Komplexitätstheorie 2

Montag und Donnerstag
14:15 – 15:45 Uhr
in C-221

echte Hierarchien

Aus der Theorie der Formalen Sprachen ist bekannt, dass es Sprachen gibt, die entscheidbar aber nicht kontextsensitiv sind: $Cs \subset Rec$. Die Beweisidee werden wir - modifiziert - auch für Komplexitätsklassen nutzen!

Sei $L_0 := \{w \mid \exists \text{ LBA } M : w = \langle M \rangle \wedge w \notin L(M)\}$

Nun gilt:

1. $L_0 \notin \mathcal{C}s$, Wiederspruchsbeweis:
   Falls dennoch $\exists$ LBA $M_0 : L(M_0) = L_0$ dann gilt doch
   $\langle M_0 \rangle \in L_0 \iff \langle M_0 \rangle \notin L(M_0) = L_0$.
2. $L_0 \in \mathcal{R}ec$, Beweis durch Simulation zeigt sogar
   $L_0 \in n^2(\log n)^2$.

Korollar:
$DSPACE\,(n) = \text{ DLBA } \subseteq \mathcal{C}s \subsetneq DSPACE\left(n^2(\log n)^2\right)$

Hierarchiesatz für $DSPACE$

Satz:
Für platzkonstruierbare $f, g : \mathbb{N} \to \mathbb{N}, g(n) \geq \log(n), f \in o(g)$ gilt:

$$DSPACE\,(f(n)) \subsetneq DSPACE\,(g(n))$$

Beweis:

1. $(\subseteq)$: $f \in o(g) \Longrightarrow \exists n_0 \forall n \geq n_0 : f(n) \leq g(n)$. Sei $L \in DSPACE\,(f)$, dann speichert neue TM Eingaben w mit $|w| \leq n_0$ in endlicher Kontrolle und erkennt alle anderen in $f(|w|) \leq g(|w|)$.

2. $(\neq)$: Zuerst Wahl von geeigneten Kodierungen von DTM'en für effiziente Simulation:
   Wenn x Codierung einer DTM M_x ist so soll auch $y = x\$^n$ für jedes $n \in \mathbb{N}$ die selbe DTM codieren. Jede DTM besitzt so eine beliebig lange Codierung.

   Nun wird g-platzbeschränkte DTM M mit $\overline{L(M)} \notin DSPACE\,(f)$ angegeben, so dass wegen $DSPACE\,(g) = \text{co}-DSPACE\,(g)$ dann $\text{L}(\text{M}) \in DSPACE\,(g) \setminus DSPACE\,(f)$:

Die DTM M

Bei Eingabe von y mit $|y| = n$ markiert M zuerst den Platz $g(n)$ auf Arbeitsband. Sobald M markierten Bereich verlässt, stoppt sie, also $L(M) \in DSPACE\,(g)$. Nun wird von M die durch y codierte DTM M_y mit Eingabe y simuliert. (Falls y keine Codierung ist, lehnt M ab!)
M akzeptiert genau dann, wenn y von M_y in $g(n)$ Platz akzeptiert wird.

Wir zeigen $\overline{L(M)} \notin DSPACE\,(f)$ durch Widerspruch:
Sei M^c eine f-platzbeschränkte DTM mit $L(M^c) = \overline{L(M)} \in DSPACE\,(f)$ und x eine, z.B. die kürzeste, Codierung von M^c. Deren Simulation benötigt bei einer Eingabe der Länge n höchstens $k(f(n)+1)$ viel Platz. Wegen $f \in o(g)$ gibt es $n \geq |x|$ mit $k(f(n)+1) < g(n)$. (k hängt nur von x, bzw. M_x ab.)
Eine Simulation von M^c auf eine Eingabe $y = x\$^{n-|x|}$ der Länge n verlässt den markierten Bereich also nicht.

Damit also: $y \in L(M)$ genau dann, wenn $y \in L(M_y)$ genau dann, wenn $y \in L(M^c)$. Mit $L(M^c) = L(\bar{M})$ folgt somit der Widerspruch

$$y \in L(M) \quad \Longleftrightarrow \quad y \in \overline{L(M)} \quad \Longleftrightarrow \quad y \notin L(M).$$

Folgerungen

Wichtig bei dem Beweis war:
1. Determinismus zur Komplementbildung,
2. das Zählen von Platz kostet nicht mehr Platz als vorhanden ist.

Korollar:

$$DSPACE\left(\log(n)\right) \subsetneq DSPACE\left(\log^2(n)\right) \subsetneq DSPACE\left(n\right) \subseteq \mathcal{C}s \subsetneq$$
$$\subsetneq DSPACE\left(n^2(\log n)^2\right) \subsetneq PSPACE \subsetneq EXPSPACE$$

Diagonalisierungen helfen nur bei
gleichartigen Klassen (nicht für $P \subset PSPACE$, o.ä.)
oder künstlichen Problemen (vergl. $L(M^c)$)

echte Zeithierarchie

Satz:
Für zeitkonstruierbare Funktion g und f mit $f(n) \cdot \log(f(n)) \in o(g(n))$ gilt:

$$DTIME\left(f(n)\right) \subsetneq DTIME\left(g(n)\right)$$

Korollar:

$$DTIME\left(2^n\right) \subsetneq DTIME\left(n^2 \cdot 2^n\right)$$

Es gilt sogar:

Korollar:

$$DTIME\left(2^n\right) \subsetneq DTIME\left(n \cdot 2^n\right)$$

Warum kann man obigen Satz für das zweite Korollar nicht sofort anwenden?

Anwendung von Padding

Mit $f(n) := 2^n$ wird aus einer versuchsweise angenommenen Identität:

$$DTIME\,(2^n) = DTIME\,(n \cdot 2^n)$$

sogleich

$$DTIME\left(2^{2^n}\right) = DTIME\left(2^n \cdot 2^{2^n}\right)$$

und mit $f(n) := n + 2^n$

$$DTIME\left(2^n \cdot 2^{2^n}\right) = DTIME\left((n + 2^n)2^n \cdot 2^{2^n}\right).$$

Beides zusammen ergibt

$$DTIME\left(2^{2^n}\right) = DTIME\left((n + 2^n)2^n \cdot 2^{2^n}\right),$$

was nun aber letztendlich gegen den Zeithierarchiesatz verstößt.

Folgerichtig also:

$$DTIME\,(2^n) \subsetneq DTIME\,(n \cdot 2^n)$$

Vermischtes ohne Beweis

Sei $DUP := \{a^n b^n \mid n \in \mathbb{N}$ dann gilt:

$$DUP \in DSPACE\,(\log n),$$

aber

$$DUP \notin DSPACE\,(f),$$

für $f \in o(\log n)$.

Sei $DMIR := \{w\$w^{rev} \mid w \in \{a,b\}^*\}$ dann gilt:

$$DMIR \in DTIME_1(n^2),$$

aber

$$DMIR \notin DTIME_1(f),$$

für $f \in o(n^2)$.

Satz:
Für $f \in o(n \cdot \log n)$ gilt:

$$DTIME\,(f(n)) = \mathcal{R}eg.$$

Hierarchiesatz für $NTIME$

Satz:
Sei g zeitkonstruierbar und f mit $f(n) \in o(g(n))$ und $f(n+1) \in O(g(n))$, dann gilt:

$$NTIME\,(f(n)) \subsetneq NTIME\,(g(n))$$

Die Voraussetzungen werden erfüllt von $f(n) := 2^n$ und $g(n) := n2^n$ nicht aber von $f(n) := 2^{2^n}$ und $g(n) := 2^n 2^{2^n}$.

Man beachte, dass zwischen f und g in obigem Satz kein log-Faktor besteht!

Für den Beweis siehe:
Seiferas/Fischer/Meyer, Separating nondeterministic time complexity classes, J.A.C.M. 25, (1978) 146-167.

Reduktion und Vollständigkeit

Bei Diagonalisierungen betrachtet man Sprachen des Typs

$$L_0 := \left\{ \langle M_i \rangle \middle| \langle M_i \rangle \notin L(M_i) \right\}$$

Es gilt: $L_0 \notin \mathcal{R}e$.

Betrachtet man

$$L_u := \left\{ \langle M_i, v \rangle \middle| v \in L(M_i) \right\},$$

dann gilt für beliebiges M_i:

$$v \in L(M_i) \iff \langle M_i, v \rangle \in L_u.$$

Die Abbildung $f_{M_i} : v \mapsto \langle M_i, v \rangle$ überführt also das Wortproblem der Sprache $L(M_i)$ in das der universellen Sprache L_u. Dies bezeichnet man als *Reduktion*.

beliebige Reduktionen

Definition:

Sei $L_A \subseteq \Sigma^*$ und $L_B \subseteq \Gamma^*$. Eine Abbildung $f : \Sigma^* \to \Gamma^*$ heißt Reduktion von L_A nach L_B, (notiert als $L_A \leq_f L_B$), wenn für alle $w \in \Sigma^*$ gilt:

$$w \in L_A \quad \Longleftrightarrow \quad f(w) \in L_B \quad \Longleftrightarrow \quad L_A = f^{-1}(L_B).$$

Sei $L \subseteq \Sigma^*$ entscheidbar, d.h. die charakteristische Funktion

$$\chi_L : \Sigma^* \to \{0, 1\}$$

berechenbar. Dann gilt

$$L \leq_\chi \{1\},$$

und χ ist eine Reduktion in der die gesamte Komplexität von L verborgen ist.

Wir werden daher die Reduktionen einschränken und verwenden dazu *k−Band off-line DTM*. (2-Weg Eingabe: nur Lesen, 1-Weg Ausgabe: nur Schreiben, k Arbeitsbänder: 2-Weg Lesen und Schreiben.)

Reduktionstypen

Definition:

Sei $f : \Sigma^* \to \Gamma^*$ eine Reduktion von $L \subseteq \Sigma^*$ nach $M \subseteq \Gamma^*$.

a) f heißt *Polynomzeitreduktion* gdw. es polynomzeitbeschränkte k-*DTM* A gibt, die f berechnet, d.h. dass es ein Polynom p gibt, so dass A für jedes $w \in \Sigma^*$ höchstens $p(|w|)$ Schritte benötigt. Notiert durch $L \leq_{pol} M$.

b) f heißt *LOG-Reduktion* gdw. es logarithmisch Platzbeschränkte k-*DTM* A gibt, die f berechnet, die also für jedes $w \in \Sigma^*$ zur Berechnung von $f(w)$ höchstens $O(\lceil \log(|w|) \rceil)$ viele Bandfelder benutzt. Notiert durch $L \leq_{log} M$.

c) f heißt *LOGLIN-Reduktion* gdw. f eine *LOG-Reduktion* ist und es ein $c > 0$ gibt mit $\forall w \in \Sigma^* : |f(w)| \leq c \cdot |w|$. Notiert durch $L \leq_{loglin} M$.

Beispiele

Beispiel: (Ordnen mit logarithmischem Platz)

$$L := \left\{ \#a_1\#a_2\#\cdots\#a_{n-1}\#a_n \middle| n \in \mathbb{N}, a_i \in 1\{0,1\}^* \right\}$$

$$M := \left\{ \#a_1\#a_2\#\cdots\#a_{n-1}\#a_n \in L \middle| [a_i]_2 \leq [a_{i+1}]_2 \right\}$$

Es gilt $L \leq_{loglin} M$, denn eine ordnende DTM braucht einen Zähler bis n, also $\log n$ Platz, und drei Zeiger in der Eingabe, also insgesamt $\log n$ Platz.

Beispiel: (Universelle Reduktion)

$$L := L(M_i),$$

$$M := \left\{ v\#\langle M_j \rangle \middle| j \in \mathbb{N}, v \in L(M_j) \right\}.$$

Es gilt auch hier $L \leq_{loglin} M$, denn $f(v) = v\#\langle M_i \rangle$ kann sogar in konstantem Platz berechnet werden.

typische, kleine Reduktionen

Beispiel: (*Knapsack* versus *Partition*)

$$\textbf{Knapsack} := \left\{ \#a_1\#a_2\#\cdots\#a_{n-1}\#a_n\#b \middle| \begin{array}{l} n \in \mathbb{N}, a_i, b \in 1\{0,1\}^*, \\ \exists I \subseteq \{1,\ldots,n\} \colon b = \sum\limits_{i \in I} a_i \end{array} \right\},$$

$$\textbf{Partition} := \left\{ \#a_1\#a_2\#\cdots\#a_{n-1}\#a_n \middle| \begin{array}{l} n \in \mathbb{N}, a_i \in 1\{0,1\}^*, \exists I \subseteq \{1,\ldots,n\} \colon \\ \sum\limits_{i \in I} a_i = \sum\limits_{i \notin I} a_i \end{array} \right\}.$$

Es gilt auch hier **Knapsack** $\leq_{loglin}$ **Partition** vermöge

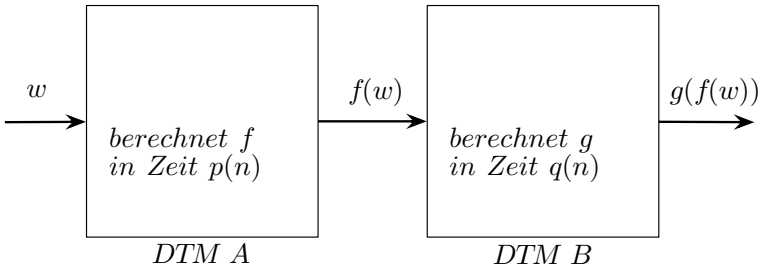
$$(\#a_1\#a_2\#\cdots\#a_{n-1}\#a_n\#b) \mapsto (\#a_1\#a_2\#\cdots\#a_{n-1}\#a_n\#b{+}1\#1{-}b{+}\sum_{i=1}^{n} a_i)$$

Es gilt sogar **Partition** $\leq_{loglin}$ **Knapsack** mit einer ähnlichen Reduktion:

$$(\#a_1\#a_2\#\cdots\#a_{n-1}\#a_n) \mapsto \begin{cases} (\#a_1\#a_2\#\cdots\#a_{n-1}\#a_n\#\frac{1}{2}\sum\limits_{i=1}^{n} a_i), & \sum\limits_{i=1}^{n} a_i \text{ gerade} \\ (\#a_1\#a_2\#\cdots\#a_{n-1}\#a_n\#\frac{1}{2}(1+\sum\limits_{i=1}^{n} a_i)), & \text{sonst.} \end{cases}$$

Transitivität der Reduktionen

Von dem Begriff der NP-Vollständigkeit ist bekannt, dass die Reduktion $\leq_{pol}$ transitiv ist. Folgendes Bild hilft das zu sehen:



Die gesamte verbrauchte Zeit ist $p(|w|) + q(|f(w)|)$ wobei $|f(w)|$ durch $p(|w|)$ beschränkt ist, denn in $p(|w|)$ Schritten kann keine längere Ausgabe erzeugt werden!

techn. Hilfsresultat

Lemma:
Sei $L \subseteq \Sigma^*, M \subseteq \Gamma^*$ und $M \in DTIME\,(t)$ (bzw. $L \in DSPACE\,(s)$), dann gilt:

a) $L \leq_{pol} M$ impliziert $\exists p \in POLY : L \in DTIME\,(\ p(n) + t(p(n))\ )$
$\Big[$bzw. $L \in DSPACE\,(\ p(n) + s(p(n))\ )\Big]$.

b) $L \leq_{log} M$ impliziert $\exists p \in POLY : L \in DTIME\,(\ p(n) + t(p(n))\ )$
$\Big[$bzw. $L \in DSPACE\,(\ \log(n) + s(p(n))\ )\Big]$.

c) $L \leq_{loglin} M$ impliziert $\exists p \in POLY, c > 0 : L \in DTIME\,(\ p(n) + t(c \cdot n))\ )$
$\Big[$bzw. $L \in DSPACE\,(\ \log(n) + s(c \cdot n))\ )\Big]$.

Die (Teil-)Beweise ähneln dem zur polynomiellen Reduktion:

"Die Details"

Beweis:

1. **Zeit**: a) ist klar. Eine logarithmisch platzbeschränkte DTM kann bei Berechnung der Reduktionsabbildung f nur polynomiell viele Schritte machen, so dass auch in den Fällen b) und c) die Länge der Ausgabe $f(w)$ bei der Reduktion durch $p(|w|)$ beschränkt ist. Bei c) zudem noch linear durch $c \cdot n$. Zur Entscheidung von $w \in L$ kommt also nach der Reduktion noch die Zeitkomplexität für die Entscheidung von "$f(w) \in M$"? hinzu.
2. **Platz**: Eine $p(n)$ zeitbeschränkte DTM ist natürlich auch $p(n)$-platzbeschränkt, wodurch die Reduktion in a) mit polynomiellem Platzbedarf auskommt. In b) und c) reicht dafür logarithmischer Platzbedarf. hinzu kommt der Platzbedarf für Entscheidung von $f(w) \in M$.

wichtige Eigenschaften von Reduktionen

Satz:

Es seien $L \subseteq \Sigma^*, M \subseteq \Gamma^*, N \subseteq \Pi^*$ Sprachen, $f : \Sigma^* \to \Gamma^*$ ein Homomorphismus und $p \in POLY^*$.

a) $L \leq_{loglin} M \implies L \leq_{log} M \implies L \leq_{pol} M$,

b) Die Reduktionen $\leq_{loglin}$, $\leq_{log}$ und $\leq_{pol}$ sind transitiv,

c) $f^{-1}(M) \leq_{loglin} M$, und

d) $L \leq_{log} L(p,a)$, wobei $(L(p,a) := \{va^m \mid v \in L, m = p(|v|) - |v|\})$.

so geht's:

Beweis:

a) Ist nach Definition klar.

b) Die Transitivität von $\leq_{pol}$ sahen wir schon. Zu der Transitivität von $\leq_{log}$ ist zu bemerken, dass das Berechnen der gesamten Ausgabe der ersten Reduktion mit $|f(w)| \in O(p(v))$ i.d.R. zuviel Platz benötigt. Daher wird in der zweiten DTM mit einem Zähler in log-Platz festgehalten, welches Symbol der Eingabe gerade verarbeitet wird. Benötigt die zweite Maschine ein weiteres Eingabsymbol, so startet die erste Maschine erneut ihre Berechnung und wird nach entsprechend vielen Schritten gestoppt!

c) zunächst gilt $v \in f^{-1}(M) \leftrightarrow f(v) \in M$ und $|f(v)|$ ist linear beschränkt durch $|v|$. Weiterhin ist $f(v)$ aus v ohne Speicheraufwand berechenbar.

d) Definiere $h(v) := va^{p(|v|)-|v|}$, so dass nun $v \in L \leftrightarrow h(v) \in L(p,a)$. Zur Berechnung von $p(|v|)-|v|$ reicht ein Zähler, der mit $O(\log(p(|v|)) = O(\log(|v|))$ Bits auskommt.

spezielle Beispiele

Beispiele:

a) (Sheila Greibach) $\exists L_0 \in \mathcal{C}f.\forall L \in \mathcal{C}f.\exists f$ *Homomorphismus*: $L = f^{-1}(L_0)$.

b) $\forall L \in PSPACE.\exists p \in POLY : L(p,a) \in DSPACE\,(n) = det\mathcal{C}s$

L_0 nennt man auch *hardest context-free language*. (Warum?)

Abschluss

Definition:

Sei $\mathcal{A}$ eine Familie von Sprachen (Problemen).

a) Der polynomielle Abschluss von $\mathcal{A}$ ist:

$$POL(\mathcal{A}) := \{L \mid \exists M \in \mathcal{A} : L \leq_{pol} M\}.$$

b) Der (logarithmische) Abschluss von $\mathcal{A}$ ist:

$$LOG(\mathcal{A}) := \{L \mid \exists M \in \mathcal{A} : L \leq_{log} M\}.$$

c) Der loglin-Abschluss von $\mathcal{A}$ ist:

$$LL(\mathcal{A}) := \{L \mid \exists M \in \mathcal{A} : L \leq_{loglin} M\}.$$

Härte und Vollständigkeit

Definition:

Sei $\mathcal{A}$ eine Familie von Sprachen (Problemen).
Eine Familie $\mathcal{B}$ heißt $\mathcal{A}$-schwer (-hart) bzgl. POL-Reduktionen (bzw. LOG- oder $LOGLIN$-Reduktionen) genau dann, wenn

$$\mathcal{A} \subseteq POL(\mathcal{B}) \text{ bzw. } \mathcal{A} \subseteq LOG(\mathcal{B}) \quad \mathcal{A} \subseteq LL(\mathcal{B}).$$

Gilt $\mathcal{B} \subseteq \mathcal{A}$ und ist $\mathcal{B}$ schwer bzgl. einer Reduktion, so bezeichnet man die Familie $\mathcal{B}$ als $\mathcal{A}$-*vollständig* (bzgl. der selben Reduktion).

Ist $\mathcal{B} = \{L_0\}$ einelementig, so nennt man die Menge L_0 schwer oder vollständig und lässt beim bilden eines Abschlusses die Mengenklammern um L_0 weg.

Wenn L_0 Greibach's schwerste kontextfreie Sprache ist, was gilt dann:

$$\mathcal{C}f = POL(L_0), \quad \mathcal{C}f = LOG(L_0) \text{ oder } \mathcal{C}f = LL(L_0)?$$

natürliche vollständige Probleme

Definition:

Das Erfüllbarkeitsproblem wird durch die Menge

$$SAT := \{w \in \Sigma^* \mid w \text{ ist ein erfüllbarer boolescher Ausdruck}\}$$

beschrieben.

Stephen A. Cook zeigte 1971, dass obige Menge SAT NP-vollständig ist!

Es ist einfach zu sehen, dass $SAT \in NP$ ist.

Um zu zeigen, dass *jedes* Problem aus NP deterministisch in Polynomzeit auf SAT reduziert werden kann, stellen wir zunächst einmal fest, dass es ausreicht, wenn wir 1-Band TM betrachten. Jede Sprache, die von einer $t(n)$ zeitbeschränkten k-Band TM erkannt wird, kann auch von einer 1-Band TM in $O(t^2(n))$ Zeit erkannt werden. Für NP macht es also keinen Unterschied, ob wir 1-Band oder k-Band Maschinen betrachten, da das Quadrat eines Polynoms wieder ein Polynom ist.

Cook's Konstruktion

Cook fand nun zu jeder polynomzeitbeschränkten 1-Band NTM boolesche Formeln, deren Konjunktion genau dann wahr ist, wenn die TM eine Erfolgsrechnung besitzt. Für jedes Polynom p wird es eine Reduktions-DTM geben, die dies leistet.

Sei $L \in NP$ beliebig und $A_L := (Z, \Sigma, \Gamma, K, q_0, Z_{\text{end}})$ eine 1-Band-NTM mit $L = L(A_L)$, die in $p(n)$-zeitbeschränkung arbeitet ($p \in POLY$).

A_L hat in jeder ihrer Erfolgsrechnungen wegen der Zeitbeschränkung bei Eingabe von w mit $|w| = n$ höchstens $p(n)$ verschiedene Konfigurationen k_i, wobei die Länge $|k_i|$ einer jeden höchstens $p(n) + 1$ ist. Eine Rechnung von A_L lässt sich also eindeutig durch eine Folge von Konfigurationen $k_0\$k_1\$\ldots k_{p(n)}$ beschreiben. In jedem der maximal $p(n)$ Schritte kann sich der LSK von A_L jeweils nur an genau einer von $p(n)$ verschiedenen Positionen befinden. Weiterhin kann sich zu jedem Zeitpunkt A_L in nur genau einem Zustand befinden. Wir können o.B.d.A. annehmen, dass sich bei vorzeitigem Akzeptieren die Konfiguration in allen weiteren Schritten nicht mehr verändert und die aktuelle Bandinschrift stets alle $p(n)$ Felder beschriftet hat, notfalls mit aufgefüllten #.

Die Formelbasis

FELD$(i, j, t) \hat{=}$ In Konfiguration k_t steht das Zeichen x_j in Feld i.

ZUSTAND$(r, t) \hat{=}$ In der Konfiguration k_t befindet sich die TM A_L im Zustand z_r.

KOPF$(i, t) \hat{=}$ In der Konfiguration k_t steht der LSK auf dem Feld i.

Die Anzahl dieser Variablen berechnet sich zu einer Größe der Ordnung $O(p(n)^2)$. Wir definieren die Formel

$$\oplus(a_1, a_2, \ldots, a_r) := (a_1 \vee a_2 \vee \ldots \vee a_r) \bigwedge_{i \neq j} (\neg a_i \vee \neg a_j)$$

mit der Bedeutung: $\oplus(a_1, a_2, \ldots, a_r) = 1$ gdw. genau ein $a_i = 1$. Die Länge dieser Formel ist von der Ordnung $O(r^2)$.

Die Formel F_w mit „$w \in L$“ gdw. „Es gibt eine Erfolgsrechnung für w mit $p(n)$ Schritten“ gdw. „Es gibt eine F_w erfüllende Variablenbelegung“ hat die Form $F_w := A \wedge B \wedge C \wedge D \wedge E \wedge F \wedge G$, in der die einzelnen Teilformeln folgendes bedeuten:

- $A \hat{=}$ In jeder Konfiguration k_t steht der LSK von A_L auf genau einem Feld.
- $B \hat{=}$ In jeder Konfiguration k_t enthält jedes Feld genau ein Zeichen.
- $C \hat{=}$ In jeder Konfiguration k_t befindet sich A_L in genau einem Zustand.
- $D \hat{=}$ Bei jedem Übergang wird genau das Feld verändert, auf das der LSK zeigt.
- $E \hat{=}$ Jeder Übergang im Zustand z_k mit x_j unter dem LSK entspricht der Turing-Tafel.
- $F \hat{=}$ Die erste Konfiguration ist $k_0 = q_0 w \# \ldots \#$.
- $G \hat{=}$ Der Zustand in der letzten Konfiguration $k_{p(n)}$ ist Endzustand aus Z_{end}.

einige Teilbeschreibungen der NTM Konfiguration

Es ist $A := A_1 \wedge A_2 \wedge \ldots \wedge A_{p(n)}$ und $\forall t \leq p(n)$ sei

$$A_t := \oplus(\mathrm{KOPF}(1,t), \mathrm{KOPF}(2,t), \ldots, \mathrm{KOPF}(p(n),t)).$$

Auch B und C sind zusammengesetzte Formeln:

$$B := \bigwedge_{1 \leq i,t \leq p(n)} B(i,t) \text{ mit}$$

$$B(i,t) := \oplus(\mathrm{FELD}(i,1,t), \ldots, \mathrm{FELD}(i,m,t)),\ m := |\Gamma|, \quad \textit{und}$$

$$C := \bigwedge_{1 \leq t \leq p(n)} C_t \text{ mit}$$

$$C_t := \oplus(\mathrm{ZUSTAND}(1,t), \ldots, \mathrm{ZUSTAND}(s,t)),\ s := |Z|.$$

spezielle Erfüllbarkeitsresultate

Satz:

$SAT_{\mathrm{KNF}} := \{w \mid w$ ist eine erfüllbare Boolesche Formel in konjunktiver Normalform$\}$ ist NP-vollständig.

(Die direkte Konstruktion von SAT aus der NTM könnte man schon gleich in KNF erstellen, und käme dann immer noch mit polynomieller Zeit aus! Dies ist in Sipser: Introduction to the Theory of Computation, S. 259, nachzulesen.)

Mit einer Reduktion $SAT <_{pol} SAT_{\mathrm{KNF}}$ zeigt man, dass auch SAT_{KNF} NP-vollständig ist.

Etwas umständlicher zu erklären ist die ebenfalls mögliche Reduktion $<_{log}$. (vergl. Hinweis in: Reischuk, Komplexitätstheorie, vol. I, Teubner (1999)).

Anstelle der Reduktion $SAT <_{pol} SAT_{\mathrm{KNF}}$ wollen wir jedoch gleich den Beweis von $SAT <_{pol} 3{-}SAT$ skizzieren, was wegen $3{-}SAT \subseteq SAT_{\mathrm{KNF}}$ zugleich letztere Menge als NP-vollständig beweist. *Wieso?*

3-SAT

1. In $O(|\alpha|)$ Schritten wird zu einer beliebigen Formel α der Syntaxbaum t_α mit den Operatoren $\wedge, \vee, \neg$ als inneren Knoten erstellt.

2. Mit Hilfe der de Morganschen Regeln wird nun dafür gesorgt, dass alle Negationen nur an den Variablen x_i vorkommen können, die so zu Literalen x_i oder $\overline{x_i}$ werden. Auch diese Kosten sind linear in der Größe von t_α.

3. Im Weiteren werden stets nur Erfüllbarkeitsäquivalente Formeln erstellt:
   Zu jedem inneren Knoten v von t_α mit Beschriftung op $\in \{\wedge, \vee, \neg\}$ und den Kindern v_L und v_R werden drei neue Aussagen-Variablen v, v_1 und v_2 definiert und die Formel $\alpha_v := v \leftrightarrow (v_L \text{ op } v_R)$ zugeordnet. Hierbei ist $v \leftrightarrow (v_L \text{ op } v_R)$ lediglich Abkürzung von

$$\begin{cases} (v \vee \overline{v_L} \vee v_1) \wedge (v \vee \overline{v_L} \vee \overline{v_1}) \wedge (\overline{v} \vee v_L \vee v_R) \wedge \\ \qquad \wedge (v \vee \overline{v_R} \vee v_2) \wedge (v \vee \overline{v_R} \vee v_2), & \text{op} = \vee, \\ (\overline{v} \vee v_L \vee v_1) \wedge (\overline{v} \vee v_L \vee \overline{v_1}) \wedge (\overline{v} \vee v_R \vee v_2) \wedge \\ \qquad \wedge (\overline{v} \vee v_R \vee v_2) \wedge (v \vee \overline{v_L} \vee \overline{v_R}), & \text{op} = \wedge. \end{cases}$$

$SAT <_{\ell og}$ 3-SAT

Es gilt: α ist erfüllbar, gdw. $\left(\bigwedge \alpha_v\right) \wedge \beta$ erfüllbar ist. Hier bei wird die Konjunktion über alle inneren Knoten von t_α gebildet und für den Wurzelknoten v_0 kommt noch die Formel

$$\beta := (v_0 \vee y_1 \vee y_2) \wedge (v_0 \vee \overline{y_1} \vee y_2) \wedge (v_0 \vee y_1 \vee \overline{y_2}) \wedge (v_0 \vee \overline{y_1} \vee \overline{y_2})$$

hinzu, die nur dann erfüllbar ist, wenn v_0 den Wahrheitswert **wahr** bekommt.

Alle verwendeten Formeln besitzen genau drei Literale und die erfüllbarkeitsäquivalente Formel wird in Polynomzeit konstruiert! Genaueres Hinsehen zeigt, dass diese Reduktionskonstruktion sogar in logarithmischem Platz durchgeführt werden kann!

Satz:

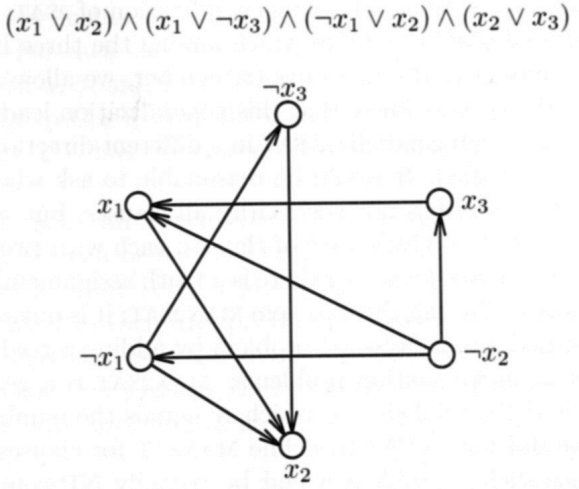
$3{-}SAT := \{w \mid w$ ist eine erfüllbare Boolesche Formel in konjunktiver Normalform mit genau 3 Literalen pro Klausel$\}$ ist NP-vollständig..

2-SAT

Satz:

$2-SAT := \{w \mid w$ ist eine erfüllbare Boolesche Formel in konjunktiver Normalform mit genau 2 Literalen pro Klausel$\}$ ist in $NL \subseteq P$.

Zu einer Formel $\alpha \in 2-SAT$ wird ein gerichteter Graph konstruiert, dessen Knoten die Variablen von α und deren Negationen sind. Zu jeder Klausel $(x \vee y)$ werden zwei Kanten $\overline{x} \longrightarrow y$ und $\overline{y} \longrightarrow x$ gezeichnet.



Lemma:

α ist unerfüllbar genau dann, wenn es eine Variable x gibt und einen Pfad von x nach $\overline{x}$ und von $\overline{x}$ nach x.

2-$SAT \in NL \subseteq P$

Beweis:

Angenommen, es gibt Pfade $x \stackrel{*}{\longrightarrow} \overline{x}$, $\overline{x} \stackrel{*}{\longrightarrow} x$ **und** erfüllende Belegung f. Falls $f(x) = \mathit{true}$ folgt $f(\overline{x}) = \mathit{false}$, und wegen des Pfades $x \stackrel{*}{\longrightarrow} \overline{x}$ muss es Kante $\alpha \longrightarrow \beta$ geben mit $f(\alpha) = \mathit{true}$ und $f(\beta) = \mathit{false}$. Mit dieser Kante gibt es (auch die Kante $\overline{\alpha} \longrightarrow \overline{\beta}$ und) in der Formel eine Klausel $(\overline{\alpha}, \beta)$, die nicht von der Belegung f erfüllt wird! (Fast der gleiche Widerspruch folgte, wenn $f(x) = \mathit{false}$ wäre!)

Falls es aber keine Variable gibt mit solchen Pfaden, so findet man erfüllende Belegung wie folgt: Für beliebige Variable x ohne bisher festgelegten Wahrheitswert setze diesen auf *true* und damit auch alle Literale, die von diesem Knoten aus in G erreichbar sind. $\overline{x}$ und alle Negationen der von x erreichbaren Knoten bekommen natürlich den Wahrheitswert *false*. Das geschieht solange, bis allen Knoten-Literalen Wahrheitswerte zugewiesen sind, und so eine gültige Belegung für die Formel gefunden wurde.

Damit is Unerfüllbarkeit von $2-SAT$ in NL und mit $NL = \text{co}-NL$ auch $2-SAT \in NL$.

NP-vollständige Varianten von SAT

Die Frage, ob es zu gegebener Menge von Klauseln mit genau 2 Literalen und einer Zahl $k \in \mathbb{N}$ eine Belegung gibt, die mindestens k Klauseln erfüllt, ist NP-vollständig.

Die Frage, ob es zu gegebener Menge von Klauseln mit genau 3 Literalen eine erfüllende Belegung gibt, die in keiner der Klauseln alle drei Literale mit dem gleichen Wahrheitswert versieht, ist NP-vollständig.

Jedoch:
Die Frage, ob eine gegebene Menge von Hornklauseln eine erfüllende Belegung besitzt, ist in P.