16 Uniforme versus nicht-uniforme Komplexität

Bei einem uniformen Maschinenmodell wenden wir ein einheitliches Programm auf alle möglichen Eingabewörter über einem gegebenen Alphabet an. Wir haben also nicht die Möglicheit, das Programm an die Länge n des Eingabewortes anzupassen. Software-orientierte Maschinenmodelle (wie zum Beispiel die Turing-Maschinen) sind in der Regel von diesem Typ.

Bei einem nicht-uniformen Maschinenmodell darf die „Maschine“ von der Eingabelänge n abhängig sein. Streng genommen hat man dann eine mit n parametrisierte Familie von Maschinen zur Lösung eines vorgegebenen Problems. Hardware-orientierte Maschinenmodelle (wie zum Beispiel Schaltkreisfamilien) sind in der Regel von diesem Typ.

Wir gehen im Folgenden davon aus, dass das Konzept eines Booleschen Schaltkreises (etwa über der Basis $\{\neg, \vee, \wedge\}$) bekannt ist. Zur Konstruktion eines Schaltkreises wird eine endliche Anzahl von Hardwaregattern azyklisch miteinander verdrahtet. Jedes Gatter berechnet dabei eine elementare Boolsche Funktionen (wie etwa die logische Negation eines Bits bzw. die logische Disjunktion oder Konjunktion zweier Bits). Durch die Verdrahtung zu einem Schaltkreis kann dann im Prinzip jede Boolesche Funktion $f : \{0,1\}^n \to \{0,1\}$ berechnet werden. Im Folgenden bezeichne

$$C := (C_n)_{n \geq 0}$$

eine Schaltkreisfamilie (kurz: SKF), wobei C_n eine Boolesche Funktion in n Variablen berechnet. Diese Funktion notieren wir der Einfachheit halber als $C_n(x)$ mit $x \in \{0,1\}^n$. Wir sagen C *realisiert* die Sprache $L \subseteq \{0,1\}^*$, wenn

$$\forall n \geq 0, \forall x \in \{0,1\}^n : C_n(x) = 1 \Leftrightarrow x \in L \ .$$

Die *Größe* eines Schaltkreises C_n ist die Anzahl seiner Gatter und wird als $size(C_n)$ notiert. $C = (C_n)$ heißt *polynomielle SKF*, wenn $size(C_n)$ polynomiell in n beschränkt ist.

Die Nicht-Uniformität eines Maschinenmodells hat weit reichende Folgen. Zum Beispiel gibt es für jede Sprache L (also auch für nicht-entscheidbare oder gar nicht-semi-entscheidbare) eine SKF, welche L realisiert (einfach darum, weil **jede** Boolesche Funktion realisiert werden kann).[1] Beim Vergleich von SKF's und TM's ist es manchmal hilfreich, nur sogenannte „uniforme“ SKF's zuzulassen. Dabei heißt eine SKF $C = (C_n)$ *uniform*, wenn die Abbildung

$$1^n \mapsto desc(C_n)$$

in Polynomialzeit[2] berechenbar ist, wobei $desc(C_n)$ eine „natürliche“ Kodierung von C_n bezeichnet.[3] Eine uniforme SKF kann nicht superpolynomiell viele Gatter haben und ist daher polynomiell.

Dieses Kapitel ist aufgebaut wie folgt:

[1] Dies wird freilich bedeuten, dass die Funktion $n \mapsto C_n$ i.A. nicht berechenbar ist.

[2] Üblicherweise wird sogar logspace-Berechenbarkeit gefordert. Für unsere Zwecke wird aber die Berechnung von C_n in $\text{pol}(n)$ Schritten ausreichend sein.

[3] „desc“ ist Abkürzung von „description“.

- In den Abschnitten 16.1 und 16.2 zeigen wir, dass jede polynomielle DTM in eine äquivalente uniforme SKF transformiert werden kann und umgekehrt.

- In Abschnitt 16.3 diskutieren wir die Familie $P/poly$ der mit polynomiellen SKF's realisierbaren Sprachen.

- In Abschnitt 16.4 gehen wir erneut auf spärliche (= dünne) Sprachen ein. Es wird sich zeigen, dass $P/poly$ übereinstimmt mit der Klasse der auf spärliche Sprachen Cook-reduzierbaren Sprachen. Außerdem lassen sich die (vermutlich falschen) Aussagen $NP \subseteq P/poly$ bzw. $P = NP$ mit Hilfe von spärlichen und sogenannten kospärlichen Sprachen charakterisieren.

- Im abschließenden Abschnitt 16.5 beweisen wir den Satz von Karp und Lipton, welcher besagt, dass $NP \subseteq P/poly$ zur Folge hätte, dass die polynomielle Hierarchie auf ihren zweiten Level kollabiert.

Hinweis auf laufende Forschung Da vermutlich $NP \not\subseteq P/poly$, ist es naheliegend, nach Sprachen aus NP zu suchen, die sich beweisbar nicht durch polynomielle SKF's berechnen lassen. Wegen $P \subseteq P/poly$ würde dies $P \neq NP$ implizieren. Leider hat sich das Problem, superpolynomielle untere Schranken für (Sprachen aus NP realisierende) SKF's zu beweisen, als außerordentlich hart erwiesen.[4] Aus „Notwehr“ diskutiert man daher hauptsächlich in ihrer Rechenkraft eingeschränkte SKFs (wie zum Beispiel motonone oder tiefenbeschränkte SKFs). Im Rahmen dieser Vorlesung können wir aber darauf nicht näher eingehen.

16.1 Konversion von Software in Hardware

Es sei M eine $S(n)$-platz- und $T(n)$-zeitbeschränkte DTM mit Zustandsmenge Z und Bandalphabet $\Gamma \supset \Sigma$. Zu einer gegebenen Eingabe $x \in \Sigma^n$ stehe M das Bandsegment mit den Zellen $0, 1, \ldots, S(n), S(n)+1$ zur Verfügung, wobei auf den Zellen 0 und $S(n)+1$ Randmarkierungen stehen, die von M nicht überschritten (und auch nicht gelöscht) werden. Der eigentliche Arbeitsbereich besteht aus den Zellen $1, \ldots, S(n)$. Diese Situation ist in Abbildung 15 illustriert. Anfangs befindet sich M im Startzustand z_0, ihr Kopf ist auf Zelle 0

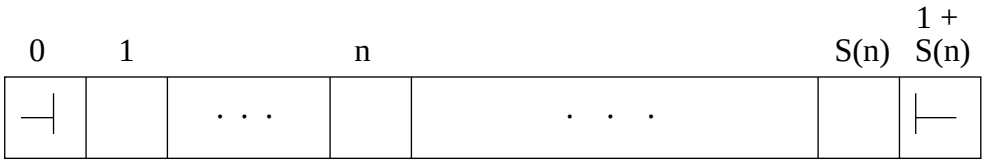


Abbildung 1: Der Arbeitsbereich einer $S(n)$-platzbeschränkten DTM.

positioniert und die Eingabe $x = x_1 \cdots x_n \in \{0,1\}^n$ steht in den Zellen $1, \ldots, n$. Nach exakt

[4] Bisher sind noch nicht einmal superlineare Schranken bekannt.

$T(n)$ Schritten befindet sich M im Zustand z_+, sofern $x \in L_M$, bzw. im Zustand z_-, sofern $x \notin L_M$. Zu diesem Zeitpunkt ist das Band (bis auf die Randmarkierungen) gesäubert (also leer).[5] Jede Konfiguration von M kann in der offensichtlichen Weise durch einen String aus $K^{2+S(n)}$ über dem Zeichenvorrat

$$K := \Gamma \cup (Z \times \Gamma)$$

beschrieben werden. Das Zeichen aus $Z \times \Gamma$ gibt dabei neben dem aktuellen Zustand auch die Kopfposition an.

Beispiel 16.1 *Die Anfangskonfiguration (zu Eingabe x) ist beschrieben durch*

$$(z_0, \dashv)x_1 \cdots x_n \square \cdots \square \vdash$$

und die akzeptierende Endkonfiguration durch

$$(z_+, \dashv)\square \cdots \square \vdash \quad .$$

Die Rechnung von M auf Eingabe x kann durch die *Rechnungstabelle*

$$R := (R_{i,j})_{0 \le i \le T(n), 0 \le j \le S(n)+1}$$

beschrieben werden. Dabei ist $R_{i,j}$ das j-te Zeichen der i-ten Konfiguration.

Zentrale Beobachtung Wegen der lokalen Arbeitsweise von DTMs ist $R_{i+1,j}$ nur von $R_{i,j-1}, R_{i,j}, R_{i,j+1}$ abhängig, d.h.,

$$R_{i+1,j} = f_\delta(R_{i,j-1}, R_{i,j}, R_{i,j+1}) \ ,$$

wobei f_δ eine durch die Überführungsfunktion δ implizit gegebene Funktion ist.[6]

Jedes Zeichen aus K kann auf einfache Weise mit

$$k := \lceil \log |K| \rceil$$

Bits kodiert werden. Es bezeichne

$$R'_{i,j} \in \{0,1\}^k$$

die Kodierung von $R_{i,j}$. Dann gilt

$$R'_{i+1,j} = f'_\delta(R'_{i,j-1}, R'_{i,j}, R'_{i,j+1})$$

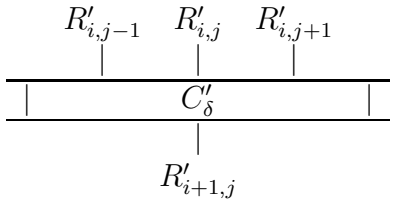
für eine (durch δ implizit gegebene) Boolesche Funktion

$$f'_\delta : \{0,1\}^{3k} \to \{0,1\}^k \ .$$

Es bezeichne C'_δ einen Schaltkreis (konstanter Größe), der f'_δ berechnet:

[5] Durch diese Normierungen haben wir nicht nur eine eindeutige Anfangskonfiguration, sondern auch eine eindeutige (akzeptierende bzw. verwerfende) Endkonfiguration.

[6] Argument $R_{i,j-1}$ entfällt für $j = 0$ (linker Rand); Argument $R_{i,j+1}$ entfällt für $j = S(n) + 1$ (rechter Rand).



Es sollte klar sein, dass durch Parallelschaltung von $2+S(n)$ Duplikaten von C'_δ (mit Besonderheiten am Rand) ein Schaltkreis C_δ entsteht, der aus der i-ten Zeile der Rechnungstabelle R' die $(i+1)$-te Zeile berechnet. Durch Serienschaltung von $T(n)$ Duplikaten von C_δ entsteht ein Schaltkreis C'_M, der die $T(n)$-te Zeile von R' aus der 0-ten Zeile berechnet. Wenn wir einen Schaltkreis C_{in} voranschalten, der aus $x_1 \cdots x_n$ die $k(2+S(n))$ Bits

$$R'_{0,0} \cdots R'_{0,k(S(n)+2)-1}$$

(also die 0-te Zeile von R') berechnet, und einen Schaltkreis C_{out} hinterherschalten, der die Abbildung

$$R'_{T(n),0} \cdots R'_{T(n),k-1} \mapsto \begin{cases} 1 & \text{falls } R_{T(n),0} = (z_+, \vdash) \\ 0 & \text{falls } R_{T(n),0} = (z_-, \vdash) \end{cases}$$

berechnet, so erhalten wir einen Schaltkreis

$$C_M = C_{out} \circ C'_M \circ C_{in} \ ,$$

der L_M realisiert, d.h.,

$$C_M(x) = 1 \Leftrightarrow M \text{ akzeptiert } x \Leftrightarrow x \in L_M \ .$$

C'_M enthält insgesamt

$$(2 + S(n))T(n) = O(S(n)T(n))$$

Duplikate von C'_δ und hat daher die Größe $O(ST) = O(T^2)$. Es ist nicht schwer zu sehen, dass auch der Gesamtschaltkreis C_M die Größe $O(ST) = O(T^2)$ besitzt und in $O(ST) = O(T^2)$ Schritten konstruiert werden kann.

Wir ziehen aus der gesamten Diskussion das folgende Fazit:

Satz 16.2 *Falls $L \in P$, dann kann L durch eine uniforme (und somit auch polynomielle) SKF $C = (C_n)$ realisiert werden.*

16.2 Konversion von Hardware in Software

Ziel dieses Abschnittes ist die Umkehrung von Satz 16.2:

Satz 16.3 *Es sei $C = (C_n)$ eine uniforme SKF und*

$$L_C := \{x \in \{0,1\}^* | \ C_{|x|}(x) = 1\}$$

die von ihr realisierte Sprache. Dann gilt $L_C \in P$.

Beweis Es bezeichne CIRCUIT EVALUATION die Sprache

$$\{\langle desc(C_n), a\rangle |\ n \geq 0, a \in \{0,1\}^n \text{ und } C_n(a) = 1\}\ .$$

Das Mitgliedschaftsproblem zu dieser Sprache besteht im Wesentlichen darin, einen gegebenen Schaltkreis C_n (mit n Boolschen Eingabewerten) an einem gegebenen Booleschen Vektor $a \in \{0,1\}^n$ auszuwerten, was natürlich in Polynomialzeit geschehen kann. Es gilt also

$$\text{CIRCUIT EVALUATION} \in P\ .$$

Der Beweis $L_C \in P$ kann daher geführt werden, indem wir L_C polynomiell auf CIRCUIT EVALUATION reduzieren. Die dazu passende Reduktionsabbildung ist

$$x \mapsto \langle desc(C_{|x|}), x\rangle\ .$$

Sie kann wegen der Uniformität von C in Polynomialzeit berechnet werden und erfüllt (per Definition der beteiligten Sprachen) die Bedingung

$$x \in L_C \Leftrightarrow \langle desc(C_{|x|}), x\rangle \in \text{CIRCUIT EVALUATION}\ .$$

qed.

Aus den Sätzen 16.2 und 16.3 ziehen wir die

Folgerung 16.4 *P ist die Klasse aller durch uniforme SKFs realisierbaren Sprachen.*[7]

16.3 P/poly: die nicht-uniforme „Schwester“ von P

Definition 16.5 *$P/poly$ bezeichne die Klasse aller Sprachen, die durch polynomielle (aber nicht notwendig uniforme) SKFs realisiert werden können.*

Aus Folgerung 16.4 ergibt sich unmittelbar die die Inklusion $P \subseteq P/poly$. Am Ende dieses Abschnittes werden wir nachweisen, dass diese Inklusion echt ist. Die eigentliche (und bis heute ungeklärte) Frage lautet: wie mächtig ist die Klasse $P/poly$? Wir werden in diesem und den beiden folgendenen Abschnitten etwas Licht auf diese Frage werfen und beginnen mit dem

Satz 16.6 *Eine Sprache L gehört zur Klasse $P/poly$ gdw ein Polynom q, eine Sprache $L_0 \in P$ und eine Folge $(a_n)_{n\geq 0}$ von binären Strings $a_n \in \{0,1\}^*$ einer durch $q(n)$ beschränkten Länge existieren, so dass*

$$L = \{x|\ \langle a_{|x|}, x\rangle \in L_0\}\ .$$

[7] Streng genommen müssten wir ein binäres Alphabet $\Sigma = \{0,1\}$ zugrunde legen. Für Sprachen über einem erweiterten Alphabet Σ gilt die Folgerung aber für die davon induzierte Sprache über $\{0,1\}$ (nach Festlegung eines Binärcodes für die Symbole von Σ).

Beweis Wir beweisen zunächst die Richtung von links nach rechts und setzen daher $L \in P/poly$ voraus. Es sei $C = (C_n)$ eine L realisierende polynomielle SKF. Mit

$$a_n := desc(C_n) \text{ und } L_0 := \{\langle desc(C_{|x|}), x\rangle |\ C_{|x|}(x) = 1\}$$

ergibt sich die gewünschte Charakterisierung von L: die Strings $a_= desc(C_n$ haben nämlich eine polynomiel beschränkte Länge, da C eine polynomielle SKF ist und L_0 gehört (als Teilproblem von CIRCUIT EVALUATION) zur Klasse P.
Für die Beweisrichtung von rechts nach links setzen wir jetzt die im Satz beschriebene Charakterisierung von L voraus und weisen $L \in P/poly$ nach. Sei also $q(n) \geq n$ ein Polynom, a_n eine Folge von Strings mit $|a_n| \leq q(n)$, $L_0 \in P$ und $L = \{x|\ \langle a_{|x|}, x\rangle \in L_0\}$. Betrachte eine Boolesche Eingabe $x \in \{0,1\}^n$. Wir können einen Schaltkreis C'_n zur Berechnung von

$$x \mapsto \langle a_n, x\rangle$$

und einen Schaltkreis $C''_{n'}$ mit $n' = |\langle a_n, x\rangle|$ zum Entscheiden von

$$\langle a_n, x\rangle \in L_0?$$

in Serie schalten und erhalten so eine SKF $C = (C_n)$ mit $C_n = C''_{n'} \circ C'_n$, welche L realisiert. Wegen $|a_n| \leq q(n)$ gilt sowohl $|\langle a_n, x\rangle| = O(q(n))$ als auch $size(C'_n) = O(q(n))$. C' ist also eine polynomielle SKF. Zusammen mit $L_0 \in P$ und Satz 16.2 ergibt sich nun, dass C'' und somit auch C eine polynomielle SKF ist. Insgesamt hat sich also $L \in P/poly$ ergeben. **qed.**

Die Bits in a_n heißen auch *Hinweisbits (advice bits)*. Satz 16.6 lässt sich dann auch folgendermaßen lesen: die Sprachen aus $P/poly$ sind dadurch charakterisiert, dass sie von einer DTM in Polynomialzeit akzeptiert werden können, wenn diese neben der eigentlichen Eingabe $x \in \Sigma^n$ polynomiell viele zusätzliche Hinweisbits a_n (die von x nur über $n = |x|$ abhängen dürfen) bekommt. So gesehen beruht der Beweis von Satz 16.6 auf zwei Grundideen:

- Gegeben eine uniforme L realisierende SKF: dann kann $desc(C_n)$ einer (C_n simulierenden) DTM in Form von Hinweisbits zugänglich gemacht werden.
- Gegeben der polynomielle Algorithmus zum Erkennen von Strings aus L mit Hilfe zusätzlicher Hinweisbits a_n: dann können die Bits aus a_n in einen (den Erkennungsalgorithmus realisierenden) Schaltkreis C_n „gehardwired“ werden.

Das Verifizieren der Aussage $x \in L$ mit Hinweisbits (wie bei Sprachen aus $P/poly$) und das Verifizieren von $x \in L$ mit Ratebits (wie bei Sprache aus NP) scheinen verwandt zu sein. Wir machen aber auf zwei wichtige Unterschiede aufmerksam:

NP	*P/poly*
Zertifikat y zum Nachweis von $x \in L$ kann in Abhängigkeit von x gewählt werden.	Hinweisbits a_n zum Nachweis von $x \in L$ hängen nur von $n = \|x\|$ ab (also gleiche Hinweisbits für Eingaben gleicher Länge).
Für $x \notin L$ existiert kein falsches Zertifikat (kein Ratestring y, der zum Akzeptieren von x führt.	Es könnte falsche Hinweise geben (Strings a'_n mit $\langle a'_n, x\rangle \in L_0$ obwohl $x \notin L$ bzw. Strings a'_n mit $\langle a'_n, x\rangle \notin L_0$ obwohl $x \in L$

Wir werden sehen, dass *NP* und *P/poly* sehr unterschiedliche Klassen sind. Zum Beispiel sind wegen $NP \subseteq PSpace$ alle Sprachen in *NP* entscheidbar. *P/poly* hingegen enthält u.a. auch nicht entscheidbare bzw. noch nicht einmal semi-entscheidbare Sprachen, wie aus dem nächsten Resultat hervorgeht:

Satz 16.7 *Jede unäre Sprache $L \subseteq \{0\}^*$ gehört zu P/poly.*

Beweis Das Hinweisbit

$$a_n := \begin{cases} 1 & \text{falls } 0^n \in L \\ 0 & \text{falls } 0^n \notin L \end{cases}$$

reicht aus, um L in Polynomialzeit zu entscheiden: die Dekodierung von a_n aus $\langle a_n, x\rangle$ liefert die Antwort. **qed.**

In der Vorlesung *Theoretische Informatik* wurden zahlreiche nicht entscheidbare (bzw. nicht einmal semi-entscheidbare) Sprachen $L \subseteq \{0,1\}^*$ präsentiert. Zu $x \in \{0,1\}^*$ bezeichne $N(x) \in \mathbb{N}$ die durch die Bitfolge $1x$ binär kodierte natürliche Zahl. Da die Abbildung

$$x \mapsto 0^{N(x)} \text{ (unäre Kodierung von } N(x))$$

berechenbar ist, ist mit L auch die unäre Sprache

$$L' := \{0^{N(x)} |\ x \in L\}$$

nicht (semi-)entscheidbar. Somit gibt es zu jeder nicht (semi-)entscheidbaren Sprache ein unäres Pendant.

Folgerung 16.8

1. *P/poly enthält insbesondere alle unären nicht (semi-)entscheidbaren Sprachen.*
2. *$P \subset P/poly$.*
3. *P/poly ist keine Teilmenge der semi-entscheidbaren Sprachen und kann daher in keiner uniformen (durch eine Platz- oder Zeitschranke gegebenen) Komplexitätsklasse enthalten sein.*

16.4 Spärliche und kospärliche Sprachen

Wir erinnern an Definition 10.7: die *Dichte* einer Sprache L ist gegeben durch

$$dens_L(n) := |\{x \in L : \ |x| \leq n\}|$$

und L heißt *dünn* oder auch *spärlich*, wenn ihre Dichte polynomiell in n beschränkt ist. In diesem Abschnitt benötigen wir weiterhin die

Definition 16.9 *Eine Sprache $L \subseteq \{0,1\}^*$ heißt* kospärlich*, wenn ihr Komplement $\bar{L} = \{0,1\}^* \setminus L$ spärlich ist.*

Der folgende Satz charakterisiert *P/poly* mit Hilfe von spärlichen Sprachen:

Satz 16.10 *P/poly stimmt überein mit der Klasse der auf spärliche Sprachen Cook-reduzierbaren Sprachen.*

Beweis Wir setzen zunächst $L \in P/poly$ voraus und wollen eine spärliche Sprache S mit $L \leq_T S$ ausfindig machen. Gemäß Satz 16.6 existiert ein Polynom q (mit Koeffizienten aus $\mathbb{N}$), Hinweise $(a_n)_{n\geq 0}$ mit $|a_n| \leq q(n)$ und eine Sprache $L_0 \in P$, so dass

$$L = \{x| \ \langle a_{|x|}, x\rangle \in L_0\} \ .$$

Idee Entwerfe S so, dass das S-Orakel nach den Bits von a_n (wobei $n = |x|$) befragt werden kann. Eine DOTM $M[S]$ kann dann $x \in L$ testen wie folgt:

1. Ermittle a_n bitweise durch $q(n)$ entsprechende Anfragen an das S-Orakel.
2. Setze die polynomiell zeitbeschränkte DTM M_0 für die Sprache L_0 auf $\langle a_n, x\rangle$ an und akzeptiere gdw M_0 akzeptiert.

Kommen wir zur Umsetzung der geschilderten Idee und definieren für alle $n \geq 0$ und alle $i = 1, \ldots, q(n)$ die Strings

$$s_i^n := 0^{i-1}10^{q(n)-i} \in \{0,1\}^{q(n)} \ .$$

Die Sprache S sei gegeben durch

$$S := \{s_i^n| \ n \geq 0, 1 \leq i \leq q(n) \text{ und das } i\text{-te Bit von } a_n \text{ ist } 1\} \ .$$

Auf Anfrage s_i^n muss das S-Orakel mit dem i-ten Bit von a_n antworten. S ist spärlich, da offensichtlich[8]

$$|S \cap \{0,1\}^m| \leq m \ .$$

Damit wäre gezeigt, dass jede Sprache aus *P/poly* auf eine spärliche Sprache Cook-reduzierbar ist.
Für die umgekehrte Beweisrichtung setzen wir $L \leq_T S$ voraus, wobei S eine spärliche Sprache bezeichnet, d.h., es gibt ein Polynom $p(n)$, so dass $|S \cap \{0,1\}^n| \leq p(n)$. Wir haben zu zeigen, dass sich das Wortproblem für L mit Hilfe geeigneter (polynomiell längenbeschränkter) Hinweise $(a_n)_{n\geq 0}$ in Polynomialzeit lösen lässt.

[8]Hierbei nutzen wir aus, dass $q(n)$ (als Polynom mit Koeffizienten aus $\mathbb{N}$) mit $n \in \mathbb{N}_0$ streng monoton wächst. Parameter m hat also maximal ein Urbild n mit $m = q(n)$.

Idee Gib eine Liste L_S aller Elemente von S (bis zu einer geeigneten Maximallänge) als Hinweis. Mit Hilfe von L_S kann das S-Orakel simuliert werden.

Kommen wir zur Umsetzung der Idee. Es sei $M'[S]$ die polynomiell zeitbeschränkte DOTM, die für jedes $x \in \Sigma^n$ in $q(n)$ Schritten entscheiden kann, ob $x \in L$. M' kann maximal $q(n)$ viele maximal $q(n)$ lange Anfragen an das S-Orakel stellen. Sei $s_1, \ldots, s_{r(n)}$ eine Auflistung aller Elemente von S der Maximallänge $q(n)$. Wegen der Spärlichkeit von S ist $r(n) \leq p(q(n))$ polynomiell beschränkt. Setze

$$a_n := \langle s_1, \ldots, s_{r(n)} \rangle$$

und beachte, dass

$$|a_n| = O(q(n)p(q(n))) \ .$$

Nun kann (wie geplant) das Wortproblem für L entschieden werden, indem eine DTM M (ohne Orakel), angesetzt auf Eingabe $\langle a_n, x \rangle$ mit $x \in \Sigma^n$, die Rechnung der DOTM M' auf Eingabe x effizient simuliert. Eine etwaige Anfrage an das S-Orakel kann durch Inspektion von a_n beantwortet werden. **qed.**

Wir wissen bereits, dass es vermutlich keine spärliche Sprache gibt, die *NP*-hart (unter polynomieller Reduktion) ist, denn nach dem Satz von Mahaney (s. Satz 10.9) hätte dies $P = NP$ zur Folge. Dies schließt aber noch nicht die Existenz von spärlichen Sprachen aus, die *NP*-hart unter (den mächtigeren) Cook-Reduktionen sind. Das folgende Resultat legt jedoch die Vermutung nahe, dass dies nicht der Fall ist:

Folgerung 16.11 *Es gilt $NP \subseteq P/poly$ gdw es eine spärliche Sprache S gibt, die NP-hart unter Cook-Reduktionen ist.*

Beweis Die Voraussetzung $NP \subseteq P/poly$ impliziert, dass SAT $\in P/poly$, was gemäß Satz 16.10 die Existenz einer spärlichen Sprache S mit SAT $\leq_T S$ zur Folge hat. Folglich ist S *NP*-hart unter Cook-Reduktionen.
Sei umgekehrt vorausgesetzt es gäbe eine spärliche und zugleich unter Cook-Reduktionen *NP*-harte Sprache S. Somit gilt $L \leq_T S$ für jede Sprache $L \in NP$. Gemäß Satz 16.10 folgt hieraus $NP \subseteq P/poly$. **qed.**

$NP \subseteq P/poly$ würde bedeuten, dass alle Sprachen aus *NP* (inklusive der *NP*-vollständigen) durch SKFs polynomieller Größe realisiert werden können. Es wird jedoch vermutet, dass dies nicht der Fall ist. Wir fassen die diversen Vermutungen noch einmal zusammen:

Vermutung 1 Es gibt keine spärliche Sprache, die *NP*-hart (unter polynomiellen Reduktionen) ist. (Ansonsten wäre $P = NP$.)

Vermutung 2 Es gibt keine spärliche Sprache, die *NP*-hart unter Cook-Reduktionen ist. (Ansonsten wäre $NP \subseteq P/poly$.)

Die gleichen Vermutungen werden auch für kospärliche Sprachen gehegt. Dies wird (zumindest für polynomielle Reduktionen) gestützt durch folgenden

Satz 16.12 *Es gilt $P = NP$ gdw es eine kospärliche Sprache S gibt, die NP-hart (unter polynomiellen Reduktionen) ist.*

Beweis Wir setzen zunächst $P = NP$ voraus. Die Menge $S := \{0,1\}^* \setminus \{0\}$ ist trivialerweise kospärlich. Darüberhinaus ist S aber auch NP-hart, da jede Sprache $L \in NP = P$ mit der Reduktionsabbildung

$$x \mapsto \begin{cases} 1 & \text{falls } x \in L \\ 0 & \text{falls } x \notin L \end{cases}$$

polynomiell auf S reduzierbar ist.
Für die umgekehrte Beweisrichtung gehen wir von einer kospärlichen und zugleich NP-harten Sprache S aus und weisen $P = NP$ nach, indem wir einen Polynomialzeitalgorithmus A für SAT entwerfen. Wegen der NP-Härte von S gilt SAT $\leq_{pol} S$ via eine geeignete Reduktionsabbildung $R : \Sigma^* \to \Sigma^*$. Der Algorithmus A für SAT, den wir im Folgenden entwerfen, ist ähnlich gestrickt wie der Algorithmus für SAT, den wir im Beweis des Satzes von Berman (s. Satz 10.10) verwendet haben. Wir „recyclen“ ein paar wesentliche Bestandteile dieses Beweises:

- Zu einer CNF-Formel F über n Booleschen Variablen $v_1, \ldots, v_n$ und einer partiellen Belegung $a \in \{0,1\}^i$ mit $i \in \{0, \ldots, n\}$ assoziieren wir wieder die vereinfachte CNF-Formel $F[a]$, die sich aus F durch Elimination aller bereits erfüllter Klauseln und aller bereits falsifizierter Literale ergibt. Wir erinnern daran, dass $F[\epsilon] = F$ und $F[a] \in \{0,1\}$ für jedes $a \in \{0,1\}^n$ (die Grenzfälle $i = 0$ bzw. $i = n$). Wir erinnern weiterhin daran, dass $F[a]$ erfüllbar ist gdw $F[a0]$ oder $F[a1]$ erfüllbar ist.

- Es bezeichne T den vollständigen binären Baum der Tiefe n, dessen 2^n Blätter in einer 1-zu-1 Beziehung zu den möglichen Belegungen $a \in \{0,1\}^n$ der Booleschen Variablen $v_1, \ldots, v_n$ stehen. Jeder Knoten der Tiefe i entspricht eindeutig einer partiellen Belegung $a \in \{0,1\}^i$. Wir verwenden für diesen Knoten die Notation $u[a]$.

- Ein Exponentialzeitalgorithmus könnte in einem „top-down pass“ den Baum T aufbauen und in einem „bottom-up pass“ die Knoten des Baumes mit Booleschen Wahrheitswerten (0 oder 1) beschriften, so dass der Knoten $u[a]$ mit 1 beschriftet wird gdw die Formel $F[a]$ erfüllbar ist. Damit wäre die ursprünglich gegebene Formel F erfüllbar gdw die Wurzel von T (also der Knoten $u[\epsilon]$) mit 1 beschriftet ist.

- Anstatt nun aber T (in Exponentialzeit) vollständig aufzubauen, versuchen wir die gleiche algorithmische Grundidee auf einem „Anfangsstück“ von T (einer polynomiell in der Kodierungslänge N der Eingabeformel F beschränkten Größe) durchzusetzen. Dabei werden die Konzepte der „Lazy Evaluation“ und des „Hashing“ eingesetzt.

- Zur Implementierung von „lazy evaluation“ erfolgen die Arbeitsschritte während eines Durchlaufes von T in Präordnung. Wenn aber die (zuerst inspizierte) Formel $F[a0]$ sich als erfüllbar erwiesen hat (der Knoten $u[a0]$ ist dann mit 1 markiert), können wir $F[a]$ als erfüllbar deklarieren (also den Knoten $u[a]$ mit 1 markieren), ohne den von $u[a1]$ induzierten Teilbaum aufzubauen (Idee des „Pruning“).

- Als Hashfunktion verwenden wir die Reduktionsabbildung R. Wenn sich eine Formel $F[a]$ als unerfüllbar erwiesen hat, dann sind auch alle Formeln $F[a']$ mit $R(F[a']) = R(F[a])$ unerfüllbar. Wenn daher in T ein Knoten $u[a]$ mit 0 markiert wird, dann können alle Knoten $u[a']$ mit $R(F[a']) = R(F[a])$ (sowie deren Nachfolgerknoten, sofern diese schon aufgebaut wurden) ebenfalls mit 0 markiert werden. Überflüssig zu erwähnen, dass wir den Teilbaum eines bereits mit 0 markierten Knoten nicht aufzubauen brauchen und dass auch keine neuen Knoten $u[a']$ aufgebaut werden, die die Bedingung $R(F[a']) = R(F[a])$ für einen bereits mit 0 markierten Knoten $u[a]$ erfüllen.

Wir verzichten auf eine formale Beschreibung des resultierenden Algorithmus A. Anhand der geschilderten Vorgehensweise sollte klar sein, dass A erstens F akzeptiert (also die Wurzel von T mit 1 markiert) gdw F erfüllbar ist (Korrektheit). Weiterhin sollte klar sein, dass die Laufzeit polynomiell in N beschränkt ist, sofern nur ein polynomiell kleines Anfangsstück T' von T aufgebaut wird. Zu diesem Zweck betrachten wir die „Hashtabelle“

$$H := \{t \in \bar{S} |\ \exists u[a] \in T' :\ u[a] \text{ ist mit 0 markiert und } R(F[a]) = t\}\ .$$

Beachte, dass diese Tabelle wegen der Kospärlichkeit von S eine in $n \leq N$ polynomiell beschränkte Größe hat, sagen wir $|H| \leq q(n)$ für ein Polynom q. Die polynomielle Größenbeschränkung von T' ergibt sich nun leicht aus folgender

Hilfsbehauptung Der Durchlauf in Präordnung habe einen Knoten $u[a] \in T'$ mit $a \in \{0,1\}^i$ und $i \in \{0,\ldots,n-1\}$ erreicht und befinde sich in „top-down“ Richtung. Dann wird während des Aufenthaltes im von $u[a]$ induzierten Unterbaum nach Aufnahme von höchstens $n-i$ neuen Knoten in T' einer der folgenden beiden Fälle eingetreten sein:

Fall 1 $u[a]$ hat eine Boolesche Markierung (0 oder 1) erhalten.

Fall 2 $u[a]$ hat noch keine Boolesche Markierung erhalten, aber die Hashtabelle H hat sich um mindestens 1 vergrößert.

Wir zeigen zunächst, wie unser Beweis mit der Hilfsbehauptung abgeschlossen werden kann. Zu diesem Zweck zerlegen wir den Präordnungsdurchlauf in Arbeits- und Backtracking-Phasen. Zu Beginn einer Arbeitsphase sei der Durchlauf beim Knoten $u[a]$ angelangt (anfangs $u[\epsilon]$) und befinde sich in „top-down“ Richtung. Die Arbeitsphase endet, mit dem Eintreten von Fall 1 bzw. Fall 2. Wenn Fall 1 eintritt, dann belasten wir (beweistechnisch) den Knoten $u[a]$ mit „Kosten“ n. Beachte, dass die in der Arbeitsphase neu in T' aufgenommenen Knoten (auch in Zukunft) nicht mit Kosten belastet werden, da der Durchlauf den von $u[a]$ induzierten Unterbaum nicht mehr betreten wird. Wenn Fall 2 eintritt, nennen wir die während der Arbeitsphase in T' neu aufgenommenen Knoten *H-generiert*. Nach der Arbeitsphase beginnt die (evtl. leere) Backtracking-Phase die solange währt, bis der Durchlauf wieder die „top-down“ Richtung anvisiert.[9] Beachte, dass der Knoten, bei welchem dann die nächste Arbeitsphase beginnt, H-generiert (oder die Wurzel) sein muss. Wegen $|H| \leq q(n)$ kann es

[9]Bechte, dass während des „backtracking“ keine neuen Knoten in T' aufgenommen werden.

höchstens $nq(n)$ H-generierte Knoten in T' geben. Die Anzahl der weiteren Knoten in T' ist nach oben durch die Gesamtkosten der H-generierten Knoten (plus der Wurzel) beschränkt. Diese betragen höchstens $(1+nq(n))n$. Folglich ist die Anzahl der Knoten in T' polynomiell in $n \leq N$ beschränkt.
Der Beweis des Satzes wird nun durch den Beweis der Hilfsbehauptung abgeschlossen. Wir verwenden vollständige Induktion nach $i = n-1, \ldots, 0$. Für $i = n-1$ ist der Durchlauf bei einem Knoten $u[a]$ mit $a \in \{0,1\}^{n-1}$ angelangt und befindet sich in „top-down“ Richtung. Die Knoten $u[a0]$ und $u[a1]$ sind dann Blätter in T. Falls Knoten $u[a0]$ in T' aufgenommen wird, dann gilt entweder $F[a0] = 1$ oder $F[a0] = 0 \wedge R(F[a0]) \notin H$. Im ersten Fall werden $u[a0]$ und $u[a]$ mit 1 markiert und Fall 1 der Hilfsbehauptung ist eingetreten. Im zweiten Fall hat sich die Hashtabelle um ein neues Element vergrößert und Fall 2 der Hilfsbehauptung ist eingetreten. Nehmen wir nun an, dass $u[a0]$ nicht in T' aufgenommen wurde. Dies kann nur daran liegen, dass ein Abgleich mit H ergeben hatte, dass $F[a0]$ nicht erfüllbar ist. In diesem Fall versucht der Präordungsdurchlauf nun sein Glück beim Knoten $u[a1]$. Es gelten nun die analogen Betrachtungen wie zuvor beim Knoten $u[a0]$. Insgesamt hat sich (wie gewünscht) ergeben, dass nach Aufnahme von maximal einem neuen Knoten in T' der Fall 1 oder der Fall 2 der Hilfsbehauptung eingetreten ist. Wir betrachten nun einen Index $i < n-1$ und setzen induktiv voraus, dass die Hilfsbehauptung sich für $i+1$ bereits als richtig erwiesen hat. Der Durchlauf ist also bei einem Knoten $u[a]$ mit $a \in \{0,1\}^i$ angelangt und befindet sich in „top-down“ Richtung. Wenn $u[a0]$ in T' aufgenommen wird, dann gilt per Induktionsvoraussetzung, dass nach Aufnahme von höchstens $n-(i+1)$ weiteren Knoten in T' (für $u[a0]$) Fall 1 oder Fall 2 eingetreten ist. Wenn für $u[a0]$ Fall 2 eintritt (neues Element in H), dann ist Fall 2 somit auch für den Knoten $u[a]$ nach Aufnahme von höchstens $n-i$ Knoten eingetreten. Wenn für $u[a0]$ Fall 1 eingetreten ist, dann unterscheiden wir zwei Unterfälle. Hat $u[a0]$ die Boolesche Markierung 1, dann wird auch $u[a]$ mit 1 markiert und für $u[a]$ ist dann ebenfalls Fall 1 eingetreten. Hat aber $u[a]$ die Boolesche Markierung 0, dann wird $R(F[a0])$ als neues Element in H aufgenommen[10] und für $u[a]$ ist Fall 2 der Hilfsbehauptung eingetreten. Dies schließt den induktiven Beweis der Hilfsbehauptung und damit auch den Gesamtbeweis ab. **qed.**

16.5 Der Satz von Karp und Lipton

Dieser Abschnitt ist dem Beweis des folgenden Resultates gewidmet:

Satz 16.13 (Karp und Lipton, 1980) $NP \subseteq P/poly \Rightarrow PH = \Sigma_2 = \Pi_2$.

Beweis Wir merken zunächst an, dass es ausreicht, $\Pi_2 \subseteq \Sigma_2$ zu zeigen. Hieraus folgt nämlich durch Dualisierung $\Sigma_2 = \text{co-}\Pi_2 \subseteq \text{co-}\Sigma_2 = \Pi_2$ und somit $\Sigma_2 = \Pi_2$, was wiederum (Anwendung 2 des Satzes von Wrathall) $PH = \Sigma_2$ zur Folge hat.

[10] Wenn dieses Element schon zu H gehören würde, dann wäre entweder $u[a0]$ gar nicht erst aufgebaut worden oder für $u[a0]$ wäre der Fall 2 der Hilfsbehauptung eingetreten.

Sei nun L eine beliebige aber fest ausgewählte Sprache aus Π_2. Somit existieren ein Polynom p und eine Sprache $L' \in \Sigma_1 = NP$, so dass

$$L = \{x|\ \forall y \in \{0,1\}^{p(|x|)} : \langle y, x\rangle \in L'\}\ . \tag{1}$$

Wir haben $L \in \Sigma_2$ zu zeigen und gemäß dem Satz von Wrathall reicht es dazu aus, eine Beschreibung der Sprache L vom „$\exists\forall$"-Typ anzufertigen.
Gemäß unserer Voraussetzung $NP \subseteq P/poly$ gehört die Sprache 3-SAT zu $P/poly$. Im Folgenden repräsentiere F_n eine CNF-Formel über den Booleschen Variablen $v_1, \ldots, v_n$ mit Klauseln der Länge höchstens 3 (kurz: 3-Klauseln).[11] Insgesamt gibt es nur $O(n^3)$ solcher 3-Klauseln, weswegen die binäre Kodierungslänge von F_n polynomiell in n, sagen wir durch $q(n)$, nach oben beschränkt ist. Im folgenden identifizieren wir die Formel F_n mit ihrer binären (auf Länge $q(n)$ ausgepolsterten) Kodierung, d.h., $F_n \in \{0,1\}^{q(n)}$. Wegen 3-SAT $\in P/poly$ gibt es eine 3-SAT realisierende polynomielle SKF (C_n^{3-SAT}). Schaltkreis C_n^{3-SAT} erhält also als Eingabe eine CNF-Formel F_n und gibt eine 1 aus gdw F_n erfüllbar ist. Auch C_n^{3-SAT} identifizieren wir mit seiner binären Kodierung, d.h., $C_n^{3-SAT} \in \{0,1\}^{r(n)}$ für ein geeignet gewähltes Polynom $r(n)$. Der Schlüssel zum Beweis liegt in folgender

Definition Eine Kollektion $(C_0, \ldots, C_m)$ von Schaltkreisen, wobei $C_i \in \{0,1\}^{r(i)}$ einen Schaltkreis mit $q(i)$ Eingangsknoten repräsentiert, heißt *3-SAT Tester*, falls

$$\forall i = 0, \ldots, m, \forall F_i \in \{0,1\}^{q(i)} : C_i(F_i) = 1 \Leftrightarrow F_i \in \text{3-SAT}\ .$$

Offensichtlich ist zum Beispiel $(C_0^{3-SAT}, \ldots, C_m^{3-SAT})$ ein 3-SAT Tester.
Wir verwenden jetzt das Konzept des 3-SAT Testers, um, ausgehend von der Beschreibung (30), zu einer neuen Beschreibung (vom „$\exists\forall$-Typ") der Sprache L zu gelangen. Hierzu nutzen wir $L' \leq_{pol}$ 3-SAT aus und bedienen uns einer Reduktionsabbildung

$$\langle y, x\rangle \mapsto F_{s(n)}^{x,y}\ ,$$

die einem String $\langle y, x\rangle$ eine CNF-Formel $F_{s(n)}^{x,y}$ über den Booleschen Variablen $v_1, \ldots, v_{s(n)}$ mit Klauseln der Länge höchstens 3 zuordnet, so dass

$$\langle y, x\rangle \in L' \Leftrightarrow F_{s(n)}^{x,y} \in \text{3-SAT}\ .$$

Die Beschreibung (30) von L lässt sich dann umschreiben wie folgt:

$$\begin{aligned} L \ = \ \Big\{ x \ \Big|\ & \exists (C_0, \ldots, C_{s(|x|)}) \in \times_{i=0}^{s(|x|)} \{0,1\}^{r(i)}, \forall y \in \{0,1\}^{p(|x|)} : \\ & C_{s(|x|)}\left(F_{s(|x|)}^{x,y}\right) = 1 \text{ und } (C_0, \ldots, C_{s(|x|)}) \text{ ist ein 3-SAT Tester} \Big\}\ . \end{aligned}$$

Die Bedingung $C_{s(|x|)}\left(F_{s(|x|)}^{x,y}\right) = 1$ lässt sich in Polynomialzeit testen (CIRCUIT EVALUATION). Falls die Sprache der 3-SAT Tester zu co-$NP = \forall[P]$ gehört, dann können wir aus der neuen Beschreibung von L folgern, dass

$$L \in (\exists)_{\text{pol}}(\forall)_{pol}(\forall)_{pol}[P] = (\exists)_{\text{pol}}(\forall)_{pol}[P]\ ,$$

[11] Es ist hier ausnahmsweise einmal praktischer auch Klauseln mit weniger als 3 Literalen zuzulassen.

womit der Beweis für $L \in \Sigma_2$ erbracht wäre. Bleibt also nur noch zu zeigen, dass

$$\text{3-SAT Tester} \in (\forall)_{pol}[P] \ .$$

Zu diesem Zweck nutzen wir wieder aus, dass

$$F_n \in \text{3-SAT} \Leftrightarrow F_n[0] \in \text{3-SAT} \vee F_n[1] \in \text{3-SAT} \ ,$$

wobei $F_n[a]$ wieder für die CNF-Formel stehe, die aus F_n vermöge einer partiellen Belegung $a \in \{0,1\}^j$ hervorgeht. Für $(C_0, \ldots, C_m) \in \times_{i=0}^{m} \{0,1\}^{r(i)}$ erhalten wir folgende äquivalente Bedingungen:

$(C_0, \ldots, C_m)$ ist ein 3-SAT Tester

$$\begin{aligned}
&\Leftrightarrow \quad \big(\forall F_m \in \{0,1\}^{q(m)} : C_m(F_m) = C_{m-1}(F_m[0]) \vee C_{m-1}(F_m[1])\big) \\
&\qquad \wedge \left((C_0, \ldots, C_{m-1}) \text{ ist ein 3-SAT Tester}\right) \\
&\Leftrightarrow \\
&\ldots \\
&\Leftrightarrow \quad \left(\forall (F_1, \ldots, F_m) \in \times_{i=1}^{m} \{0,1\}^{q(i)}, \forall i \in \{1, \ldots, m\} : \overbrace{C_i(F_i) = C_{i-1}(F_i[0]) \vee C_{i-1}(F_i[1])}^{\text{in Polynomialzeit testbar}} \right) \\
&\qquad \wedge \underbrace{(C_0 \text{ ist ein 3-SAT Tester})}_{\text{in konstanter Zeit testbar}}
\end{aligned}$$

Wir haben damit für die Sprache der 3-SAT Tester eine Beschreibung vom „∀-Typ“ gefunden, womit der gesamte Beweis abgeschlossen ist. **qed.**