mit $T_{i_1} \cup \ldots \cup T_{i_n} = M$.

Satz 19.9

Mengenüberdeckung ist NP-vollständig.

Beweis:

(1) Mengenüberdeckung ist in NP:

- Rate Indizes $i_1, \ldots, i_n$ und
- überprüfe deterministisch in Polyzeit, ob $T_{i_1} \cup \ldots \cup T_{i_n} = M$ gilt.

(2) NP-Härte durch Reduktion von 3SAT auf Mengenüberdeckung:

Sei also $\varphi = K_1 \wedge \ldots \wedge K_m$ 3-Formel mit Variablen $x_1, \ldots, x_n$

Wir definieren $M := \{1, \ldots, m, m+1, \ldots, m+n\}$.

Für jedes $i \in \{1, \ldots, n\}$ sei

$$T_i := \{j \mid x_i \text{ kommt in } K_j \text{ als Disjunkt vor}\} \cup \{m+i\}$$

$$T'_i := \{j \mid \neg x_i \text{ kommt in } K_j \text{ als Disjunkt vor}\} \cup \{m+i\}$$

Wir betrachten das Mengensystem $T_1, \ldots, T_n, T'_1, \ldots, T'_n$ und fragen, ob es eine Überdeckung von M mit n Mengen gibt.

Es gibt eine große Vielzahl von NP-vollständigen Problemen in verschiedensten Teilbereichen der Informatik:

- Graphenprobleme, Probleme auf Mengen, Optimierungsprobleme,
- Schedulingprobleme, Lineare Gleichungssysteme ganzzahlig Lösen,
- etc.

Ein Klassiker der Theoretischen Informatik ist das Buch

„Computers and Intractability: A Guide to the Theory of NP-completeness“
von M. Garey und D. Johnson, 1979

mit ca. 300 NP-vollständigen Problemen; heute sind tausende bekannt.

Zur Erinnerung:

Wegen $\mathsf{NP} \subseteq \mathsf{ExpTime}$ existiert für jedes dieser Probleme ein deterministischer Exponentialzeitalgorithmus

Für keines dieser Probleme ist ein deterministischer Polyzeitalgorithmus bekannt oder dessen Nicht-Existenz mit heutigen Mitteln beweisbar

Einige weitere NP-vollständige Probleme:

- Sudoku.

  Gegeben ein partiell gelöstes Sudoku, kann es zu einem vollständig gelösten erweitert werden?

- Minesweeper.

  Gegeben den momentanen Stand eines Minesweeper Spieles (mit Brett beliebiger Größe), ist an einer bestimmten Stelle mit Sicherheit eine Mine versteckt?

- Bundesliga.

  Gegeben den momentanen Punktestand der Bundesliga (mit beliebig vielen Manschaften), kann eine bestimmte Manschaft noch Meister werden?

Informell: NP-vollständige Probleme haben zwei wichtige Eigenschaften:

- Klein darstellbare und leicht zu verifizierende Belege für Ja-Instanzen.

  Z.B. Sudoku:

  - Restlösung ist Beleg dafür, dass Eingabe (partielle Lösung) Ja-Instanz
  - ist klein: nicht größer als die Eingabe (das Spielfeld)
  - leicht (in Polyzeit) prüfbar, ob Restlösung wirklich Lösung ergibt

- Ein "kombinatorischer" Charakter.

  Z.B. Sudoku:

  Lösung finden durch Ausprobieren, Hypothesen aufstellen, weiter probieren, Entscheidungen gegebenenfalls zurücknehmen (Backtracking), usw.

Allerdings ist die Unterscheidung zwischen Problemen, die „kombinatorisch sind“ und solchen, die nur so aussehen, manchmal nicht ganz leicht:

- 3SAT ist NP-vollständig, 2SAT ist in P
- Bundesliga ist
  - NP-vollständig mit der aktuellen 3-Punkte-Regel
    (Sieg bringt 3 Punkte, Unentschieden 1 Punkt, Niederlage 0 Punkte)
  - in P mit der bis 1994/1995 gültigen 2-Punkte-Regel
    (Sieg bringt 2 Punkte, Unentschieden 1 Punkt, Niederlage 0 Punkte)
- Hamiltonkreis ist NP-vollständig: gibt es in gegebenem ungerichteten Graph einen Kreis, der jeden Knoten genau einmal enthält?
- Eulerkreis ist in P: gibt es in gegebenem ungerichteten Graph einen Kreis, der jede Kante genau einmal enthält?

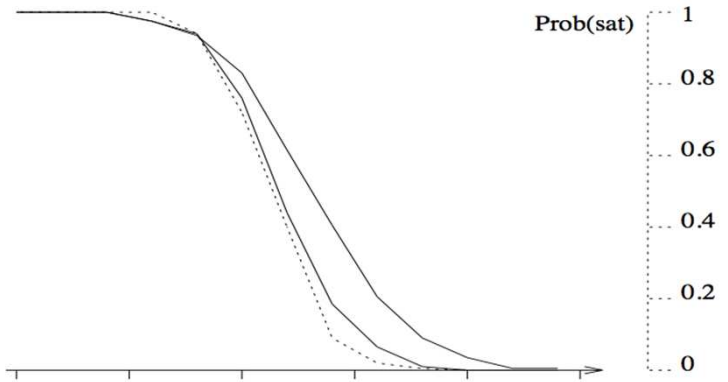
NP-Vollständigkeit bedeutet nicht, dass Probleme in der Praxis unlösbar sind:

Es gibt heute hervorragend SAT-Solver, die SAT-Instanzen mit Millionen von Variablen verarbeiten können.

Der Grund:

- in der Komplexitätstheorie betreiben wir ein worst case Analyse
- in manchen Anwendungen tritt der worst case so gut wie nie auf

Für SAT lässt sich das erklären:
es gibt eine Phasentransition für das Verhältnis von Klauseln zu Variablen



Bei nicht-deterministischen Klassen wie NP ist Vorsichtig mit Komplementen angebracht:

$\mathsf{SAT} \in \mathsf{NP}$:

1. Bei Eingabe von Formel φ mit Variablen $x_1, \ldots, x_n$, rate Wahrheitsbelegung B für $x_1, \ldots, x_n$

2. Überprüfe deterministisch und in Polyzeit, ob B die Formel φ erfüllt.

Wie steht es mit $\overline{\mathsf{SAT}}$, also:

Gegeben:	Aussagenlogische Formel φ
Frage:	Ist φ unerfüllbar?

Es ist nicht klar, was man hier raten sollte, um dann leicht prüfen zu können, dass die Eingabe eine ja-Instanz (also unerfüllbar) ist.

Definiere Komplexitätsklasse $\text{co-NP} := \{\overline{L} \mid L \in \text{NP}\}$

Also: co-NP besteht genau aus den Komplementen der Probleme in NP

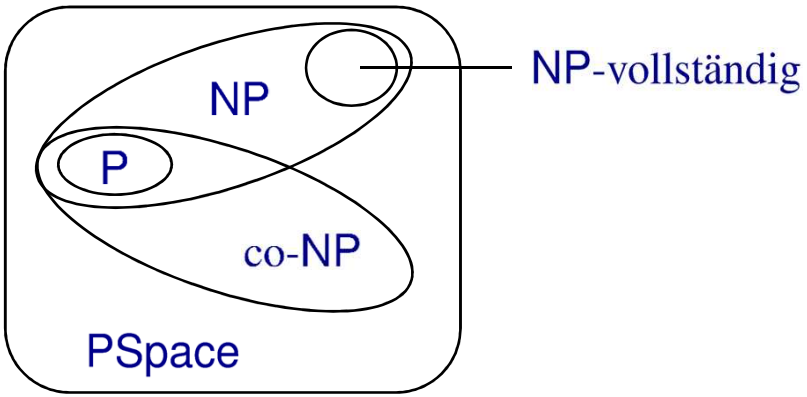
und $\overline{\text{SAT}} \in \text{co-NP}$.

Es wird vermutet, dass $\text{NP} \neq \text{co-NP}$, aber ein Beweis ist nicht bekannt.

Es ist leicht zu sehen, dass $\text{P} \subseteq \text{NP} \cap \text{co-NP}$.

Es wird vermutet, dass $\text{P} \neq \text{NP} \cap \text{co-NP}$, aber ein Beweis ist nicht bekannt.

Gesamtbild also:



Entsprechend zur Definition von NP-vollständig kann man auch den Begriff der PSpace-Vollständigkeit definieren.

Definition (PSpace-hart, PSpace-vollständig)

1. Problem L heißt PSpace-hart, falls für alle $L' \in$ PSpace gilt: $L' \leq_p L$, d.h. mindestens so schwer zu lösen wie jedes PSpace-Problem.

2. L heißt PSpace-vollständig, falls $L \in$ PSpace und L PSpace-hart

Man nimmt an, dass PSpace-vollständige Probleme echt schwieriger sind als NP-vollständige Probleme.

Man kann zeigen, dass folgende Probleme PSpace-vollständig sind:

- das Äquivalenzproblem für NEAs und reguläre Ausdrücke
- das Wortproblem für kontextsensitive Grammatiken

Zusammenfassung Entscheidungsprobleme

	Wortproblem	Leerheitsprob.	Äquivalenzprob.
Typ 0 Grammatik	unentsch.	unentsch.	unentscht.
Typ 1 Grammatik	PSpace-vollständig	unentsch.	unentsch.
Typ 2 Grammatik / PDA	polyzeit	polyzeit	unentsch.
det. PDA	linearzeit	polyzeit	entscheidbar
NEA / reg. Ausdruck / Typ 3 Grammatik	linearzeit	linearzeit	PSpace-vollständig
DEA	linearzeit	linearzeit	polyzeit

Informatik junge Diszplin, entwickelt sich rasant, oft getrieben von Moden

Informatik im Jahr 2100, eine Vorhersage:

Java	$\rightsquigarrow$	Mozambique
Perl	$\rightsquigarrow$	Diamond++## V18-5-21(b)
XML	$\rightsquigarrow$	ZZ-Top
reguläre Sprachen	$\rightsquigarrow$	reguläre Sprachen
berechenbare Funktionen	$\rightsquigarrow$	berechenbare Funktionen
Probleme in P	$\rightsquigarrow$	Probleme in P

Mit theoretischer Informatik seid ihr gut gewappnet!

Vielen Dank für's zuhören!