§ 19. Komplexität

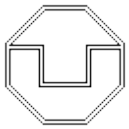
Wir haben mehrere Verfahren betrachtet, die Erfüllbarkeit aussagenlogischer Formeln entscheiden können:

- Aufstellen der Wahrheitstafel
- Tableaux-Verfahren
- Resolutionsverfahren

Für jedes dieser Vefahren haben wir festgestellt, daß es exponentielle Laufzeit benötigen kann, d.h. betrachtet man die benötigte Laufzeit als Funktion in der Größe der Eingabe, so kann diese Funktion nur durch eine Exponentialfunktion (nicht durch ein Polynom) nach oben beschränkt werden.

Beachte:

- Es kann trotzdem manche Eingaben geben, für die das jeweilige Verfahren schneller Antworten liefert.
- Aber es gibt jeweils auch Eingaben für die die exponentielle Schranke erreicht wird: man spricht hier von exponentieller Komplexität im schlimmsten Fall (worst-case complexity).

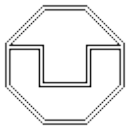


Komplexitätstheorie:

- Betrachtet nicht die Komplexität eines speziellen Entscheidungsverfahrens:
  Wieviel Aufwand benötigt dieses Verfahren?
- Stattdessen wird hier die Komplexität eines Entscheidungsproblems betrachtet:
  Wieviel Aufwand benötigt der „beste“ Algorithmus im „schlimmsten“ Fall?

Speziell für das aussagenlogische Erfüllbarkeitsproblem versucht man hier, die folgende Frage zu beantworten:

- Ist die exponentielle Komplexität tatsächlich eine inhärente Eigenschaft des Erfüllbarkeitsproblems,
- oder sind die bisher entwickelten Verfahren nur nicht gut genug, d.h. es gibt ein polynomielles Verfahren, das man nur noch nicht gefunden hat?



Zur formalen Definition des Begriffs der Komplexität

- betrachten wir wie bei der Definition von Berechenbarkeit/Entscheidbarkeit wieder Turing-Maschinen.
- stellen wir Entscheidungsprobleme als Sprachen über einem endlichen Alphabet dar.

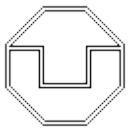
Aussagenlogische Formeln ϕ können z.B. dargestellt werden als

- Wörter $code(\phi)$ über dem Alphabet $\Sigma = \{\wedge, \vee, \neg, \rightarrow, \leftrightarrow, (,), 0, 1\}$,
- wobei für $\mathcal{P} = \{p_0, p_1, p_2, \ldots\}$ die Variable p_i durch die Dualzahldarstellung von i repräsentiert wird.

Das Erfüllbarkeitsproblem für aussagenlogische Formeln
entspricht dann der Sprache

$SAT := \{w \in \Sigma^* \mid w = code(\phi)$ für eine erfüllbare aussagenlogische Formel $\phi\}$.

satisfiability



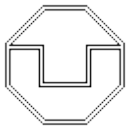
Dresden

Definition 19.1 ($f(n)$-zeitbeschränkt)

Es sei $f : \mathbb{N} \to \mathbb{N}$ eine Funktion.

1. Die DTM $\mathcal{A}$ heißt $f(n)$-zeitbeschränkt, falls
   - für alle Eingaben $w \in \Sigma^*$
   - die Maschine $\mathcal{A}$ bei dieser Eingabe
   - nach $\leq f(|w|)$ Schritten anhält.
2. Die NTM $\mathcal{A}$ heißt $f(n)$-zeitbeschränkt, falls
   - für alle Eingaben $w \in \Sigma^*$
   - jede Berechnung der Maschine $\mathcal{A}$ bei dieser Eingabe
   - nach $\leq f(|w|)$ Schritten anhält.

Es kann $2^{f(|w|)}$ viele Berechnungen geben!



Definition 19.2 (Zeitkomplexitätsklassen)

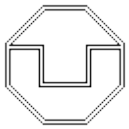
$$DTIME(f(n)) := \{L \mid \text{es gibt eine } f(n)\text{-zeitbeschränkte DTM, die } L \text{ entscheidet}\}$$

$$NTIME(f(n)) := \{L \mid \text{es gibt eine } f(n)\text{-zeitbeschränkte NTM, die } L \text{ akzeptiert}\}$$

Da eine DTM, die L entscheidet, offenbar auch als eine NTM, die L akzeptiert, aufgefaßt werden kann, gilt:

Satz 19.3

$$DTIME(f(n)) \subseteq NTIME(f(n))$$

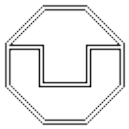


Zwei fundamentale Zeitkomplexitätsklassen:

Definition 19.4 (P, NP)

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

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



Lemma 19.5

$SAT \in NP$

Beweis:

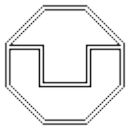
Die NTM, welche die Elemente von SAT mit polynomieller Laufzeit akzeptiert, arbeitet wie folgt:

1. Teste, ob die Eingabe eine aussagenlogische Formel darstellt.

   Dies ist eine Instanz des Wortproblems für kontextfreie Grammatiken, welches in polynomieller Zeit entscheidbar ist (CYC-Algorithmus).

2. Rate eine Wertzuweisung w, d.h. entscheide nicht-deterministisch für jede in ϕ vorkommende Variable p_i, ob $w(p_i) = 1$ oder $w(p_i) = 0$ sein soll (polynomiell viele solche Entscheidungen).

3. Werte ϕ mit dieser Wertzuweisung aus (geht polynomiell) und akzeptiere, falls das Ergebnis 1 ist.



Warum sind die in Definition 19.4 eingeführten Komplexitätsklassen interessant?

Bemerkung 19.6

1. Alle üblichen berechnungsuniversellen Maschinenmodelle lassen sich auf TM in Polynomialzeit simulieren, d.h.:

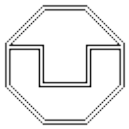
   Benötigt das Maschinenmodell zur Berechnung n Schritte, so benötigt die simulierende TM $q(n)$ Schritte mit q ein Polynom.

   Es folgt, daß die in Definition 19.4 eingeführten Komplexitätsklassen unabhängig vom speziellen Maschinenmodell sind.

2. Probleme, die in P sind, werden im allgemeinen als effizient lösbar („tractable“, d.h. „machbar“) bezeichnet.

   Dies liegt daran, daß Polynome im Vergleich zu Exponentialfunktionen ($n \mapsto 2^n$) ein recht moderates Wachstum haben.

   Aber: In Anwendungen, bei denen man für sehr große Eingaben sehr schnell (in Echtzeit) Antworten haben will, ist ein Polynom mit hohem Grad häufig auch nicht mehr tolerierbar.



Warum sind die in Definition 19.4 eingeführten Komplexitätsklassen interessant?

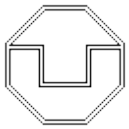
Bemerkung 19.6

3. NP enthält viele Probleme, für die derzeit nur deterministische Entscheidungsverfahren mit exponentiellem Zeitaufwand bekannt sind.

   Man bezeichnet diese Probleme daher häufig als nicht effizient lösbar („intractable“).

   Aber: Die eingeführten Komplexitätsklassen (wie NP) betrachten stets den schwierigsten Fall („worst-case“-Komplexität).

   Es kann durchaus sein, daß es wegen gewisser pathologischer Situationen keine polynomiale Schranke für die Rechenzeit gibt, aber für typische Fälle oder im Mittel doch polynomiales Verhalten erzielt werden kann.



Wie ist das Verhältnis zwischen diesen beiden Komplexitätsklassen?

- Satz 19.3 liefert:

$$P \subseteq NP$$

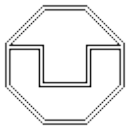
- Ob diese Inklusion echt ist, ist bisher nicht bekannt.

  Die Frage, ob P gleich NP ist, d.h. das

$$P{=}NP\text{-Problem},$$

  ist eines der fundamentalen offenen Probleme der Informatik, dessen Beantwortung weitreichende Konsequenzen hätte:

  - Wäre $P{=}NP$, so könnte man viele in der Praxis wichtige Probleme, für die es bisher nur exponentielle Verfahren gibt, effizient lösen.
  - Wäre $P{=}NP$, so wären viele bisher verwendete kryptographische Verfahren nicht sicher.



Wir zeigen nun, daß SAT „das schwierigste“ NP-Problem ist.

Definition 19.7 (polynomialzeitberechenbar, -reduzierbar, NP-vollständig)

1. Die Funktion $f : \Sigma^* \to \Sigma^*$ heißt polynomialzeitberechenbar, falls es
   - ein Polynom p und
   - eine $p(n)$-zeitbeschränkte DTM gibt,
   - die f berechnet.

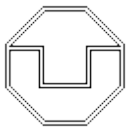
2. $L \subseteq \Sigma^*$ ist polynomialzeitreduzierbar auf $L' \subseteq \Sigma^*$ (geschrieben als $L \leq_p L'$), falls es eine polynomialzeitberechenbare Funktion f gibt mit

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

   für alle $w \in \Sigma^*$.

3. L_0 heißt NP-vollständig, falls gilt:
   - $L_0 \in NP$ (L_0 ist in NP)
   - Für alle $L \in NP$ gilt: $L \leq_p L_0$

     L_0 ist NP-hart, d.h. mindestens so hart zu lösen wie jedes andere NP-Problem.



Satz 19.8 (Cook)

1. Ist L_0 NP-vollständig, so gilt:

$$L_0 \in P \;\Rightarrow\; P = NP$$

2. SAT ist NP-vollständig.

Es ist bisher kein NP-vollständiges L_0 bekannt, das in P ist!

(1) Es sei L_0 NP-vollständig und $L_0 \in P$,

d.h. L_0 wird von einer $p(n)$-zeitbeschränkten DTM (für ein Polynom p) entschieden.

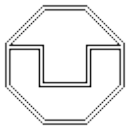
Zu zeigen: daraus folgt $NP \subseteq P$.

Sei also $L \in NP$.

Es ist daher $L \leq_p L_0$,

d.h. es gibt ein f, das mit Zeitaufwand $\leq q(n)$ berechenbar ist (für Polynom q), so daß

$$w \in L \text{ gdw } f(w) \in L_0.$$



Die DTM, welche L entscheidet, geht wie folgt vor:

- Bei Eingabe w berechnet sie $f(w)$.

  Sie benötigt dazu $\leq q(|w|)$ viel Zeit.

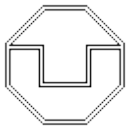
  Daher ist auch die erzeugte Ausgabe $\leq |w| + q(|w|)$.

- Wende Entscheidungsverfahren für L_0 auf $f(w)$ an.

Insgesamt benötigt man somit höchstens die folgende Anzahl von Schritten:

$$\leq q(|w|) + p(|w| + q(|w|))$$

was ein Polynom in $|w|$ ist.



(2) SAT is NP-hart, d.h.

$$\forall L : L \in NP \Rightarrow L \leq_p SAT$$

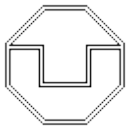
Sei $\mathcal{A}$ eine polynomialzeitbeschränkte NTM, welche $L \subseteq \Sigma^*$ akzeptiert.

Die gesuchte Reduktionsfunktion f weist jedem Wort $w \in \Sigma^*$ eine aussagenlogische Formel β_w zu, so daß

- Der Übergang von w zu β_w ist in Polynomialzeit berechenbar.
- $\mathcal{A}$ akzeptiert w gdw β_w ist erfüllbar.

Die genaue Definition von β_w ist aufwendig.

Es muß ja die Arbeitsweise von $\mathcal{A}$ auf Eingabe w durch eine aussagenlogische Formel beschrieben werden.



Wir beschreiben nur die wesentliche Idee:

- aussagenlogische Variablen $Band_{t,i,a}$ für

  „Zum Zeitpunkt t steht auf Feld i das Symbol a“

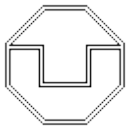
- aussagenlogische Variablen $Kopf_{t,i}$ für

  „Kopf zum Zeitpunkt t auf Feld i“

- $Zustand_{t,q}$

  „Zustand zum Zeitpunkt t ist q“

Da $\mathcal{A}$ polynomialzeitbeschränkt ist, muß man nur
polynomiell viele Bandzellen und Zeitpunkte betrachten!



- Übergänge: (p, a, a', r, q) liefert z.B.:

  $Band_{t,i,a} \wedge Kopf_{t,i} \wedge Zustand_{t,p} \rightarrow Band_{t+1,i,a'} \wedge Kopf_{t+1,i+1} \wedge Zustand_{t+1,q}$

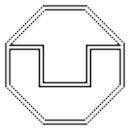
- Kodierung der Eingabe $w = a_1 \ldots a_n$ zum Zeitpunkt 0:

  $$Band_{0,1,a_1} \wedge \ldots \wedge Band_{0,n,a_n} \wedge Band_{0,n+1,\not b} \ldots$$

- Kodierungsbedingungen

  z.B. $\neg(Band_{t,i,a} \wedge Band_{t,i,b})$ für $a \neq b$

- etc.



Weitere NP-vollständige Probleme erhält man durch polynomielle Reduktion.

Satz 19.9

1. Ist $L_2 \in NP$ 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:

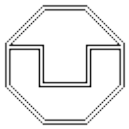
(1)

$L_2 \in NP \quad \Rightarrow \quad$ es gibt polynomialzeitbeschränkte NTM $\mathcal{A}$, die L_2 akzeptiert

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

Die polynomialzeitbeschränkte 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) Es sei L_1 NP-hart und gelte $L_1 \leq_p L_2$.

Zu zeigen: für alle $L \in NP$ gilt $L \leq_p L_2$.

Die polynomialzeitberechenbare Reduktionsfunktion f mit

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

erhält man wie folgt:

- Da L_1 NP-hart ist, gibt es eine polynomialzeitberechenbare Funktion g mit

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

- Wegen $L_1 \leq_p L_2$ gibt es eine polynomialzeitberechenbare Funktion h mit

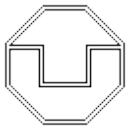
$$u \in L_1 \quad \text{gdw} \quad h(u) \in L_2$$

Wir definieren

$$f(w) := h(g(w)).$$

Dann gilt:

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



Wegen Satz 19.9 kann man die Tatsache, daß SAT NP-vollständig ist, in zweifacher Weise nutzen:

- In der Theorie kann man die NP-Härte eines Problems nachweisen, indem man SAT polynomiell darauf reduziert.

- In der Praxis kann man die Tatsache, daß man jedes Problem in NP polynomiell auf SAT reduzieren kann, dazu verwenden, NP-vollständige Probleme mit Hilfe von SAT-Lösern zu lösen:
  - Es gibt inzwischen hochoptimierte SAT-Löser (SAT solver) wie MiniSAT, Chaff, ..., die in der Praxis auch auf sehr großen Eingabe schnell Antworten liefern.
  - Reduktion of SAT wird z.B. bei der Verifikation (model checking) und bei der automatischen Planung verwendet.