Über die Struktur polynomialer Reduzierbarkeit

Stefan Büttcher
<stefan@buettcher.org>

Eine Wiedergabe des zentralen Ergebnisses

$$P \neq NP \Rightarrow NP - P \neq NPC$$

in Richard E. Ladner:
„On the Structure of Polynomial Time Reducibility"

Vortrag im Küchenkolloquium der Henke-WG
Januar 2003

1 Vorbereitung

Es sind zunächst einige Begrifflichkeiten zu klären, ohne die eine weitere Einlassung keinen Sinn zu machen scheint.

- Ein **Entscheidungsproblem** ist ein Problem, das mit der Aufgabe verbunden ist, einen Algorithmus zu finden, der für jedes $x \in \Sigma^*$ (bzw. $x \in \{0,1\}^*$) bestimmt, ob $x \in A$ für ein festes $A \subseteq \Sigma^*$ (bzw. $A \subseteq \{0,1\}^*$). A heißt dann *Sprache* oder auch manchmal selbst *Problem*.

- Eine allgemeinere Klasse als die der Entscheidungsprobleme bilden die **Suchprobleme**. Ein Suchproblem Π besteht aus einer Menge D_Π endlicher Objekte, den *Instanzen*, und für jede Instanz $I \in D_\Pi$ einer Menge $S_{\Pi,I}$, den *Lösungen zu I*. Ein Algorithmus *löst* ein Suchproblem, wenn er für jede Eingabe $I \in D_\Pi$ ein $s \in S_{\Pi,I}$ ausgibt bzw. „NO", falls $S_{\Pi,I}$ leer ist.

- Das formale Gegenstück zum Suchproblem ist die **String-Relation**. Für ein beliebiges endliches (nichtleeres) Alphabet Σ ist eine String-Relation eine binäre Relation $R \subseteq \Sigma^+ \times \Sigma^+$. Durch $R := \{(x,s) : \ x \in L\}$ mit festem s lässt sich eine Sprache L leicht als String-Relation darstellen. Eine Funktion $f : \ \Sigma^+ \to \Sigma^+$ *realisiert* eine String-Relation R genau dann, wenn $f(x) = \epsilon$, falls $\nexists\, y : \ (x,y) \in R$, und $f(x) = y$ für irgendein $(x,y) \in r$ sonst.

- Ein **Orakel** ist eine Turing-Maschine, die eine beliebige Transformation durchführt, also zum Beispiel ein Entscheidungsproblem löst. Dabei spielt es keine Rolle, auf welche Weise und mit welchem Aufwand sie das tut.

- Eine **Orakel-Turing-Maschine** (OTM) ist eine gewöhnliche DTM, die um ein *Orakel-Band* erweitert wurde. Das Orakel-Band, das in gewohnter Weise mit $.., -2, -1, 0, 1, 2, ..$ durchnummeriert wird, besitzt wie das Arbeitsband einen Lese-Schreib-Kopf.

  Eine OTM ist einer DTM sehr ähnlich, es sind jedoch die folgenden Veränderungen vorzunehmen:

- Die Menge Q der Maschinenzustände erhält außer dem Startzustand q_0 und dem Haltzustand q_h nun auch noch einen Orakel-Konsulationszustand q_c und einen Resume-Computation-Zustand q_r als Zustände mit besonderer Funktion.
- Ist der aktuelle Zustand q_c, so wird das Orakel befragt. Dabei wird der aktuelle Inhalt x des Orakel-Bands (beginnend in der Zelle 1) gemäß der Orakelfunktion g transformiert, das Orakel löst also innerhalb eines Schritts ein Suchproblem. Nach Befragung des Orakels steht (wiederum beginnend in der Zelle 1) $y = g(x)$ auf dem Orakel-Band, rechts und links von y jeweils unendlich viele Blanks. Der Zustand der OTM geht in q_r über.
- Die Übergangsfunktion muss entsprechend angepasst werden, damit die OTM auch auf das Orakel-Band schreiben und von ihm lesen kann:

  $$\delta : (Q - \{q_h, q_c\}) \times \Gamma \times \Sigma \to Q \times \Gamma \times \Sigma \times \{-1, 0, +1\} \times \{-1, 0, +1\}.$$

- In allen anderen Fällen verhält sich die OTM wie eine DTM.

Abschließend soll noch die Terminologie hinsichtlich der Komplexitätsklassen, mit denen wir uns befassen, festgelegt werden: Die Klasse der in polynomialer Zeit auf deterministische Weise lösbaren Probleme soll mit **P** bezeichnet werden, die der in polynomialer Zeit auf nichtdeterministische Weise lösbaren mit **NP**. Innerhalb von **NP** sei **NPC** die Klasse der **NP**-vollständigen Probleme.

2 Many-One- und Turing-Reduzierbarkeit

In den Anfängen der **NP**-Theorie haben sich zwei verschiedene Arten polynomialer Reduzierbarkeit herausgebildet, die auf der von Post festgeklopften *Many-One-Reduzierbarkeit* und *Turing-Reduzierbarkeit* basieren.

Cook hat in [Cook1971] polynomiale Turing-Reduzierbarkeit verwandt, um **NP**-Härte von bestimmten Problemen nachzuweisen, während Karp in [Karp1972] polynomiale Transformierbarkeit (Many-One-Reduzierbarkeit) benutzt hat, um die **NP**-Härte bestimmter Probleme zu zeigen. Man spricht daher auch manchmal von *Cook-Reduzierbarkeit* bzw. *Karp-Reduzierbarkeit.*

2.1 Many-One-Reduzierbarkeit

Eine Sprache (ein Entscheidungsproblem) $A \subseteq \Sigma^*$ lässt sich auf eine Sprache (ein Entscheidungsproblem) $B \subseteq \Sigma^*$ many-one-reduzieren, falls es eine berechenbare Funktion $f: \Sigma^* \to \Sigma^*$ gibt, so dass

$$x \in A \Leftrightarrow f(x) \in B \quad \forall\, x \in \Sigma^*$$

gilt. Man spricht dann häufig auch von einer Transformation. Ist die Funktion f durch eine deterministische Turing-Maschine in polynomialer Zeit berechenbar, das der Transformation zugeordnete Suchproblem also in **P**, dann nennt man die resultierende Reduktion eine *polynomiale Many-One-Reduktion* (*polynomiale Transformation*).

Die Eigenschaft zweier Probleme A und B „A lässt sich polynomial auf B transformieren" schreibt man formal als $A \leq_m^P B$, wobei das m für „many-one" steht und das P für „polynomial". Gilt sowohl $A \leq_m^P B$ als auch $B \leq_m^P A$, so schreibt man $A \equiv_m^P B$ und hat eine Äquivalenzrelation auf der Menge der **NP**-Probleme erklärt.

2.2 Turing-Reduzierbarkeit

Von einem etwas anderen Standpunkt aus betrachtet, lässt sich eine Many-One-Reduktion als Einbettung einer Subroutine in ein Programm ansehen: Ein Entscheidungsproblem Π wird auf ein anderes Entscheidungsproblem Π' transformiert, indem zunächst für eine Eingabe-Instanz I eine äquivalente Instanz I' $(= f(I))$ konstruiert wird. Der Ausgabewert des Algorithmus („yes" oder „no") ist dann der Ausgabewert des Unterprogramms, das das Problem Π' löst und mit der Eingabe I' aufgerufen wird. Sind die Transformation und das Unterprogramm zur Lösung von Π' in polynomialer Zeit ausführbar, so erhält man auf diese Weise ein polynomiales Programm zum Lösen von Π.

Dieser Ansatz, innerhalb eines Programms ein Unterprogramm aufzurufen, lässt sich verallgemeinern, indem man nicht nur einen einzigen, sondern viele Aufrufe der Subroutine erlaubt. Ist die Anzahl der Subroutinenaufrufe polynomial in der Länge der Eingabe I, so erhält man wiederum einen polynomialen Algorithmus zum Lösen von Π.

Die formale Definition von Turing-Reduzierbarkeit hat natürlich mit Turing-Maschinen zu tun. Ein Problem A lässt sich auf ein Problem B turing-reduzieren, falls eine OTM existiert, die A löst und dabei ein Orakel verwendet, das B lösen kann. Dabei werden bestimmte Anforderungen an die Anzahl der Arbeitschritte und der Befragungen des Orakels gestellt. Löst die OTM das Problem A in polynomial vielen Schritten und mit Hilfe polynomial vieler Befragungen des Orakels, so spricht man von polynomialer Turing-Reduktion.

Gänzlich analog zur Many-One-Reduktion schreibt man dann $A \leq_T B$ und $A \equiv_T B$ bzw. $A \leq_T^P B$ und $A \equiv_T^P B$.

2.3 Gemeinsamkeiten und Unterschiede

Da die Many-One-Reduktion den Spezialfall der Turing-Reduktion mit einem einzigen Subroutinenaufruf und sofortiger Ausgabe des Ergebnisses darstellt, ergibt sich:

$$A \leq_m B \Rightarrow A \leq_T B \ , \ A \not\leq_T B \Rightarrow A \not\leq_m B$$

und ebenso:

$$A \leq_m^P B \Rightarrow A \leq_T^P B \ , \ A \not\leq_T^P B \Rightarrow A \not\leq_m^P B.$$

Interessanter ist aber, dass Turing-Reduktion, im Gegensatz zu Many-One-Reduktion, nicht zwischen einem Problem und seinem Komplement unterscheiden kann (da die Transformation von „YES" auf „NO" und umgekehrt leicht nach dem ersten Aufruf des Orakels vorgenommen werden kann). Für Cook gab es daher zunächst keine Veranlassung, zwischen den Klassen **NP** und co-**NP** zu unterscheiden. Diese Verfeinerung brachte erst Karp mit seiner Art der polynomialen Reduktion.

Wir haben gesehen, dass $\leq_m^P$ und $\leq_T^P$ Quasi-Ordnungen sind, aus denen sich – wie oben angegeben – leicht Äquivalenzklassen bauen lassen. Durch die Definition

$$A \bigoplus B := \{0x : x \in A\} \cup \{1x : x \in B\}$$

kann das Vereinigungsproblem von A und B konstruiert werden. $A \bigoplus B$ ist das Supremum von A und B bezüglich $\leq_m^P$ bzw. $\leq_T^P$, es liegt also in beiden Fällen ein oberer Halbverband vor.

An den Stellen, wo die Unterscheidung zwischen den verschiedenen Arten der Reduktion nicht notwendig ist, soll im Weiteren einfach der Begriff *polynomial reduzierbar* oder, falls auch die Polynomialität klar ist, noch einfacher *reduzierbar* verwendet werden.

3 Existenz NP-unvollständiger Probleme in NP-P

Wir gehen im Folgenden davon aus, dass $\mathbf{P} \neq \mathbf{NP}$ gilt, die $\mathbf{NP}$-vollständigen Probleme also nicht in $\mathbf{P}$ liegen.

Weiter oben hatten wir gesehen, dass durch $\leq_m^P$ ($\leq_T^P$) bzw. $\equiv_m^P$ ($\equiv_T^P$) eine Quasi-Ordnung bzw. eine Partition auf Mengen von Problemen erklärt ist. Es soll nun zunächst gezeigt werden, dass es innerhalb dieser Quasi-Ordnung keine maximale Klasse geben kann.

Beweis. Sei $M_0, M_1, \ldots$ eine Aufzählung aller Orakel-Turing-Maschinen und A ein entscheidbares Problem. Sei ferner $\delta(A) := \{1^i : 1^i \notin M_i(A)\}$ eine Art Diagonalmenge. Da A entscheidbar ist, ist auch $\delta(A)$ berechenbar. Es gilt: $\delta(A) \not\leq_T^P A$, denn $\delta(A) = M_k(A)$ für irgendein k führt sofort zum Widerspruch. Da nun

$$A \leq_m^P A \bigoplus \delta(A) \ \wedge \ A \bigoplus \delta(A) \not\leq_T^P A,$$

ist klar, dass es keine maximale Klasse bezüglich einer der beiden Reduktionsarten geben kann.

Um zu zeigen, dass $\mathbf{NP} - \mathbf{P} \neq \mathbf{NPC}$ gilt, genügt es, zu beweisen, dass es außerhalb von $\mathbf{P}$ ein Problem A gibt, das sich auf ein $\mathbf{NP}$-vollständiges Problem B reduzieren lässt, während B sich nicht auf A reduzieren lässt.

Satz. (Ladner) Sei B berechenbar und $B \notin \mathbf{P}$. Dann existiert ein berechenbares A, so dass $A \notin \mathbf{P}$, $A \leq_m^P B$ und $B \not\leq_T^P A$.

Beweis. Wir nehmen an, dass durch $P_0, P_1, \ldots, T_0, T_1, \ldots, M_0, M_1, \ldots$ Aufzählungen der jeweils in polynomialer Zeit laufenden erkennenden Turing-Maschinen (P_i), der transformierenden Turing-Maschinen (T_j) und der Orakel-Maschinen (M_k) gegeben sind. Mit P_i soll hinfort sowohl die Turing-Maschine als auch die von ihr erkannte Sprache bezeichnet werden, mit T_j neben der Turing-Maschine auch die von ihr berechnete Funktion und mit $M_k(A)$ die vom Automaten M_k mit Orakel A erkannte Sprache.

Sei nun T eine Transformations-DTM mit polynomialer Laufzeit, deren genaues Aussehen später noch erklärt werden soll. Dann definieren wir

$$A := \{x \in B : |T(x)| \text{ ist gerade}\}.$$

Offenbar ist $A \leq_m^P B$, denn die geforderte Reduktion kann durch eine Überführungsfunktion

$$g(x) := \begin{cases} x, & \text{falls } |T(x)| \text{ gerade ist} \\ x_0, & \text{falls } |T(x)| \text{ ungerade ist} \end{cases}$$

realisiert werden, wobei $x_0 \in \Sigma^* - B$ (B kann nicht Σ^* sein, denn sonst wäre $B \in \mathbf{P}$, was nach Definition nicht der Fall ist). Zu zeigen bleibt, dass $A \notin \mathbf{P}$ und $B \not\leq_T^P A$. Dafür sehen wir uns zunächst die Funktion T etwas genauer an:

Es sind drei Fälle zu untersuchen, von denen zwei schnell abgehandelt werden können: $T(\varepsilon) = \varepsilon$, und $T(x) = T(0^{|x|})$ für $|x| \geq 1$. Die Brisanz ist also komplett auf den Fall $x = 0^n$ komprimiert. Für 0^n, $n \geq 1$ führt T folgende Operationen aus:

1. Verbringe n Rechenschritte damit, die Sequenz $T(\varepsilon), T(0), T(00), T(000), \ldots$ zu konstruieren. Sei $T(0^m)$ das letzte Element dieser Sequenz, das berechnet werden kann, bis die Abbruchbedingung (n Schritte) erreicht ist.

2. Es sind zwei Fälle zu unterscheiden:

    - Fall (a): $l := |T(0^m)|$ ist gerade. Dann sei $i := \frac{l}{2}$. T versucht nun n Schritte lang, ein $z \in \Sigma^*$ zu finden, so dass $A(z) \neq P_i(z)$, wobei es sich selbst simulieren muss. Wenn kein solches z gefunden werden kann, gibt T als Ergebnis 1^{2i} aus, sonst 1^{2i+1}.

- Fall (b): $l := |T(0^m)|$ ist ungerade. Dann sei $i := \frac{l-1}{2}$. T versucht nun n Schritte lang, ein $z \in \Sigma^*$ zu finden, so dass $B(z) \neq M_i(A)(z)$ ist. Dafür müssen B und M_i simuliert werden. Kann kein solches n innerhalb der vorgegebenen Zeit gefunden werden, wird als Ergebnis 1^{2i+1} ausgegeben, sonst 1^{2i+2}.

Die angegebene Funktion T lässt sich in polynomialer Zeit berechnen. Wenn also $\text{Bild}(T) = \{1\}^*$ ist, dann haben wir $A \leq_m^P B$ und wegen der Struktur von T, die gerade dafür sorgt, dass alle Bedingungen

$$(0) A \neq P_0, (1) B \neq M_0(A), (2) A \neq P_1, (3) B \neq M_1(A), \ldots$$

erfüllt werden, weil der Übergang von i auf $i+1$ eben erst dann erfolgt, wenn die Bedingung (i) erfüllt worden ist:

$$A \neq P_i \ \forall\, i \in \mathbb{N}_0 \ \wedge \ B \neq M_i(A) \ \forall\, i \in \mathbb{N}_0 \,,$$

also $A \notin \mathbf{P}$ und $B \not\leq_T^P A$.

Behauptung. $|T(0^n)| \leq |T(0^{n+1})| \leq |T(0^n)| + 1 \ \ \forall\, n \in \mathbb{N}_0$ und $\text{Bild}(T) = \{1\}^*$.

Beweis. Der erste Teil lässt sich leicht zeigen: Dass $|T(0^n)|$ monoton mit n steigt, ist klar. Durch Induktion nach n wird nun schnell klar, dass in beiden Fällen (1 und 2) entweder $T(0^{n+1}) = T(0^n)$ ist oder noch eine einzige weitere 1 angehängt wird.

Für $\text{Bild}(T)$ ergeben sich nun aufgrund des ersten Teils nur noch zwei Möglichkeiten: Da wegen $T(\varepsilon) = \varepsilon$ das leere Wort in $\text{Bild}(T)$ enthalten ist, ist entweder $\text{Bild}(T) = \{1\}^*$, oder es gibt ein $n_0 \in \mathbb{N}_0$, so dass $T(0^n) = T(0^{n_0}) \ \forall\, n \geq n_0$. Wir elimieren die zweite Möglichkeit, so dass nur noch der Fall $\text{Bild}(T) = \{1\}^*$ übrig bleibt.

Sei also $T(0^n) = T(0^{n_0}) \ \forall\, n \geq n_0$. Es sind zwei Fälle zu unterscheiden:

1. $j := |T(0^{n_0})|$ ist gerade. Sei $i := \frac{j}{2}$. In diesem Fall ist $A(x) = B(x)$ für alle x mit $|x| \geq n_0$. Wegen $B \notin \mathbf{P}$ folgt: $A \notin \mathbf{P} \Rightarrow A \neq P_i$. Sei z die kleinste Eingabe (bezüglich der natürlichen Ordnung auf Σ^*), für die $A(z) \neq P_i(z)$. Sei ferner n groß genug, so dass T in höchstens n Schritten die Sequenz $T(\varepsilon), T(0), \ldots, T(0^{n_0})$ berechnen und ebenfalls die entsprechenden Simulationen von T und P_i bis zum Input z ablaufen lassen kann. Dass es ein solches n geben muss, ist klar. Es gilt dann:

   $$T(0^n) = 1^{2i+1} = 1^{j+1} \Rightarrow T(0^n) \neq T(0^{n_0}).$$

   Widerspruch!

2. $j := |T(0^{n_0})|$ ist ungerade. Sei $i := \frac{j-1}{2}$. In diesem Fall muss $x \notin A \ \forall\, |x| \geq n_0$ gelten. A enthält damit nur endlich viele Elemente, und $M_i(A) \in \mathbf{P}$. Wegen $B \notin \mathbf{P}$ muss dann auch $B \neq M_i(A)$ gelten. Es gibt nun wiederum ein kleinstes z, das den Unterschied zwischen B und $M_i(A)$ aufweist. Sei $n \geq n_0$ groß genug, so dass die Sequenz $T(\varepsilon), T(0), \ldots, T(0^{n_0})$ berechnet und B und M_i simuliert werden können. Es folgt ganz analog zum obigen Schluss:

   $$T(0^n) = 1^{2i+2} = 1^{j+1} \Rightarrow T(0^n) \neq T(0^{n_0}).$$

   Widerspruch!

Es ist nun bewiesen, dass $\text{Bild}(T) = \{1\}^*$ ist, der auf diese Weise konstruierte Transformator T somit die Ungleichungen $A \neq P_i, B \neq M_i(A)$ für alle $i \in \mathbb{N}_0$ erzwingt, so dass $A \notin \mathbf{P}, A \leq_m^P B, B \not\leq_T^P A$. □

4 Anmerkungen

Statt Ausgabewerte der Form 1^j zu berechnen kann man T auch einfach auftragen, die Zahl j zu berechnen und in irgendeiner geeigneten Kodierung zu speichern. Dass hier von Ladner die Steinzeitkodierung gewählt wurde, hat gegenüber den meisten Anwendungen keine weiteren Auswirkungen, da die Funktion g ,,beinahe konstant" ist:

Die durch den Transformator T berechnete Funktion g hat ein seltsames Wachstumsverhalten, sie wächst nämlich fast überhaupt nicht. Wenn man die Erfüllung der Bedingungen (0), (1), ... einmal außer Acht lässt, ergibt sich bereits: $g \in \mathcal{O}(log(log(n)))$.

Die im letzten Abschnitt dargestellten Ergebnisse haben in den folgenden Jahren zu einer ganzen Reihe weiterer Ergebnisse über die Struktur der Reduktion und der Klasse **NP** geführt.

Weiter oben wurde darauf hingewiesen, dass es hinsichtlich $\leq_m$ bzw. $\leq_T$ keine größte Klasse geben kann. Wenn wir unser Augenmerk statt auf die größte nun einmal kurz auf die kleinste Klasse richten, dann ist sofort klar, dass **P** natürlich diese kleinste Klasse ist, weil $P \leq A \ \ \forall\, P \in \mathbf{P}$ für ein beliebiges Problem A. Andererseits sagt uns der in 3. bewiesene Satz von Ladner, dass für jedes Problem $A \notin \mathbf{P}$ ein Deformator $D \in \mathbf{P}$ existiert (oben war das der Transformator T),, so dass $A \cap B \notin \mathbf{P}$ und $A \cap B \leq A$ und $A \not\leq A \cap B$. Es lassen sich also unendliche echt absteigende Ketten erzeugen.

Literatur

[Cook1971] S.A. Cook: ,,The complexity of theorem-proving procedures", Proceedings of the 3rd annual ACM symposium on Theory of Computing, ACM, New York, 151-158.

[Karp1972] R.M. Karp: ,,Reducibility among combinatorial problems" in R.E. Miller and J.W. Thatcher, ,,Complexity of Computer Computations", Plenum Press, New York, 85-103.

[Ladner1975] R.E. Ladner: ,,On the Structure of Polynomial Time Reducibility", Journal of the ACM 22, ACM, New York, 155-171.

Diesen Artikel findet man zum Download als Postscript oder PDF unter
`http://stefan.buettcher.org/cs/ladner1975/`

Weitere Aufsätze im Rahmen des Küchenkolloquiums gibt es hoffentlich bald unter
`http://stefan.buettcher.org/kuechenkolloquium/`

Kommentare, Kritik und Anregungen sind herzlich erbeten!