BuK-Script

Mark Morschhäuser

27. März 2008

Inhaltsverzeichnis

Rechnermodelle

Turingmaschine

- Q endliche Zustandsmenge
- Σ Eingabealphabet
- Γ Bandalphabet
- B Blank
- q_0 Startzustand
- $\bar{q}$ Endzustand
- δ Übergangsfunktion (DTM), bzw. -relation (NTM)

Zu beachten ist, dass die Eingabe einer TM beliebig lang sein kann, aber stets endlich ist. Daher kann man die Länge der Eingabe nicht im Zustandsraum speichern.

δ der verschiedenen TM

DTM $\delta : (Q \setminus \{\bar{q}\}) \times \Gamma \to Q \times \Gamma \times \{N, L, R\}$

k-Spur $\delta : (Q \setminus \{\bar{q}\}) \times \Gamma^k \to Q \times \Gamma^k \times \{N, L, R\}$

k-Band $\delta : (Q \setminus \{\bar{q}\}) \times \Gamma^k \to Q \times \Gamma^k \times \{N, L, R\}^k$

NTM $\delta \subseteq ((Q \setminus \{\bar{q}\}) \times \Gamma) \times (Q \times \Gamma \times \{N, L, R\})$

Eselsbrücke: ein beliebiges Programm einer TM (Zustand $\times$ Bandinhalt $\to$ neuer Zustand $\times$ geschriebenes Zeichen $\times$ Kopfbewegung.

Mit ein bissel Nachdenken kommt man dann auf die k-Potenzen bei Mehrband/Mehrspur (bei der k-Spur gibts nur einen Kopf der sich bwegen kann, daher dort keine k-Potenz) und bei der NTM wird aus dem $\to$ ein $\times$ und dem : ein $\subseteq$, weil δ dort eine Relation ist und keine Abbildung.

Charakteristische Funktion der TM M

DTM akzeptiert $\chi_M = \begin{cases} 1 & w \in L \\ 0 & w \notin L \end{cases}$, d.h. Sprache ist entscheidbar (aka rekursiv)

DTM erkennt $\chi_M = \begin{cases} 1 & w \in L \\ \text{undef} & w \notin L \end{cases}$, d.h. Sprache ist semi-entscheidbar (aka rekursiv aufzählbar)

NTM wie DTM, d.h. 1 wenn es einen Lösungsweg gibt, sonst 0.

RAM

CPU-ähnliches Ding, das aus beliebig vielen Registern $c(i)$ mit beliebiger Länge besteht, wobei $c(0)$ der Akkumulator ist, in dem Rechenergebnisse stehen. Die Eingabedaten stehen zu Beginn in den Registern $c(1) \ldots c(k)$, was man angeben sollte.

Zusätzlich gibt es noch einen Befehlszähler b, d.h. jede Codezeile wird nummeriert. Das kann man sich sparen, wenn man Sprungmarken verwendet.

Die Befehle erinnern an Assembler:

- LOAD i
- STORE i
- ADD i
- SUB i
- MULT i
- DIV i (ganzzahlig, abrundend)
- GOTO j oder GOTO Sprungmarke
- IF c(0) $\{=, <, \leq\}$ x GOTO j (wobei x eine Konstante ist, z.B. 0)
- $Sprungmarke
- END bewirkt die Programmtermination. Das Endergebnis sollte in $c(0)$ stehen.

Mit Ausnahme des STORE-Befehls gilt immer:

- Ergebnis steht in $c(0)$, bzw. Rechnung wird mit $c(0)$ ausgeführt (und dort wieder gespeichert).
- In obiger Befehlsauflistung entspricht i dem Registerinhalt des Registers $c(i)$. Will man mit Zahlen arbeiten, verwendet man den jeweiligen C... Befehl, z.B. CLOAD.
- Verwendet man ein Register als Referenz auf ein anderes Register, verwendet man den jeweiligen IND... Befehl: INDFOO auf $c(c(i))$ holt die Zahl aus dem c, auf das $c(i)$ zeigt und wendet es auf Befehl FOO an.

STORE nimmt den Wert aus $c(0)$ und speichert ihn in $c(i)$ oder $c(c(i))$ (INDSTORE). Ein CSTORE-Befehl ergibt daher keinen Sinn und existiert nicht!

Logarithmisches Kostenmaß

Das logarithmische Kostenmaß $L(n)$ einer Zahl n ist nirgends vernünftig oder gar einheitlich definiert, immer nur „das Maß ist abhängig von der Länge der (binär codierten) Eingabe“. Konkrete Rechengesetze für Grundrechenarten o.ä. sucht man vergeblich, vermutlich wegen verschiedener Definitionen der RAM, ka.

Es gilt $L(n) = \begin{cases} 1 & n = 0 \\ \lfloor log(n) \rfloor + 1 & n \geq 1 \end{cases}$

d.h. man codiert eine Zahl größer 0 binär, zählt dann ihre Stellen und erhält so das Kostenmaß der Zahl.

Bei allen RAM-Befehlen wendet man dann für Zahlen und verwendete Register das logarithmische Kostenmaß an, so ist bspw. der Befehl ADD 5 gleich $L(c(0)) + L(c(5))$. Befehle ohne Zahlen oder Register haben Länge 1, z.B. GOTO und END und Befehle mit Konstanten wie CSUB 4 haben $L(c(0)) + L(100_2) = L(c(0)) + 3$.

Durch Nutzung der $\mathcal{O}$-Notation, muss man sich aber eh nur um die aufwendigsten Programmteile kümmern.

Gegenseitige Simulation

1-Band TM simuliert k-Band TM

... mittels 2k Spuren: Auf jeder ungeraden Spur steht ein Band der k-Band-TM und auf der Spur darunter ein Marker, der die Kopfposition der k-Band-TM anzeigt.

Rechenzeit quadratisch: $\mathcal{O}(t^2(n))$, Platzbedarf gleich $\mathcal{O}(s(n))$.

TM simuliert RAM

... mittels zweier Bänder: Für jede Programmzeile, die Initialisierung der TM und für die Ausgabeaufbereitung schreibt man ein Unterprogramm.

Die Register werden binär codiert und mit Trennsymbolen versehen auf einem Band gespeichert. Der Befehlszähler wird im Zustandsraum gespeichert. Das zweite Band wird verwendet, um die im jeweiligen Unterprogramm verwendeten Register zwischenzuspeichern, zu bearbeiten und dann wieder zurückzukopieren.

Initialisierung linear $\mathcal{O}(n)$, Unterprogramme polynomiell vom Band abhängig $\mathcal{O}(n + t(n))$.

RAM simuliert TM

Register 1 speichert Kopfpos, Register 2 Zustände, andere Register die Inhalte der Bandpositionen, sofern sie zur Eingabe gehören oder von der TM besucht wurden. Dabei wird angenommen, dass die Bänder linksseitig beschränkt sind, so wie die Register auch, was eine TM ja nicht weniger mächtg macht.

Simulation in $\mathcal{O}(n + t(n))$ (uniform) oder $\mathcal{O}((n + t(n)) \cdot log(t(n) + n))$ (logarithmisch).

NTM simuliert DTM

klar ;-)

DTM simuliert NTM

Ja, das geht[1]. Man nimmt ohne Einschränkung an, dass die DTM nur 0/1-Verzweigungen macht (das sollte man sich eh merken, dass das geht!) und die DTM probiert jeden dieser Rechenwege durch, z.B. mit Breitensuche.

Wenn eine akzeptierende Berechnung zu einer Eingabe existiert, dann findet die DTM sie auf diese Weise.

Programmiersprachen-Modelle

WHILE

Turingmächtig. Sprachbestandteile:

- konstante Anzahl Variablen $x_0, x_1, \ldots$
- Konstanten -1 0 1
- Symbole ; $:=$ $+$ $\neq$
- Schlüsselwörter WHILE DO END

Syntax (Klausurfrage):

- $x_i := x_j + c$ ist WHILE-Programm für jedes $c \in \{-1, 0, 1\}$
- $P_1; P_2$ ist WHILE-Programm
- WHILE $x_i \neq 0$ DO P END ist WHILE-Programm für jedes WHILE-Programm P

Beachte

- dass man immer nur mit 1 addieren/subtrahieren darf (Multiplikationen/Divisionen sind dadurch auch immer Doppelschleifen!)
- Zuweisungen immer Additionen mit der Konstanten 0 sind
- Subtraktionen immer Additionen mit -1 sind
- dass die Schleife i.d.R. immer auf ungleich 0 prüft → Klausurfrage: „kann man -1 weglassen?“→ Nein, dann kann man die Schleife nicht mehr benutzen!

LOOP

Nicht Turingmächtig (Ackermannfunktion kann nicht berechnet werden). Aufbau wie WHILE, jedoch wird WHILE ersetzt durch LOOP x_i DO P END und x_i darf nicht in P vorkommen. Die Schleife läuft von sich aus x_i mal.

[1] Genauen Algorithmus siehe Asteroth und Baier S. 124.

Ackermannfunktion

$$\begin{aligned} A(0,n) &= n+1 \\ A(m+1,n) &= A(m,1) \\ A(m+1,n+1) &= A(m,A(m+1,n)) \end{aligned}$$

wobei immer gilt, dass $m,n \geq 0$ sind.

Sätze und Definitionen

Church Turing These

Die Klasse der intuitiv berechenbaren Funktionen ist äquivalent zur Klasse der rekursiven Funktionen. Unbeweisbar wegen Formulierung („intuitiv berechenbar“).

Rice

Aussagen über die von einer TM berechneten Funktion sind nicht entscheidbar. Beweis mit Unterprogrammtechnik: $M_{L(S)}$ ist Unterprogramm von M_ε, die H_ε entscheidet.

Cook Levin

SAT ist NPC. Beweis von SAT $\in$ NP mit Zertifikat (gültige Lösung für Formel aus AL) und Verifizierer (gültige Lösung wird in polynomieller Zeit geprüft). Beweis von SAT $\in$ NPC mit Masterreduktion, d.h. alle Sprachen L aus NP werden auf SAT reduziert:

1. Sei $L \subseteq \Sigma^*$ ein beliebiges Prob aus NP. $\forall x \in \Sigma^*$ müssen wir in pol. Zeit eine Formel ϕ konstruieren, so dass
$$x \in L \Leftrightarrow \phi \in SAT$$

2. L wird von einer TM M in einer Laufzeit p erkannt. Konstruiere $\forall x \in \Sigma^*$ eine Formel ϕ mit der Eigenschaft
$$M \text{ akzeptiert } x \Leftrightarrow \phi \text{ ist erfüllbar}$$

3. Stelle M als Formel dar.

   a) Vereinbarungen: o.B.d.A. bewegt sich der Kopf niemals links über die Startposition hinaus und es gibt nur einen akzeptierenden Zustand q_{accept}, bei dessen Erreichen die TM endlos in dem Zustand bleibt.

   b) Für eine beliebige Eingabe $x \in \Sigma^*$ mit $|x| = n$ wird die Konfigurationsfolge $K_0 \vdash K_1 \vdash \cdots \vdash K_{p(n)}$ von M dargestellt, wobei K_0 die Startkonfig ist und $K_{p(n)}$ im Zustand q_{accept} ist.

   c) Definiere die verwendeten Variablen der Formel, abhängig von der Bearbeitungszeit $t \in \{0,\ldots,p(n)\}$:

- $Q(t,k)$ ist der Zustand $k \in Q$
- $H(t,j)$ ist die Kopfposition (Head) $j \in \{0,\ldots,p(n)\}$
- $S(t,j,a)$ ist der Bandinhalt $a \in \Gamma$ unter dem Kopf

Q,H,S sind jeweils wahr (Wert 1), wenn die Rechnung zum Zeitpunkt t durch das jeweilige Q,H,S ausgedrückt wird.

d) Die Kodierung der Konfigurationen wird für jedes t durch eine Formel ϕ_t beschrieben, die nur dann erfüllt ist, wenn die Belegung der Variablenmenge $Q(t,k),H(t,j),S(t,j,a)$ eine korrekte Konfiguration beschreibt. Dazu kodieren wir

- $\exists!$ Zustand k mit $Q(t,k)=1$
- $\exists!$ Bandposition j mit $H(t,j)=1$
- $\forall j$ gibt es jeweils genau ein Zeichen a mit $S(t,j,a)=1$

mit der allgemeinen Formel $(y_1 \vee \cdots \vee y_m) \wedge \bigwedge_{i\neq j}(\neg y_i \vee \neg y_j)$ (d.h. genau ein $y_i = 1$ und alle anderen sind 0).
$\Rightarrow |\phi_t| = \mathcal{O}(p(n)^2)$

e) Die Kodierung der Konfigurationsübergänge entspricht den Formeln $\phi_0 \ldots \phi_{p(n)}$, was allerdings noch ungültige Übergänge beinhalten kann. Konstruiere daher eine Formel ϕ'_t, die nur durch Belegungen erfüllt wird, bei denen K_t eine direkte Nachfolgekonfiguration von K_{t-1} ist, d.h.

i. der Bandinhalt darf nur unter dem Kopf verändert werden…

$$\bigwedge_{i=0}^{p(n)} \bigwedge_{z\in\Gamma} ((S(t-1,i,z) \wedge \neg H(t-1,i)) \Rightarrow S(t,i,z))$$

ii. und aus dem aktuellen Zustand folgt einer der möglichen Folgezustände

$$(Q(t-1,k) \wedge H(t-1,j) \wedge S(t-1,j,a)) \Rightarrow \bigvee_{(k,a,k',a',\kappa)\in\delta} (Q(t,k') \wedge H(t,j+\kappa) \wedge S(t,j,a'))$$

f) Setze die Teilformeln zusammen zu einer Formel der Länge $\mathcal{O}(p(n)^3)$, die genau dann erfüllbar ist, wenn eine akzeptierende Konfigurationsfolge von M auf x existiert:

$$Q(0,q_0) \wedge H(0,0) \wedge \bigwedge_{i=0}^{n} S(0,i,x_i) \wedge \bigwedge_{i=n+1}^{p(n)} S(0,i,B) \wedge \bigwedge_{i=0}^{p(n)} \phi_t \wedge \bigwedge_{i=1}^{p(n)} \phi'_t \wedge Q(p(n),q_{accept})$$

Sprachen

Eine Sprache L ist definiert als $L \subseteq \Sigma^* = \bigcup_{k=0}^{\infty} \Sigma^k$, d.h. eine Sprache ist eine Teilmenge von Wörtern über einem Alphabet (die von einer Grammatik erzeugt werden und sich aus dem Startsymbol ableiten lassen, aber das ist ATFS), denn Σ^k ist die Menge aller Wörter der Länge k über Σ.

- Die Menge der Sprachen ist überabzählbar, die der TM abzählbar.
- Sprachen sind unter Schnitt und Vereinigung abgeschlossen, daher bleiben Rekursivität, Aufzählbarkeit und rekursive Aufzählbarkeit unter $\cup, \cap$ erhalten. Für das Komplement gilt Abgeschlossenheit nur bei rekursiven Sprachen.
- Sind L und $\bar{L}$ rekursiv aufzählbar, so ist L rekursiv.
- Ist L unentscheidbar $\Rightarrow$ $\bar{L}$ unentscheidbar.
- Ist L nicht rekursiv $\Rightarrow$ L oder $\bar{L}$ nicht rekursiv aufzählbar.

In der Auflistung steht (n)r(a) für (nicht) rekursiv (aufzählbar), die roten muss man ganz tot-sicher kennen (die anderen auch).

S_{diag}	$=$	$\{i \in \mathbb{N} \mid A_{i,i} = 1\}$
$\bar{S}_{diag}$	$=$	$\{i \in \mathbb{N} \mid A_{i,i} = 0\}$
$D(nr)$	$=$	$\{w \in \{0,1\}^* \mid w = w_i \text{ und } M_i \text{ akzeptiert } w \text{ nicht}\}$
$\bar{D}(nr,ra)$	$=$	$\{w \in \{0,1\}^* \mid w = w_i \text{ und } M_i \text{ akzeptiert } w\}$
$H(nr,ra)$	$=$	$\{< M > w \mid M \text{ hält auf } w\}$
$H_\varepsilon(nr,ra)$	$=$	$\{< M > \mid M \text{ hält auf } \varepsilon\}$
$\bar{H}_\varepsilon(nra)$	$=$	$\{< M > \mid M \text{ hält nicht auf } \varepsilon\}$
$H_{all}(nra)$	$=$	$\{< M > \mid M \text{ hält auf jeder Eingabe}\}$
$\bar{H}_{all}(nra)$	$=$	$\{< M > \mid M \text{ hält nicht auf jeder Eingabe}\}$
$H_\emptyset$	$=$	$\{< M > \mid M \text{ hält immer und akzeptiert mindestens ein Wort}\}$
H_{Σ^*}	$=$	$\{< M > \mid M \text{ hält immer und verwirft mindestens ein Wort}\}$
$L(S)$	$=$	$\{< M > \mid M \text{ berechnet Funktion aus } S : \emptyset \neq R \subsetneq S\}$
$D'(nr)$	$=$	$\{w \in \{0,1\}^* \mid w = w_i \text{ und } M_i \text{ hält nicht auf } w_i\}$
$A_{all}(nr)$	$=$	$\{< M > \mid M \text{ akzeptiert alle Eingaben}\}$
$A_{EQ}(nr)$	$=$	$\{< M_1 >< M_2 > \mid L(M_1) = L(M_2)\}$
$A(nr)$	$=$	$\{< M > w \mid M \text{ akzeptiert } w\}$
$OR_w(nr)$	$=$	$\{< M_i >< M_j > \mid M_i \text{ oder } M_j \text{ akzeptiert } w \text{ mit } i \in \mathbb{N}\}$
$OR_w^M(r \Leftrightarrow M \text{ akz } w)$	$=$	$\{< M_i > \mid M_i \text{ oder } M \text{ akzeptiert } w \text{ mit } i \in \mathbb{N}\}$
$XOR_w^M(nr)$	$=$	$\{< M_i > \mid \text{Entweder } M_i \text{ oder } M \text{ akzeptiert } w \text{ mit } i \in \mathbb{N}\}$
$H_{\leq 17}(r)$	$=$	$\{< M > \mid M \text{ hält nach max.17 Schritten auf jeder Eingabe}\}$
$LIN-Tape(r)$	$=$	$\{< M > w \mid M \text{ benutzt bei Eingabe } w \text{ nur Bandzellen mit Index } i \text{ für } -\lvert w\rvert \leq i \leq \lvert w\rvert\}$
$POS-Tape(nr)$	$=$	$\{< M > w \mid M \text{ benutzt bei Eingabe } w \text{ nur Bandzellen mit Index } i \geq 0\}$
$PKP(nr,ra)$		
$\overline{PKP}(nra)$		
$Hilbert10(nr,ra)$	$=$	$\{p \mid p \text{ ist Polynom mit Nullstelle aus } \mathbb{Z}\}$
$\overline{Hilbert10}(nr,nra)$	$=$	$\{p \mid p \text{ ist Polynom ohne Nullstelle aus } \mathbb{Z}\}$

Für L(S) ist R die Menge mit den TM-berechenbaren Funktionen.

Eselsbrücke für (nicht) rekursiv aufzählbar: Die TM muss alle Wörter einer Sprache aufs Band schreiben. Wenn sie nur aus Wörtern besteht, die sie nicht akzeptiert, oder immer hält o.ä, dann

ist das nicht mehr möglich, d.h. sie ist nicht rekursiv aufzählbar. Und wenn man schon weiß, dass L rekursiv aufzählbar aber nicht rekursiv ist, dann darf $\bar{L}$ nicht mehr rekursiv aufzählbar sein.

Jede mit einer endlich laufenden TM entscheidbare Sprache ist immer rekursiv, siehe $H_{\leq 17}$.

Semi-Entscheidbarkeit und rekursive Aufzählbarkeit

Es handelt sich um 2 verschiedene Defs, die aber äquivalent sind:

1. Sprache L ist semi-entscheidbar, wenn es eine TM ergibt, die L erkennt
2. L ist rekursiv aufzählbar, wenn es eine TM mit Druckerband gibt (auf dem nur nach rechts gelaufen wird), auf dem jedes Wort der Sprache ausgedruckt wird (egal in welcher Reihenfolge oder Anzahl)

Komplexitätsklassen

P Es gibt eine DTM, die das Problem in polynomieller Zeit entscheidet, d.h. es existiert ein Algorithmus.

NP

1. Es gibt eine NTM, die das Problem entscheidet.
2. Es gibt eine Lösung in Form eines Zertifikats polynomieller Länge, das von einem verifizierenden Polnomialzeitalgorithmus berechnet werden kann.

NP-hart L ist NP-hart wenn $\forall L' \in NP : L' \leq_p L$, d.h. L ist mindestens so schwierig wie alle L', d.h. L' ist mindestens so schwierig wie L und findet man einen Algorithmus für L, dann auch für L'.

schwach NP-hart ist ein NP-hartes Problem, das bei unärer Kodierung einen polynomiellen Algorithmus hat: Das Problem ist nicht leichter lösbar, aber durch den (exponentiellen) Wechsel zur unären Darstellung erscheint das Problem leichter.

stark NP-hart ist ein NP-hartes Problem, das bei unärer Kodierung NP-hart bleibt.

NPC L ist NPC wenn L in NP liegt und L NP-hart ist. Dies sind die schwierigsten Probleme in NP.

PSPACE ist die Klasse der Probleme, die eine TM mit polynomiell beschränktem Band/Platz lösen kann.

EXPTIME ist die Klasse der Probleme mit Laufzeitschranke $2^{p(n)}$ auf einer TM.

Es ist bekannt, dass $P \subsetneq EXPTIME$.

Approximation

FPTAS – Fully Polynomial-Time Approximation Scheme

Eine Klasse von Problemen, die eine Approximation erlauben, so dass für ein beliebiges $\varepsilon > 0$ ein Algorithmus existiert, der garantiert eine Lösung findet, deren Kosten um höchstens $1 + \varepsilon$ von der optimalen Lösung abweichen.

Außerdem ist die Laufzeit des Algorithmusses polynomiell in n (der Größe des Problems) und in $\frac{1}{\varepsilon}$ beschränkt.

Stark NP harte Probleme haben keinen FPTAS, wenn $P \neq NP$.

PTAS – Polynomial-Time Approximation Scheme

Eine Klasse von Problemen, die eine Approximation erlauben, so dass für ein beliebiges $\varepsilon > 0$ ein Polynomialzeitalgorithmus existiert, der garantiert eine Lösung findet, deren Kosten um höchstens $1 + \varepsilon$ von der optimalen Lösung abweichen. Dennoch kann der Exponent des Polynoms stark von ε abhängen.

PTAS beinhaltet FPTAS. Bspw. liegt TSP in der euklidischen Ebene in PTAS.

Diagonalisierung/Reduktionen

Diagonalisierung

Bilde eine Matrix aus Turingmaschinen und Eingabeworten. Auf der Diagonalen steht, ob die jeweilige TM auf dem zugehörigen Wort hält oder nicht (oder akzeptiert, oder sonstwas macht). Nun definieren wir uns eine TM, die sich komplementär zur Diagonalen der Matrix verhält.

Da wir alle TMs auflisten, taucht die komplementäre TM auch irgendwann an einer Stelle in der Matrix auf und bekommt am Diagonaleintrag ein Wort übergeben:

- Ist das Ergebnis auf der Diagonalen wahr, dann terminiert die TM (oder akzeptiert oder whatever), was aber ein Widerspruch zur Definition der komplementären TM ist.
- Ist das Ergebnis auf der Diagonalen nicht wahr, dann terminiert die TM (oder macht whatever) nicht, was aber auch ein Widerspruch ist.
- Alternativ folgert man den Widerspruch $w \in L \Leftrightarrow w \notin L$, was das gleiche ist, aber einen Widerspruch in der Sprache anstatt der TM benutzt.

Der Kniff ist also, einen Algorithmus zu konstruieren (also eine TM), der sich genau komplementär zu den anderen verhält, legt ihn auf die Diagonale und prüft ihn dann nachher selber an seinem Diagonaleintrag um den Widerspruch zu konstruieren.

Unterprogrammtechnik (Turing-Reduktion)

Sei eine Sprache B gegeben, von der gezeigt werden soll, dass sie nicht rekursiv ist und sei A eine für Unterprogrammtechnik günstige (d.h. zu B möglichst ähnliche) Sprache, deren Unentscheidbarkeit bekannt ist.

1. Schreibe: Angenommen, B sei rekursiv $\Rightarrow$ es existiert eine TM, die B entscheidet, d.h. TM für B hält immer und akzeptiert gdw $w \in B$ und verwirft wenn $w \notin B$ (nimm für w die Eingabe aus der Definition von B).

2. Erstelle eine TM (aufschreiben und dann als Skizze) die A entscheidet und in der B ein Unterprogramm ist. Am besten immer mit Syntaxcheck am Anfang, der verwirft oder eine korrekte Eingabe für A ausgibt, die in einer Blackbox verschwindet, die eine korrekte Ausgabe für B ausgibt, die in B geht, welches dann ACC oder REJ ausgibt (ggf. invertiert)

3. Schreibe „Syntaxcheck funktioniert immer korrekt und terminiert immer, klar."

4. Schreibe: TM A führt Eingabe aus (entscheidet also Sprache A) und schreibt genau dann die Eingabe für TM B aufs Band (was der tricky-Part ist, weil die Eingabe irgendwie berechnet werden muss) und führt TM B auf dieser Eingabe aus und übernimmt das Akzeptanzverhalten.

5. Zeige Korrektheit der Konstruktion:

   a) $w \in A \rightarrow$ korrekte Umwandlung/Simulation $\rightarrow$ korrektes Akzeptanzverhalten

   b) $w \notin A \rightarrow$ korrekte Umwandlung/Simulation $\rightarrow$ korrektes Akzeptanzverhalten

6. Zeige korrektes Terminierungsverhalten der Konstruktion (auf Definition der Sprachen/TMs zurückführen).

Reduktion

$L_1 \leq L_2$ bedeutet: Es existiert eine **berechenbare** Funktion f mit $x \in L_1 \Leftrightarrow f(w) \in L_2$

1. Konstruiere eine TM M_1 die L_1 entscheidet [erkennt] und M_2 für L_2.

2. Wandle die auf Korrektheit geprüfte Eingabe von M_1 in Eingabe von M_2 um (d.h. M_1 berechnet $f(w)$ aus ihrer Eingabe w) und zeige, dass dies korrekt ist: $w \in L_1 \Leftrightarrow f(w) \in L_2$, d.h. schreibe „offensichtlich ist f berechenbar und es gilt…" und zeige dann, dass f für $w \in L_1$ und für $w \notin L_1$ korrekt ist und folgere zur Aussage $f(w) \in L_2$, bzw $f(w) \notin L_2$.

Tips: $w \notin L$, wenn der Syntaxcheck verwirft. Wenn die Reduktion schwierig ist, nutze Transitivität aus, falls möglich, d.h. falls $L_1 \leq L_2$ gezeigt werden soll, zeige $L_3 \leq L_2$ wenn wir bereits $L_1 \leq L_3$ hatten.

Es gilt: Wenn L_2 rekursiv [aufzählbar] ist, so auch L_1.

polynomielle Reduktion

Seien A, B Sprachen aus Σ^*.

$A \leq_p B$ bedeutet Reduktion von A nach B , d.h. man will die Eingabe des bekannten/allgemeinen Problems A in eine Eingabe des neuen Problems/Spezialfalles B überführen (A ist nicht schwerer als B).

Verglichen mit der Zeichnung einer Unterprogrammtechnik ist B das Unterprogramm von A. Es ist sehr wichtig, dass man unbedingt mit der Eingabe von A startet und sie dann als Eingabe von B darstellt.

Die Korrektheit der Reduktion zeigt man dann mit

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

wovon man beide Richtungen zeigen muss (A erfüllt $\Rightarrow$ B erfüllt und umgekehrt).

Die Eingabeumwandlung f muss dabei in polynomieller Zeit vorgenommen werden können, was meistens klar ist, aber auch zu zeigen/begründen ist, notfalls mit $\mathcal{O}$-Notation.

Es gilt $B \in P \Rightarrow A \in P$ und A NP-hart $\Rightarrow$ B NP-hart.

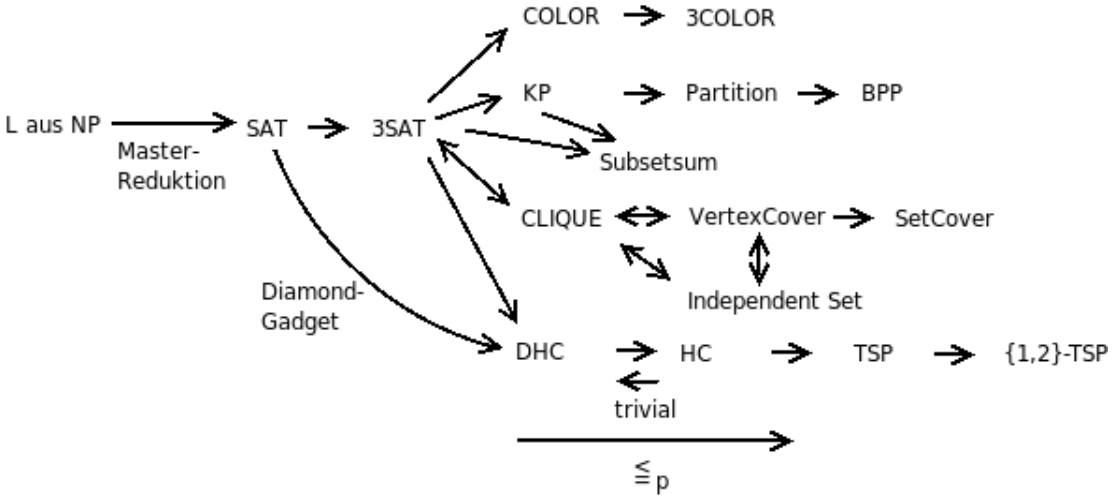
Für einen polynomiell beschränkten Algorithmus A existiere eine Laufzeitschranke von $\mathcal{O}(n^\alpha)$, mit festem α (z.B. 5). Eine Reduktion $B \leq_p A$ benötige $\mathcal{O}(n^\beta)$ mit festem β (z.B. 2). n ist jeweils die Eingabelänge.

Als Laufzeit für B folgert man $\mathcal{O}(n^{\alpha^\beta})$, hier also $n^{5^2} = n^{25}$.

Problemübersicht

Beherrsche (auch in andere Richtungen) die Reduktionen des folgenden Baumes, das sind Reduktionen aus Vorlesung, Übung und Klausur.

Lösungen kann man in der Infolib im Buch *Theoretische Informatik – kurzgefasst* (Spektrum) von Uwe Schöning nachlesen; oder einem anderen Buch aus dem Handapparat.



Definitionen bekannter Probleme

SAT (NPC)

Gegeben Aussagenlogische Formel in KNF.

Frage Existiert eine erfüllende Interpretation der Formel?

3SAT (NPC)

Gegeben Aussagenlogische Formel in 3-KNF.

Frage Existiert eine erfüllende Interpretation der Formel?

2SAT (P)

Gegeben Aussagenlogische Formel in 2-KNF.

Frage Existiert eine erfüllende Interpretation der Formel?

CLIQUE (stark NP hart und NPC)

Gegeben Graph $G = (V,E)$, Zahl $k \in \{1,\ldots,V\}$

Frage Existiert ein k-vollständiger Teilgraph von G?

COLOR (NPC)

Gegeben Graph $G = (V,E)$, Zahl $k \in \{1,\ldots,|V|\}$

Frage Existiert eine k-Färbung von G, d.h. $c : V \to \{1,\ldots,k\}$ so dass $\forall (u,v) \in E : c(u) \neq c(v)$?

KP (schwach NP hart)

Gegeben (obere) Bepackungsgrenze B, (untere) Profitgrenze P, Objekte mit Gewichten $w_1 \ldots w_N \in \{1,\ldots,B\}$ und Profiten p_i.

Frage (Ent) Gibt es eine Bepackung $K \subseteq \{1 \ldots N\}$ mit $\sum_{i\in K} w_i \leq B$ und $\sum_{i\in K} p_i \geq P$?

Frage (Opt) Optimiere $\sum_{i\in K} p_i$

PARTITION (schwach NP hart und NPC)

Gegeben $a_1,\ldots,a_N \in \mathbb{N}$

Frage Gibt es $K \subseteq \{1,\ldots,N\}$ mit $\sum_{i\in K} a_i = \sum_{i\in\{1,\ldots N\}\setminus K} a_i$, bzw. gibt es so ein K mit Summenwert $\frac{1}{2}\sum_{i=1}^{N} a_i$?

SUBSET-SUM (schwach NP hart)

Gegeben $a_1 \dots a_N \in \mathbb{N}, b \in \mathbb{N}$

Frage Gibt es $K \subseteq \{1, \dots N\}$ mit $\sum a_i = b$?

BPP (stark NP hart)

Gegeben $B, k \in \mathbb{N}, w_1, \dots, w_N \in \{1, \dots, B\}$

Frage (Opt) Gibt es eine Funktion $f : \{1, \dots, N\} \to \{1, \dots, k\}$, so dass $\forall i \in \{1, \dots, k\}$ gilt $\sum_{j \in f^{-1}(i)} w_j \leq B$. Minimiere k.

Frage (Ent) Man soll auf höchstens k Behälter verteilen.

DHC

Gegeben Gerichteter Graph G.

Frage Existiert ein HC (Kreis in dem jeder Knoten von G genau einmal vorkommt)?

HC

Gegeben Graph G.

Frage Existiert ein HC (Kreis in dem jeder Knoten von G genau einmal vorkommt)?

TSP (stark NP hart und NPC)

Gegeben Vollständiger, gewichteter Graph $G = (V, E)$, d.h. $c(i, j) \in \mathbb{N}$ mit $i, j \in \{1, \dots, |V|\}$, wobei $c(j, i) = c(i, j)$.

Frage (Opt) Existiert ein HC mit kleinstmöglichen Kosten? Finde eine Permutation π auf $\{1, \dots, |V|\}$, so dass $\sum_{i=1}^{|V|-1} c(\pi(i), \pi(i+1)) + c(\pi(|V|), \pi(1))$ minimal wird.

Frage (Ent) Zusätzlich sei $b \in \mathbb{N}$ gegeben, entscheide ob es einen HC mit höchstens b Kosten gibt.

$\{1,2\}$-TSP-E (NP hart)

Gegeben Vollständiger, gewichteter Graph mit Gewichten 1 oder 2.

SETCOVER (NPC)

Gibt es zu einer Menge X und mehreren Teilmengen $F \subseteq X$ eine Vereinigung von $k \leq b$ F-Mengen, die X ergibt?

Gegeben Menge $X = \{x_1, \ldots, x_n\}$ und Mengen $F_1, \ldots, F_m$ mit $F_i \subseteq X$ für $1 \leq i \leq m$ und eine Zahl $b \in \mathbb{N}$

Frage Gibt es eine Indexmenge I mit $|I| \leq b$, so dass $\bigcup_{i \in I} F_i = X$ gilt?

INDEPENDENT SET (NPC)

Inverses Problem zu CLIQUE: Der Teilgraph besteht nur aus isolierten Ecken, d.h. kein Knoten des Teilgraphen besitzt eine Kante (zu einem anderen Knoten des Teilgraphen).

Gegeben $G = (V, E)$

Frage Für $I \subseteq V$ gilt: $i, j \in I \Rightarrow (i, j) \notin E$

VERTEX COVER (NPC)

Gegeben Graph $G = (V, E), b \in \mathbb{N}$

Frage Existiert $K \subseteq V$ so dass gilt $(i, j) \in E \Rightarrow i \in K$ oder $j \in K$ (d.h. alle Kanten sind inzident zu einem Knoten aus K)

Longest Path (NPC)

Gegeben Graph $G = (V, E), c \in \mathbb{N}$

Frage (Ent) Existiert in G ein kreisfreier Weg der Länge $\geq c$

Bounded Degree MST (NPC)

Gegeben gewichteter Graph G, sowie $c, d \in \mathbb{N}$

Frage (Ent) Existiert ein Spannbaum mit Grad $\leq d$ und Kosten $\leq c$

DNF-SAT (P)

Gegeben φ in DNF ($\bigvee_i \bigwedge_j \neg x_{ij}$).

Frage (Ent) φ erfüllbar?

KNF-VAL (P)

Gegeben φ in KNF.

Frage (Ent) φ Tautologie (d.h. $\neg\varphi$ unerfüllbar)?