Über das Studium der Grundlagen der Informatik

Johannes Waldmann, Institut für Informatik, Uni Leipzig

17. September 2001

1 Was ist Informatik?

Das Wort stammt von “Information” (noch allgemeiner: “Wissen”) und die wollen wir *darstellen* und *verarbeiten*.

Zum Darstellen brauchen wir eine exakte *Sprache*. Das ist die Mathematik im Allgemeinen, und die formale Logik im Besonderen.

Das Verarbeiten beschreiben wir durch *Algorithmen*. Ein Algorithmus ist eine Vorschrift zum Umformen von Informationsdarstellungen.

Die Interpretation der Information ist im Kopf (das Wissen), nur die Darstellung (Zeichenfolgen) ist auf dem Papier oder der Festplatte.

2 Was ist ein Algorithmus?

Eine *endliche* Vorschrift, nach der ein Rechner (Mensch oder Maschine) aus *Eingaben Ausgaben* erzeugt, und zwar auf *klar festgelegte* Weise, d. h. ein anderer Rechner erhält bei gleichem Algorithmus und gleichen Eingaben identische Ausgaben.

Algorithmen gibt es schon sehr lange, denke an geometrische Konstruktionen, oder das schriftliche Multiplizieren, schließlich an mechanische Rechenmaschinen.

Man war lange Zeit der Meinung, daß sich *jede* Frage algorithmisch beantworten läßt. Leibnitz wollte eine Universalsprache erfinden, mit passenden

Umformungsregeln, so daß bei jedem Streit zweier Menschen (Philosophen, Politiker, ...) eine Entscheidung einfach durch Rechnung herbeizuführen sei.

In der Mathematik wollte man Polynomgleichungen beliebigen Grades lösen, und hatte auch Verfahren für die Grade 1 bis 4. Dann bewies Bolyai, daß Grad 5 nicht immer geschlossen durch Radikale lösbar ist. (Manchmal schon, aber meistens nicht.)

Hilbert frage 1900 (zehntes Problem), wie man entscheiden kann, ob eine gegebene Polynomgleichung (mit ganzen Koeffizienten) (wenigstens) eine ganzzahlige Lösung besitzt.

3 Gibt es einen Algorithmus für diese Aufgabe?

Wir betrachten ein Puzzlespiel: `http://www.informatik.uni-leipzig.de/~pcp/`

Die Idee stammt von Emil Post `http://www-groups.dcs.st-and.ac.uk:80/~history/Mathematicians/Post.html`

Formal haben wir eine Menge von Paaren von Zeichenketten (= Wörtern) $P = \{(u_1, v_1), \ldots, (u_n, v_n)\}$ und betrachten Wörter w, die aus u_i zusammengesetzt sind. Zu jedem solchen w gehört dann ein Wort w', das aus den entsprechenden v_i besteht. Es geht nun darum, ob es für gegebenes P Wörter w gibt, die mit ihrem w' übereinstimmen.

Wir wünschen uns einen Algorithmus, der auf Eingabe von M diese Frage beantwortet.

Wir können natürlich einfach probieren. Falls die Aufgabe eine Lösung hat, finden wir sie dadurch auch. Falls aber nicht? Woher wissen wir, wann wir mit Probieren aufhören dürfen? Der Algorithmus soll die Frage ja beantworten, und zwar in endlicher Zeit.

Beachte: in der Algorithmen-Definition ist *nicht* gefordert, daß jede Rechnung nur endlich lange dauert. Wenn wir jedoch, wie hier, eine Antwort auf eine Frage haben wollen, dann verlangen wir das.

4 Spezialfälle der vorigen Aufgabe

Weil wir zunächst keine Algorithmus finden, der PCP entscheidet, beschäftigen wir uns mit Spezialfällen. Wir beschränken die Anzahl der Paare in der Eingabe:

Bei zwei Stück passiert offenbar nicht viel. Für jedes n hat

$$\begin{pmatrix} 0^n1 & 1 \\ {\scriptstyle A} & {\scriptstyle B} \\ 0 & 0^n1 \end{pmatrix}$$

hat die kürzeste Lösung A^nB^n.

Das ist kein Beweis, aber tatsächlich ist seit 1982 bekannt, daß die PCP mit nur zwei Paaren entscheidbar sind.

Als nächstes drei Paare. Da gibt es einige PCP, die lösbar sind, aber deren kleinste Lösung sehr groß ist (siehe Rekord-Tabelle). An verschiedenen Plätzen der Welt werden diese Tiere gejagt, und wir in Leipzig sind einer davon.

Wir betrachten hier den Rekord für PCP(3,5): $\begin{pmatrix} 11010 & 1 & 11111 \\ {\scriptstyle A} & {\scriptstyle B} & {\scriptstyle C} \\ 11 & 10101 & 01 \end{pmatrix}$.

Zunächst stellen wir fest, daß jede Lösung mit A beginnen und mit C enden muß, und währendher das obere Wort immer länger als das untere ist.

Wir können auch das Verhältnis der Anzahlen der Paare in einer Lösung bestimmen: wir zählen die Ziffern 0:

$$2A - 2B - C = 0$$

und die Ziffern 1:

$$A - 2B + 4C = 0$$

Das ist ein Gleichungssystem mit der Lösungsmenge

$$A = 10x, B = 9x, C = 2x.$$

Nehmen wir nun an, die Differenz (also der Rest der oberen Wortes, der unten noch keine Entsprechung besitzt) ist von der Form 1^{2k+1}. (Das ist nach ACC für $k = 4$ der Fall.)

Dann ist diese Fortsetzung erzwungen:

$$1^{2k+1} \stackrel{A^{k+1}}{\to} (10101)^k \; 1010 \stackrel{B^{k+1}}{\to} 1^k$$

Wenn wir mit 1^{2k} starten (das ist am Anfang für $k = 0$ wahr!), dann

$$1^{2k} \stackrel{A^{k+1}}{\to} 0101(10101)^k \; 1010 \stackrel{CC}{\to} (10101)^k \; 10101^{10} \stackrel{B^k}{\to} 1^{9+k}$$

Das ergibt eine Kette, die bei 1^0 beginnt und endet:

$$\begin{aligned} 0 \to 9 \to 4 \to 11 \to 5 \to 2 \to 10 \to 14 \to 16 \\ \to 17 \to 8 \to 13 \to 6 \to 12 \to 15 \to 7 \to 3 \to 1 \to 0 \end{aligned}$$

Das sind 18 Schritte, dabei wurde genau 9 mal die zweite Regel benutzt, so daß 18 mal C im Lösungswort steht. Also ist (siehe oben) $x = 9$, und die Lösung hat die Länge $21x = 189$.

Die Zahl 18 können wir leicht erklären: statt $(2k+1) \mapsto k$ und $(2k) \mapsto k+9$ schauen wir uns die Abbildung mal in der anderen Richtung an: das ist nämlich $x \mapsto (2x+1) \pmod{19}$. Siehe $0 \mapsto 1 \mapsto 3 \mapsto 7 \mapsto 15 \mapsto 31 = 12 \mapsto \ldots$ Verschieben wir das noch um eins nach oben, also $1 \mapsto 2 \mapsto 4 \mapsto 8 \mapsto 16 \mapsto 32 = 13 \mapsto \ldots$, dann sehen wir genau die Folge $2^k \pmod{19}$. Jetzt wird klar, warum sie höchstens die Länge 18 haben kann: es gibt ja nur soviele Restklassen (die Klasse 0 kommt nicht vor). Daß auch alle wirklich erscheinen, heißt in der Algebra "2 ist eine primitive Wurzel modulo 19".

Das eben betrachtete PCP enthielt also versteckt eine Berechnungsvorschrift "teile durch zwei und addiere 9, wenn kein Rest auftrat", und diese führte von Eingabe 0 nach 18 Schritten zum Halt, und damit einer Lösung des PCP.

Wir können uns nun vorstellen, daß man mit größeren PCP (mehrere Paare) auch kompliziertere Vorschriften kodieren kann. In der Tat bewies Emil Post, daß man sogar *jeden* Algorithmus auch in einem PCP unterbringen kann.

5 Das Halteproblem

Die Lösbarkeit eines PCP entspricht damit dem Halten einer Berechnung. Das ist ein durchaus praktisches Problem:

Eingabe: ein Programmtext T und eine Zeichenkette X
Ausgabe: JA oder NEIN, je nachdem, ob eine Maschine, die T(X) ausführt, halten würde oder nicht.

Der naive Ansatz ist natürlich der, die Rechnung einer Maschine mit Programm T auf Eingabe X schrittweise zu simulieren, aber ist nicht das geforderte Entscheidungsverfahren, da wir dadurch niemals die Antwort NEIN erhalten können.

Wenn man das aber schafft, *ohne das Programm auszuführen*, dann wäre das sehr praktisch! Nie mehr auf Endlos-Schleifen warten, weil der Rechner sofort anhält mit der Fehlermeldung "wenn ich weiterrechne, werde ich sowieso nicht fertig"!

Aber vielleicht gibt es trickreichere Methoden? Man könnte ja nach bestimmten "verbotenen" Mustern in T oder in X oder während des Ablaufs der Simulation suchen. — Nein, das hat alles keinen Zweck, wie man durch folgendes Argument sieht:

Angenommen, es gibt doch einen Algorithmus H, der obiges Problem entscheidet. Dann konstruieren wir das Programm P mit einem Argument Y und der Eigenschaft "if H(Y,Y) then Endlos-Schleife else Stop". (Wenn wir den Programmtext für H haben, dann geht das ja.)

Was passiert nun beim Aufruf P(P)? Es gibt ja nur zwei Möglichkeiten: diese Rechnung hält, oder sie hält nicht. Angenommen, P(P) stoppt. Das passiert ja deswegen, weil die Bedingung im "if" falsch war, also H(Y,Y) = NEIN. Bei dem Aufruf war aber Y = P, also H(P, P) ist NEIN. Das bedeutet aber, daß das Programm P bei Eingabe P *nicht* hält, im Widerspruch zur Voraussetzung. (Alle Umformungen waren äquivalent, deswegen ergibt sich sofort auch im anderen Fall ein Widerspruch.)

6 Andere unentscheidbare Probleme

Wir wissen also (seit 1930/40, der angegebene Beweis war Church, Turing, Post und anderen bekannt) daß es Probleme gibt, die nicht durch einen Algorithmus gelöst werden können. Interessanterweise ist das typische solche Problem (das Halteproblem) eine Problem, bei dem wir einen Programmtext (d. h. eine Repräsentation eines Algorithmus) eingeben!

Wir müssen uns das nochmal verdeutlichen: die Funktion H(T,X) ist für alle

Argumente wohl definiert, sie "existiert" also. Was *nicht* existiert, ist ein Algorithmus, der H(T,X) ausrechnen kann. Anders gesagt, jeder Algorithmus, von dem sein Programmierer behauptet, er gebe immer die richtige Antwort, macht bei irgendeiner Eingabe eben *doch* einen Fehler.

Nun ist nicht nur das Halteproblem unentscheidbar, sondern auch viele andere Fragestellungen, zum Beispiel das eben betrachtete PCP. Wir führen eine *Reduktion* durch (vom Halteproblem auf das PCP): wenn wir ein Entscheidungsverfahren für das PCP haben, dann auch ein Entscheidungsverfahren für das Halteproblem. Wir müssen dazu aus dem Programmtext eine PCP-Instanz basteln. Da es ein Entscheidungsverfahren für das Halteproblem aber nicht gibt, existiert auch kein solches für das PCP.

Ebenfalls eine Klarstellung hier: es gibt nicht *das* unentscheidbare PCP. Jede Instanz hat entweder eine Lösung, oder sie hat keine. Worum es geht, ist: es gibt *keinen Algorithmus*, der für *jede* PCP-Instanzen entscheiden kann, ob sie eine Lösung hat.

7 Grammatiken

Das PCP wiederum kann man benutzen, um die Unentscheidbarkeit weiterer Fragen an formale Sprachen zu beweisen: es genügt ja jetzt, ein PCP zu kodieren! Betrachten wir ein Beispiel: eine *kontextfreie Grammatik* ist so eine Regelmenge:

$$G = \{S \to 11010S11, S \to 1S10101, S \to 11111S10, S \to 2\}$$

und eine *Ableitung* in einer Grammatik ist eine Folge von Regelanwendungen

$$S \to 11010S11 \to 1101011111S1011 \to 110101111111111S101011 \to \dots$$

bis schließlich das Symbol S verschwindet. Alle auf diese Weise ableitbaren Wörtern bilden die Menge $L(G)$. Enthält diese Menge ein nichtleeres Palindrom (d. h. ein Wort, das von vorn wie von hinten gelesen aussieht)? Offenbar ist das äquivalent zur Lösbarkeit des vorhin betrachteten PCP, also heißt in diesem Fall die Antwort JA.

Da wir aber gerade bewiesen haben, daß Lösbarkeit von PCP nicht entscheidbar ist, ist auch die folgende Frage nicht entscheidbar:

Eingabe: eine kontextfreie Grammatik G
Frage: enthält $L(G)$ ein Palindrom?

Da die Menge aller Palindrom selbst kontextfrei ist - sie wird erzeugt von der Grammatik

$$P \to 0P0, P \to 1P1, P \to \epsilon, P \to 0, P \to 1$$

, ist auch das unentscheidbar:

Eingabe: zwei kontextfreie Grammatiken G_1, G_2
Frage: haben $L(G_1)$ und $L(G_2)$ ein Wort gemeinsam?

Und wozu brauchen wir das? Grammatiken beschreiben Sprachen, zum Beispiel Programmiersprachen, und alle üblichen Programmiersprachen sind im wesentlichen kontextfrei. Man muß also beim Festlegen dieser Sprachen, und dann beim Verarbeiten von Programmen, sehr genau über Grammatiken bescheid wissen.

8 Folgerungen aus der Unentscheidbarkeit des Halteproblems

Zunächst scheint das eine ganz bestürzende Erkenntnis: es *gibt* unentscheidbare, d. h. algorithmisch unlösbare Probleme.

Man weiß aber, woran es liegt: Selbst-Anwendbarkeit. Das Programm H (aus dem Halteproblem) sollte über den Text von H selbst eine Aussage treffen. Immer, wenn ein System (eine Programmiersprache) so mächtig ist, daß man *innerhalb* des Systems selbst Aussagen *über* das System machen kann (diese Eigenschaft nennt man *Universalität*), dann ist es auch *unvollständig*.

Diese Aussage treffen wir auf verschiedenen Gebieten, zum Beispiel beim bekannten Paradoxon des Barbiers, der sich (oder nicht) selbst balbiert, oder der Menge aller Mengen, die nicht Element von sich selbst sind.

Es ist besonders hervorzuheben, daß Aussagen über Unentscheidbarkeit, völlig unabhängig von der technischen Entwicklung der Computer, immer richtig sind und bleiben. Also auch der dreimalsoschnelle Prozessor von übermorgen kriegt nicht *alle* PCPs raus, und wenn sich seine Programmmierer noch so anstrengen.

Das zeigt, wie wichtig und erfolgreich Grundlagenforschung in der Informatik ist.

Universalität ist einerseits etwas Wünschenswertes (man *will* einen Compiler für die Sprache C in der Sprache C selbst schreiben) aber andererseits eben gefährlich (viele Eigenschafte von C-Programmen sind unentscheidbar).

Da es Universalität und Entscheidbarkeit nicht gleichzeitig gibt, muß man wählen. Oft gibt man die Universalität auf (weil man sie gar nicht braucht) und gewinnt dafür Entscheidbarkeit (von Programm-Eigenschaften).

Alle diese Aspekte spielen auch beim Studium der Informatik eine wesentliche Rolle: die hier erwähnten Begriffe, Beispiele und Beweise behandeln wir im 4. Semester (Grundstudium, Berechenbarkeit).

Die Logik (2. Semester) und den Compilerbau (Hauptstudium) können wir auffassen als die einerseits theoretische, andererseits praktische Konstruktion eines universellen Systems.

Die verschiedenen (nicht mehr universellen) eingeschränkten Algorithmenmodelle behandeln wir im 3. Semester und 4. Semester. Dabei beschränken wir im 3. Semester (Automaten und formale Sprachen) den Befehlssatz (endliche Automaten: nur lesen, Kellerautomaten: beim Zurücksetzen löschen), im 4. Semester (Komplexität) die Rechenzeit (Polynomialzeit) und den Platz.

Wir haben damit *Maschinen*- (d. h. Algorithmen-Klassen) definiert, und untersuchen dann die entsprechenden *Problem*klassen (Problem P gehört zur Klasse C_P, wenn es eine Maschine $M \in C_M$ gibt, die P löst.)

Damit ist es schließlich möglich, die "innere Schwierigkeit" praktisch vorkommender Problem (Weglängen-Optimierungen, Stundenplan-Bauen, Behälter-Packen) festzustellen, indem wir sie in die richtige Problemklasse einordnen, inden wir Reduktionen von typischen (sogenannten *vollständigen*) Problemen der jeweiligen Klasse finden.

Dann können wir nämlich, bevor wir auch nur eine einzige Programmzeile getippt haben, abschätzen, mit welcher Laufzeit wir besten- oder schlimmstenfalls rechnen können, und welche Algorithmen dazu geeignet oder ungeeignet sind. So hat also das Studium der Grundlagen der Informatik doch ganz praktische Anwendungen. Oder, wie schon vor Jahrzehnten (über die Mathematik) gesagt: *Es gibt nichts praktischeres als eine gute Theorie.*