Theoretische Informatik 1

Prof. Dr. Hans-Jörg Kreowski

Studiengang Informatik

Sommersemester 2000

MZH 3260
Tel.: 2956, 3697 (Sekr.), Fax: 4322

E-Mail: kreo@informatik.uni-bremen.de
www.informatik.uni-bremen.de/theorie

1. Szenarien – theoretisch
   begründet den Sinn von Theorie im allgemeinen und von Berechenbarkeits- und Komplexitätstheorie im besonderen (vgl. Studienordnung Informatik vom 1.10.1993, Anlagen, S. 6).
2. Lauter Wörter
   führt wichtige Eigenheiten von Wörtern ein, die zum Entwurf und zur Analyse von Algorithmen auf dieser Datenstruktur benötigt werden.
3. Berechnung von Operationen auf Zeichenketten mit Hilfe bedingter Gleichungen – die Sprache CE-S
   erweitert die Überlegungen des zweiten Kapitels zu einer Spezifikationssprache für Operationen auf Zeichenketten und liefert damit einen Ansatz, in dem Spezifikationen eine formale Semantik besitzen und Korrektheitseigenschaften nachgewiesen werden können. Fast nebenbei ergibt sich eine Präzisierung des Berechenbarkeitsbegriffs.
4. Wie beim Rechnen die Zeit vergeht
   erläutert den Umstand des Zeitaufwands beim Berechnen von Algorithmen und stellt Konzepte zur Verfügung, die es in günstigen Fällen erlauben, den Zeitaufwand zu ermitteln.
5. Matrizenmultiplikation schneller und schneller
   vergleicht den Zeitaufwand verschiedener Algorithmen am Beispiel von Matritzenmultiplikation.
6. PASCALchen macht Funktionen berechenbar
   präzisiert den Begriff der berechenbaren Funktion anhand der imperativen Programmiersprache PASCALchen.
7. Programme sind aufzählbar – doch ihr (Ver-)Halten macht Kummer
   zeigt einerseits, daß alle PASCALchen-Programme und damit auch alle berechenbaren Funktionen mittels eines Algorithmus aufgezählt werden können und führt andererseits anhand des Halteproblems von Programmen an die Grenzen der Berechenbarkeit.
8. Ein Programm rechnet für alle
   stellt eine berechenbare universelle Funktion vor und kommentiert die Grenzlinie, welche diese vom Halteproblem trennt.
9. Turing-Maschinen
   führt den Begriff der Turingmaschinen ein und stellt einen Bezug zwischen diesem Maschinenmodell und der Sprache CE-S her.
10. Literaturhinweise
    enthält eine Liste von Büchern, die einen Einblick in die Gebiete der Theoretischen Informatik bieten bzw. das dafür benötigte Basiswissen liefern.

1 Szenarien – theoretisch

Mit diesem einführenden Kapitel sollen Gegenstände, Ideen und Grundsätze der theoretischen Informatik dargestellt und motiviert werden. Insbesondere wird auf Sachverhalte und Überlegungen eingegangen, die in der Berechenbarkeits- und Komplexitätstheorie eine wichtige Rolle spielen. Den Ausgangspunkt bilden einige Szenarien, in denen fiktiv Anwendungen von Informationstechnik und Situationen aus dem Leben von Informatikerinnen und Informatikern beschrieben werden. Entstehende Ähnlichkeiten mit tatsächlichen Problemen sind beabsichtigt. Die Szenarien sind so zugespitzt, daß die korrekte Beantwortung der gestellten Fragen Kenntnisse und Einsichten der theoretischen Informatik verlangt. Wer die Antworten weiß oder sich richtig überlegen kann, braucht diese Lehrveranstaltung nicht mitzumachen, sondern kann sich den Leistungsnachweis nach einer kurzen Rücksprache abholen.

Anknüpfend an die Szenarien werden einige Wesenszüge der theoretischen Informatik erläutert und einige zentrale Gesichtspunkte von Berechenbarkeits- und Komplexitätstheorie angerissen.

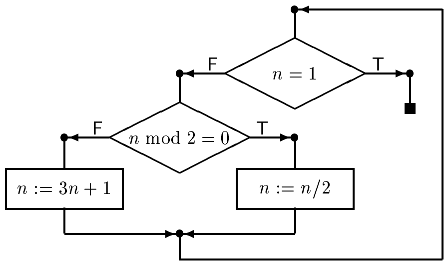
1.1 Terminationstest endlich lieferbar

In der Zeitschrift ITCT, die monatlich von etwa 40.000 Computerfachleuten gelesen wird, fand sich neulich folgende Anzeige:

> Müssen Sie bei Ihren Datenbankanfragen oft lange auf Antwort warten, oder kommt häufig gar keine vernünftige Antwort? Sind Ihre Programme sehr zeitintensiv? Erhalten Sie bei der Ausführung Ihrer Programme oft Fehlermeldungen statt eines Ergebnisses? Brechen Sie häufig Berechnungen ab, weil Sie nicht mehr warten wollen oder können? Wenn Sie eine dieser Fragen mit JA beantworten, dann haben wir etwas für Sie: *HaltCheck*. Die Weltneuheit *HaltCheck* läuft auf allen gängigen Rechnern und ist für die Datenbankanfragesprache SQL und für die Programmiersprachen C++ und ML erhältlich. Für jede Datenbankanfrage bzw. für jedes Programm und jede Eingabe testet es in Sekundenschnelle die Termination und gibt im positiven Fall eine obere Zeitschranke an, bis zu der man höchstens auf das Berechnungsergebnis warten muß. *HaltCheck* spart Ihnen Zeit und Nerven. Diese sensationelle Entwicklung ist im Fachhandel für nur DM 1.199,– erhältlich. Kaufen Sie sofort.

Ist das Angebot interessant und günstig? Sollte man mit dem Kauf warten, bis der Preis fällt? Oder was ist von einem derartigen Hilfsprogramm zu halten?

Wenn ein Programm wie *HaltCheck* existierte, wäre es ausgesprochen nützlich. Wer das bezweifelt, möge folgendes Flußdiagramm betrachten, das die Berechnungen der sogenannten Ulam-Funktion beschreibt.



Eingabe ist eine natürliche Zahl $n \geq 1$. Ist $n = 1$, so ist das auch das Ergebnis. Ist $n \neq 1$, so wird n halbiert, falls es gerade ist, oder sonst verdreifacht und um 1 erhöht. Mit dem jeweiligen Ergebnis (als neue Eingabe n) wird der Gesamtvorgang wiederholt. Für jede Zahl gibt es offenbar nur zwei Möglichkeiten: Entweder es kommt 1 als Ergebnis heraus oder der Berechnungsvorgang wird unendlich fortgeführt. Für welche Zahlen tritt welcher Fall ein? Zur Warnung sei gesagt, daß man sich jedoch nicht allzu sehr in die Beantwortung der Frage verbeißen sollte, weil sie seit Jahrzehnten ein offenes Problem ist.

1.2 Auf dem schnellsten Wege

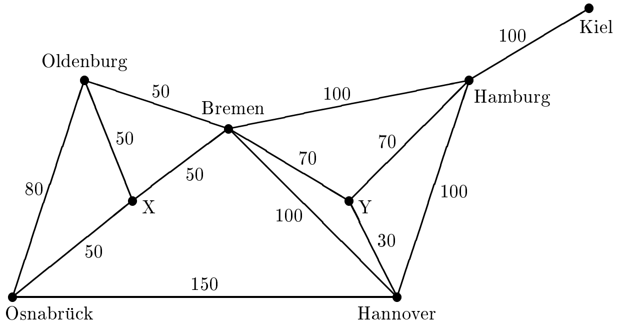
In einem Geldtransportunternehmen, das auch andere Wertsachen befördert, wird erwogen, die tägliche Fahrtroutenplanung zu automatisieren. Bei dem dafür zu erstellenden Softwaresystem muß von folgenden Vorgaben ausgegangen werden:

(a) Berechnungsgrundlage ist eine “Landkarte”, in der alle bundesdeutschen Ortschaften mit Geldinstituten als Knoten eingetragen sind. Je zwei Orte sind durch eine Kante miteinander verbunden, wenn es zwischen ihnen eine direkte (über keinen anderen Ort aus der Karte führende) Straßenverbindung gibt. Die Kanten sind mit der Kilometerzahl beschriftet.

(b) Die Eingabe besteht aus einer sich täglich ändernden Liste geplanter Fahrten, bei denen jeweils Start und Ziel angegeben sind. Aus Sicherheitsgründen werden keine Zwischenstationen gemacht und auch nicht mehrere Transporte zusammengefaßt.

(c) Bestimmt werden soll für jeden Eintrag der eingegebenen Liste der kürzeste Weg in der Landkarte, das ist eine Folge von Kanten, die Start und Ziel miteinander verbindet und deren Kilometersumme unter allen Verbindungswegen die kleinste ist.

Betrachte zur Illustration folgende Karte. Von Bremen nach Hannover gibt es viele Wege (sogar unendlich viele, wenn man erlaubt, beliebig oft hin und her oder im Kreis zu fahren), aber nur der direkte und der über Y sind mit 100 km die kürzesten.



Prinzipiell ist es nicht schwierig, einen Algorithmus zu entwerfen, der diese Aufgabe bewältigt; kritisch jedoch ist sein Laufzeitverhalten. Die Eingabeliste ist aus organisatorischen Gründen nicht vor 0.00 Uhr verfügbar, die ersten Fahrzeuge müssen jedoch spätestens um 6.00 Uhr morgens abfahren. Für die tägliche Einsatzplanung stehen also nur wenige Stunden zur Verfügung.

Das Transportunternehmen besitzt einen Matrixrechner, dessen Architektur insbesondere die Multiplikation von Matrizen unterstützt. Diese Operation kann deshalb besonders schnell ausgeführt werden. Die Automatisierung der Fahrtroutenplanung kommt aus den geschilderten zeitlichen Gründen nur dann in Frage, wenn das algorithmische Problem mit möglichst wenigen Matrizenmultiplikationen gelöst werden kann.

Ist das Vorhaben unter diesen Randbedingungen realisierbar?

1.3 Maschinenbelegung im Sonderangebot

Ein mittelständisches Unternehmen, das Spezialanfertigungen von elektronischen Geräten herstellt, möchte seine Produktion weiter automatisieren. Dazu benötigt es ein Maschinenbelegungsprogramm, das für täglich durchschnittlich 75 Aufträge mit unterschiedlichen Fertigungsplänen den Durchlauf durch 128 Fertigungsstationen organisiert, wobei ein einzelner Auftrag meist nur auf einem Teil der Stationen erledigt wird, das jedoch mit wechselnden Reihenfolgen und Verweilzeiten. Das Unternehmen schreibt ein derartiges Programm aus, wobei es neben den Kosten insbesondere Zuverlässigkeitsgarantien und Angaben über die Laufzeit in der Zeiteinheit μsec verlangt. Folgende Angebote gehen ein, wobei k die Zahl der Aufträge und m die Zahl der insgesamt eingesetzten Stationen ist.

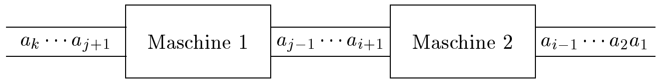
1. DM 100.000,–; optimale Maschinenbelegung; ungünstigste Laufzeit: m^k.
2. DM 200.000,–; optimale Maschinenbelegung; durchschnittliche Laufzeit: $k \cdot m!$.
3. DM 150.000,–; höchstens 100% über optimaler Maschinenbelegung; Laufzeit: $k^2 \cdot m^3$.
4. DM 120.000,–; höchstens das Dreifache des Optimums; Laufzeit: $128 \cdot k^2$.

5. Das Angebot 4 zum halben Preis, wenn die Zeiteinheit auf sec gesetzt werden darf.
6. DM 80.000,–; Optimum; Laufzeit: $c \cdot k$, wobei c zwischen 10.000 und 100.000 liegt. Allerdings muß dafür noch ein Zusatzprogramm für rund DM 10.000,– beschafft werden, das für einen Auftrag, der als nächstes auf mehreren Stationen gefertigt werden kann, die aktuell günstigste auswählt.
7. DM 250.000,–; Optimum; durchschnittliche Laufzeit: $k \cdot m^2$.
8. Das Angebot 7 zum doppelten Preis, dafür aber Laufzeit für den schlechtesten Fall.
9. DM 300.000,–; optimale Lösung mit 55% Garantie; Laufzeit: $128 \cdot k \cdot \log(k)$. (Anmerkung: Programm benötigt einen guten Zufallsgenerator.)
10. DM 75.000,–; bis 10% über optimaler Lösung; ungünstigste Laufzeit: $\binom{128}{m} \cdot \binom{m}{k}$.

Es gab noch einige weitere Angebote, die aber zu teuer, offenbar schlechter oder unseriös waren.

Welche Lösung soll das Unternehmen kaufen? Und warum? Wären alternative Angebote denkbar, die die zehn vorgestellten deutlich übertreffen?

Das Fließband, bei dem Aufträge in einer vorgegebenen Folge durch die festinstallierte Sequenz der Maschinen geschleust werden, hat – so scheint es – als Maschinenbelegungsplan ausgedient.

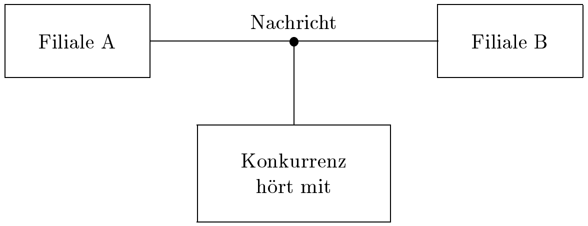


In der Illustration für zwei Maschinen werden gerade zwei Aufträge bearbeitet (a_j auf Maschine 1 und a_i auf Maschine 2). Was ist an einer Fließbandlösung für die Maschinenbelegung eigentlich schlecht?

1.4 Mit Netz und doppeltem Boden

Eine große internationale Bank mit einem weltumspannenden Netz von über 3000 Filialen möchte ihren Kundendienst verbessern. Dazu sollen alle Zweig- und Geschäftsstellen in ein Kommunikationssystem einbezogen werden, das je zwei Filialen erlaubt, in wenigen Minuten untereinander Informationen auszutauschen. Kostengünstig läßt sich das jedoch nur bewerkstelligen, wenn in Kauf genommen wird, daß Unbefugte ohne großen technischen Aufwand übermittelte Nachrichten auffangen können. Um das Bankgeheimnis zu wahren und mißbräuchliche Verwendung der Informationen zu verhindern, ist eine Verschlüsselung unumgänglich.

Unrealistisch wäre ein Konzept, in dem je zwei Filialen für ihre Kommunikation einen eigenen geheimen Schlüssel hätten. Dafür wären 10.000.000 Schlüssel erforderlich. Auch die jeweilige Vergabe eines Schlüssels durch eine Zentrale, wenn zwei Filialen Verbindung aufnehmen wollen, ist umständlich und schwierig, zumal auch die Übersendung der



Schlüssel verschlüsselt werden müßte. Da behauptet eine übereifrige Vorstandsassistentin, daß sie einen Artikel gelesen hätte über die Verwendung "öffentlicher" Schlüssel: Jede Filiale bildet nach einem festen, einfachen Prinzip aus nur ihr bekannten Bestandteilen einen Schlüssel, der in eine allen Zweigstellen zugängliche Liste verzeichnet wird. Will nun Filiale A Filiale B eine Mitteilung machen, sucht sie den Schlüssel von B aus dem Verzeichnis heraus, chiffriert nach einem bestimmten Verfahren die Nachricht und sendet sie an B. Die Zweigstelle B entschlüsselt diese Nachricht mit Hilfe der nur ihr bekannten Bestandteile ihres Schlüssels nach einem allen bekannten Verfahren. Ohne die Bestandteile zu kennen, ist die Dechiffrierung jedoch praktisch unmöglich.

Will sich die Vorstandsassistentin nur wichtig machen? Oder sind nicht knackbare Codes möglich, selbst wenn die verwendeten Verfahren für Ver- und Entschlüsseln sowie die Schlüssel allgemein bekannt sind?

1.5 Geheimnisvoller Auftrag

Ein Geheimdienst hört seit Monaten einen gegnerischen Sender ab. Die aufgefangenen Buchstaben-, Zahlen-, Silben- und Wörterfolgen sind jedoch bisher nicht entschlüsselt.

Die erste wichtige Beobachtung ist, daß jede Viertelstunde Informationen von besonderer Form übermittelt werden. Es handelt sich immer um zwei gleichlange Sequenzen von Zeichenfolgen, die durch Hupsignale voneinander getrennt sind. Wenn das Hupen durch einen Gedankenstrich symbolisiert wird, haben die Nachrichten folgendes Aussehen:

$$- - u_1 - u_2 - \cdots - u_k - - v_1 - v_2 - \cdots - v_k - -$$

Dabei sind u_i, v_j irgendwelche Zeichenfolgen aus Buchstaben und Ziffern. Ein Beispiel dafür ist:

$$- - aac - a - bb - aac - b - - a - baa - acb - ca - acb - -$$

Die DechiffrierexpertInnen hatten schon längere Zeit vermutet, daß sich in diesen Doppelsequenzen der Schlüssel für die Entschlüsselung aller aufgefangenen Sendungen befindet. Aber wie?

In dieser Situation kam dem Geheimdienst der Zufall zu Hilfe. Ein gegnerischer Agent ging ihm ins Netz und verriet in langen Verhören so manches Geheimnis. Insbesondere

bestätigte er, daß sich unter den viertelstündlich gesendeten Folgen u_i und v_j für $i, j = 1, \ldots, k$ der Schlüssel befindet – jedoch nur einmal alle zwei bis drei Tage. Alle anderen Doppelsequenzen sind Tarnung. Die einzige signifikante Doppelsequenz unterscheidet sich von den anderen dadurch, daß es für sie allein keine Indexfolge $i_1 \cdots i_n$ gibt, derart daß die zusammengesetzten Zeichenketten $u_{i_1} u_{i_2} \cdots u_{i_n}$ und $v_{i_1} v_{i_2} \cdots v_{i_n}$ gleich sind. Diese Figuren können nach der Art ihres Zustandekommens *Zwillingspuzzles* genannt werden.

Der Rest schien einfach. Der EDV-Abteilung wurde der Auftrag erteilt, ein Programm zu entwickeln, das für eine eingegebene Doppelfolge testet, ob sie ein Zwillingspuzzle ergibt. Doch die SystemanalytikerInnen und ProgrammiererInnen des Geheimdienstes brachten ein solches Programm nicht zustande. Deshalb soll die Aufgabe einem Softwarehaus übertragen werden, das bei erfolgreicher Erledigung des Auftrags pauschal 120.000,– DM erhält.

Ein gutes Geschäft? Was ändert sich an der Situation, wenn der jeweils aktuelle Schlüssel in einer Doppelsequenz mit Zwillingspuzzle versteckt wäre und die Tarnsequenzen keine Zwillingspuzzles haben?

1.6 Die Antworten der Theorie

Was haben diese (und ähnliche) Situationen und Problemstellungen mit theoretischer Informatik zu tun? – Die Theorie beantwortet die gestellten Fragen. Aber das allein wäre nichts Besonderes; denn solche Fragen treten häufig bei Softwarentwicklungen auf und werden auch irgendwie beantwortet. Jedoch nur die Theorie gibt *exakte* Antworten (das sind Antworten, die in einem bestimmten Kontext zweifelsfrei und nachweisbar gültig sind). Etwas vereinfachend könnte man sogar sagen: Theoretische Informatik ist gerade dadurch definiert, daß derartige Fragen in einem formalen Rahmen mit mathematischen Methoden und daher exakt beantwortet werden. Wann immer und wo immer in der Informatik so vorgegangen wird, entsteht theoretische Informatik.

Theoretische Informatik ist Informatik mit mathematischen Mitteln.

Die Szenarien münden in Fragen, die ein gemeinsames Grundmuster besitzen: Geht das und das (oder lohnt es sich) mit den und den Methoden unter den und den Bedingungen? Es sollte klar sein, daß in der Informatik (und in manchen anderen Wissenschaften) an vielen Stellen so gefragt werden kann. Es sollte auch klar sein, daß exakte Antworten nicht nur von wissenschaftlichem, sondern auch ökonomischem Wert sein können. Denkt man an computer-gesteuerte Kernkraftwerke, Flugzeuge, Frühwarnsysteme und ähnlich brisante Aufgaben, die mit Hilfe von Computertechnik und Informatikwissen erledigt werden, können (un-)gesicherte Kenntnisse offenbar über Leben und Tod entscheiden. In solchen lebenswichtigen Fragen sind jedoch leider die Antworten der Theorie eher deprimierend. Wegen der technischen Komponenten, in die die Berechnungsprobleme lediglich eingebettet sind (Stichwort: embedded system), sind oft nur statistische Aussagen möglich.

Beispielsweise wird sich unter "geeigneten" Annahmen vielleicht herausstellen, daß das russische Frühwarnsystem in einer Million Jahren wahrscheinlich nur 17mal fälschlich einen Atomangriff von Westeuropa aus meldet, ohne daß der Fehler vor Auslösen des Gegenschlags festgestellt wird. Doch die Theorie könnte nicht verhindern, daß die erste dieser Fehlmeldungen bereits morgen erfolgt und das atomare Inferno in Europa auslöst.

1.7 Die Schwierigkeiten der Theorie

Wenn Theorie so wichtig und wertvoll ist wie behauptet, warum ist sie dann in der Informatik und bei vielen Informatikerinnen und Informatikern so umstritten und unbeliebt, ja teilweise verpönt? Ein Grund dafür ist wahrscheinlich, daß es oft nicht einfach ist, grundlegende Fragen der geschilderten Art korrekt zu beantworten. Theorie ist kein Frage-Antwort-System, in das man eine Frage nur hineinstecken muß, um nach einiger Zeit automatisch die richtige Antwort zu erhalten. Oft müssen Fragen und Antworten in einem vorhandenen Rahmen entwickelt werden. Oft fehlt der geeignete theoretische Rahmen und muß erst noch geschaffen werden. Oft sind zwar Rahmen, Frage und Antwort prinzipiell bereits verfügbar, doch ist das schwer zu erkennen, weil die vorgefertigte Theorie anders formuliert ist, als es das aktuelle Problem erfordert. Meist sind deshalb Zeit, Phantasie, Intuition, Wissen und Erfahrung nötig, um existierende Theorie nutzbringend zu verwenden und fehlende zu ergänzen.

Um ein Problem exakt lösen zu können, muß schrittweise und systematisch eine Theorie aufgebaut werden. Die einzelnen Überlegungen müssen aber nicht nur verstanden werden, sondern auch nachprüfbar richtig sein. Das macht Theoriebildung zu einem langsamen, mühsamen Unternehmen. Wie umständlich dieser Vorgang sein kann, mag folgender Zeitplan verdeutlichen: Das Thema dieser Lehrveranstaltung ist Berechenbarkeit und Komplexität; alle Szenarien haben direkt und zentral mit diesem Thema zu tun; dennoch wird nur eins der Probleme in diesem Semester gelöst (vielleicht noch ein zweites).

> Theorie ist ein – häufig aufwendiger – sprachlicher Rahmen,
> in dem grundsätzliche Probleme exakt gelöst werden können.

1.8 Das Prinzip der Abstraktion

Damit der hohe Aufwand der Theoriebildung gerechtfertigt ist, werden Theorien meist so allgemein und breit entwickelt, daß möglichst viele Probleme damit gelöst werden können. In der Regel haben sogar mehrere verschiedene konkrete Probleme innerhalb einer entsprechenden Theorie eine gemeinsame Lösung. Erreicht wird dieser "ökonomische" Effekt durch einen Abstraktionsprozeß, dem die konkreten Probleme unterzogen werden, bevor sie im Rahmen einer Theorie behandelt werden. Abstrahieren heißt, von unwichtigen Details abzusehen und sich auf das Wesentliche zu konzentrieren. Beispielsweise ist im fünften Szenario die "aufregende" Spionagegeschichte für das Problem völlig irrelevant; interessant ist allein, wie festgestellt werden kann, ob eine Doppelsequenz aus Zeichenketten

ein Zwillingspuzzle besitzt. Oder im vierten Szenario könnte es sich statt um eine Bank auch um die Mafia, den CIA, die Botschaften eines Staates u.ä. handeln, statt um über 3000 Filialen, die miteinander kommunizieren wollen, auch um 428 Mafia-Organisationen, um 7000 AgentInnen, um 113 diplomatische Vertretungen u.ä. In einer Theorie, in der das Problem der öffentlichen Schlüssel diskutiert werden kann, wird also nicht definiert werden, was eine Bank ist und wie viele Filialen sie hat; davon wird abstrahiert. Stattdessen wird man von der Annahme ausgehen, daß es n Instanzen gibt, wobei n eine beliebige natürliche Zahl ist, und daß je zwei von ihnen Verbindung miteinander aufnehmen können sollen.

Das Prinzip der Abstraktion hat jedoch seine Tücken. Die ursprünglichen konkreten Probleme sind in der abstrakten Theorie nicht ohne weiteres erkennbar. Beim Abstrahieren kann es passieren, daß ein entscheidender Aspekt vernachlässigt wird; dann spiegeln die theoretischen Ergebnisse die wirkliche Situation nicht mehr adäquat wider. Zu einem konkreten Problem mag es längst eine geeignete Theorie geben, die die fertige Antwort parat hält; aber das nutzt nichts, wenn diejenigen, die das Problem haben, die Theorie nicht kennen oder ihnen die richtige Abstraktion nicht gelingt. Ist es da verwunderlich, daß viele Theorien nutzlos und wie abstrakte Spinnerei erscheinen?

1.9 Probleme – unberechenbar

Als allgemeine Vorbemerkungen zur Theorie mag das erst einmal ausreichen. Aber worum geht es nun speziell in dieser Lehrveranstaltung?

Informatik löst spezifische Aufgaben mit Hilfe von Rechenanlagen, untersucht die Gesetzmäßigkeit eines Lösungsvorgangs und stellt Methoden für Lösungsschritte bereit. Der Schlüssel zu all dem ist der Begriff des Algorithmus, worunter eine maschinell ausführbare Lösung eines vorgegebenen Problems verstanden wird. Die theoretische Informatik verfolgt dieselben Ziele, soweit sie sich mathematisch untermauern lassen. Im Zentrum stehen dabei einige grundsätzliche Fragen, was beispielsweise ein Computer berechnen kann oder welche Probleme algorithmisch lösbar sind.

Beide Fragen sind so noch sehr unscharf formuliert. Denn es gibt ja viele verschiedene Rechnertypen mit verschiedenen Ausstattungen. Man müßte also genauer fragen: Was kann der Computer X, was der Computer Y? Aus praktischer Sicht wäre es fatal, wenn für je zwei Rechner Unterschiedliches und Unvergleichbares herauskäme. Dann wüßte niemand, der ein Problem nicht lösen kann, ob er nicht einfach an der falschen Maschine sitzt. In der zweiten Frage wird unterstellt, daß es nur *ein* "algorithmisch" gibt. Tatsächlich könnte aber die Lösungsfähigkeit und -vielfalt von der gewählten (Programmier-)Sprache oder von anderen Faktoren abhängen. Gibt es also ein "Modula-algorithmisch", das grundverschieden ist von "ML-algorithmisch"? Können in C++ völlig andere Probleme gelöst werden als mit SmallTalk? Bejahende Antworten hätten die unangenehme Konsequenz, daß allein schon die Wahl der Programmiersprache über Erfolg oder Mißerfolg eines Problemlösungsversuchs entscheiden könnte. Schließlich könnten sich auch die Möglichkeiten von Rechner und Problemlösungssprache in die Quere kommen. Denn was nützt die beste Sprache, wenn kaum eine Lösung darin auf dem vorhandenen Rechner laufen kann? Was bringt der phantastischste Computer, wenn die Programmiersprache seine Rechenfähigkeit nicht voll ausschöpfen kann?

Die *Berechenbarkeitstheorie* bietet eine These an, deren Gültigkeit viele der gerade angenommenen Schwierigkeiten beseitigte. Die These wurde in den 40er Jahren von A. Church formuliert und lautet:

> Alle genügend allgemeinen Berechnungsmodelle erlauben, dieselbe Klasse von Problemen zu lösen.

In der Theorie der Berechenbarkeit werden vielfältige Belege zusammengetragen, die die CHURCHsche These stützen. Ein repräsentativer Ausschnitt davon wird in dieser Lehrveranstaltung entwickelt.

Überlegungen und Ansätze zur Berechenbarkeit lassen sich in zwei Klassen einteilen. Eine davon bilden die Rechner- oder Maschinenmodelle wie Turingmaschinen und Random-Access-Maschinen, die als abstrakte Versionen von Computern gesehen werden können. Mathematische Modelle, die stellvertretend für Rechner untersucht werden, haben den Vorteil, nicht so schnell zu veralten und vom Markt zu verschwinden. Ihr Nachteil allerdings besteht darin, daß oft viel Phantasie erforderlich ist, ihre Verwandtschaft mit konkreten Rechnern zu erkennen. Jedes Maschinenmodell MM definiert eine Klasse der von MM berechenbaren Probleme. Die zweite tragende Säule der Berechenbarkeitstheorie beschäftigt sich mehr mit der Problemseite. Untersucht werden rechnerunabhängige Beschreibungen von algorithmischen Problemlösungen, z.B. als rekursive oder iterative Funktionen. Insgesamt wird eine Vielzahl von Berechnungsmodellen eingeführt; jedes einzelne stellt eine Präzisierung des Begriffs "Algorithmus" dar und bestimmt eine Klasse lösbarer Probleme.

Ein wichtiger Teil der Berechenbarkeitstheorie ist dem Vorhaben gewidmet, die Übereinstimmung der verschiedenen Problemklassen im Sinne der CHURCHschen These zu zeigen. In dieser Lehrveranstaltung wird beispielsweise vorgeführt, daß die von Turingmaschinen berechenbaren Probleme gerade mit den rekursiven Funktionen übereinstimmen.

Berechenbarkeit teilt die "Welt" in zwei Teile: das Berechenbare und das Unberechenbare. Beispiele, die zur ersten Kategorie gehören, entwickelt jede Informatikerin und jeder Informatiker in der täglichen Arbeit. Aber es kann auch von Nutzen sein, etwas über das mit Rechnern nicht Leistbare zu wissen. Das ist nicht allein von philosophischem Belang, sondern kann praktisch auch viel unnütze, vergebliche Anstrengung vermeiden helfen. Einige Exemplare nicht berechenbarer Probleme werden in dieser Lehrveranstaltung diskutiert. Nicht geklärt werden kann, was PolitikerInnen meinen, wenn sie sagen: "Unsere Politik muß berechenbar bleiben!"

1.10 Schnelles Rechnen – viel zu langsam

Es ist beruhigend, daß ein einziger Algorithmusbegriff prinzipiell auszureichen scheint, um das mit Rechnern Machbare untersuchen zu können. In vielen Fällen jedoch ist das Wissen um die Berechenbarkeit nicht sehr hilfreich. Zu diversen Problemen findet man mühelos Algorithmen, doch die Programme laufen nicht zufriedenstellend. Ein Grund dafür ist, daß Turingmaschinen beispielsweise (als Vertreterinnen eines allgemeinen Algorithmus- und Rechnerkonzepts) mit zwei bei wirklichen Computern knappen Ressourcen äußerst verschwenderisch umgehen: Platz und Zeit. Beides steht einer Turingmaschine unbegrenzt zur Verfügung. Vielleicht gilt das prinzipiell sogar für reale Maschinen (die Physik legt sich in diesen Punkten nicht ganz fest), de facto aber können wir nicht immer genügend Speichermedien heranschaffen und schon gar nicht lange warten.

Ein konkretes Beispiel soll die Rolle der Zeit beim Rechnen veranschaulichen. (Analoges läßt sich für den Speicherplatz überlegen.) Computer sind berühmt dafür, daß sie stupideste Aufgaben mit umfangreichem Informationsmaterial in unglaublich kurzer Zeit erledigen können. Aber selbst dabei stoßen sie schnell an ihre Grenzen. Betrachte etwa folgendes simple Problem: Eingegeben werden soll eine beliebige natürliche Zahl $n \in \mathbb{N}$, ausgegeben werden soll eine Liste aller Teilmengen der Menge $[n] = \{0, 1, \ldots, n-1\}$ der ersten n natürlichen Zahlen, d.h. eine Liste der Elemente der Potenzmenge $\mathcal{P}([n])$. Nun hat diese Potenzmenge bekanntlich 2^n Elemente. Die Länge der Ausgabeliste wächst also exponentiell mit der Größe der Eingabe. Einen Eindruck vom schnellen Anwachsen der Funktion 2^n gegenüber der Funktion n^2 vermittelt die folgende Tabelle:

n	n^2	2^n
10	100	1024
20	400	1048576
30	900	1073741824
40	1600	1099511627776
50	2500	1125899906842624
60	3600	1152921504606846976
70	4900	1180591620717411303424

(ohne Gewähr)

Es ist nicht sonderlich schwierig, einen Algorithmus zu finden, der die Aufgabe löst. Aber selbst wenn der Rechner nur eine Instruktion bräuchte, um eine Teilmenge zu bestimmen und aufzulisten, und 25 Millionen Instruktionen pro Sekunde schaffte (ein Supercomputer) und wenn ich mich nicht verrechnet habe, rechnete das Gerät bei Eingabe von 70 nach einer Million Jahren immer noch.

Das theoretische Gebiet, das sich mit dem Zeit- und Platzbedarf von Algorithmen beschäftigt, ist die *Komplexitätstheorie*. In dieser Lehrveranstaltung werden einige in diesen Bereich einführende Aspekte behandelt.