Complexity and Real Computation: A Manifesto *

LENORE BLUM[†] FELIPE CUCKER[‡||]

MIKE SHUB[§||] STEVE SMALE[¶||]

TR-95-042

Abstract. Finding a natural meeting ground between the highly developed complexity theory of computer science —with its historical roots in logic and the discrete mathematics of the integers— and the traditional domain of real computation, the more eclectic less foundational field of numerical analysis —with its rich history and longstanding traditions in the continuous mathematics of analysis— presents a compelling challenge. Here we illustrate the issues and pose our perspective toward resolution.

This article is essentially the introduction of a book with the same title (to be published by Springer) to appear shortly.

*Webster: A public declaration of intentions, motives, or views.

||Partially supported by NSF grants.

[†]International Computer Science Institute, 1947 Center St., Berkeley, CA 94704, U.S.A., lblum@icsi.berkeley.edu. Partially supported by the Letts-Villard Chair at Mills College.

[‡]Universitat Pompeu Fabra, Balmes 132, Barcelona 08008, SPAIN, cucker@upf.es. Partially supported by DGICyT PB 920498, the ESPRIT BRA Program of the EC under contracts no. 7141 and 8556, projects ALCOM II and NeuroCOLT.

[§]IBM T.J. Watson Research Center, Yorktown Heights, NY 10598-0218, U.S.A., shub@watson.ibm.com.

[¶]Dept. of Mathematics, Univ. of California, Berkeley, CA 94720, U.S.A., smale@math.berkeley.edu.

1 Aim

The classical theory of computation had its origins in work of logicians —of Gödel, Turing, Church, Kleene, Post, among others— in the 1930's. The model of computation that developed in the following decades, the Turing machine, has been extraordinarily successful in giving the foundations and framework for theoretical computer science.

Our point of view is that the Turing model (we will call it "classical") with its dependence on 0's and 1's is fundamentally inadequate for giving such a foundation to the theory of modern scientific computation, where most of the algorithms —with origins in Newton, Euler, Gauss, et al.— are *real number algorithms*. Our viewpoint is not new. Already in 1948, John von Neumann, in his Hixon Symposium lecture, articulated the need for "a detailed, highly mathematical, and more specifically analytical theory of automata and of information". In this lecture, von Neumann was particularly critical of the limitations imposed on the "theory of automata" by its foundations in formal logic:

> There exists today a very elaborate system of formal logic, and specifically, of logic as applied to mathematics. This is a discipline with many good sides, but also serious weaknesses. . .. Everybody who has worked in formal logic will confirm that it is one of the technically most refractory parts of mathematics. The reason for this is that it deals with rigid, all-or-none concepts, and has very little contact with the continuous concept of the real or of the complex number, that is, with mathematical analysis. Yet analysis is the technically most successful and best-elaborated part of mathematics. Thus formal logic, by the nature of its approach, is cut off from the best cultivated portions of mathematics, and forced onto the most difficult part of the mathematical terrain into combinatorics.
>
> The theory of automata, of the digital, all-or-none type as discussed up to now, is certainly a chapter in formal logic. It would, therefore, seem that it will have to share this unattractive property of formal logic. It will have to be, from the mathematical point of view, combinatorial rather than analytical.

We propose a formal theory of computation which integrates major themes of the classical theory and builds on the classical foundations, yet at the same time is more mathematical, perhaps less dependent on logic, and more directly applicable to problems in mathematics, numerical analysis and scientific computing.

We propose a theory of *real computation*.

We propose to do this in a way which preserves the Turing theory as a special case of the new theory, i.e. by an appropriate choice of the fields admitted. In this way, results from computer science give insight to numerical analysis and the reverse holds as well.

2 Seven Examples

2.1 The Mandelbrot Set
2.2 A Julia Set
2.3 Newton's Method
2.4 The Knapsack Problem
2.5 The Hilbert Nullstellensatz as a Decision Problem
2.6 4-Feasibility
2.7 Linear Programming, Integer Programming.

2.1 Is the Mandelbrot Set Decidable?

The British mathematical physicist, Roger Penrose, in a popular book, "The Emperor's New Mind", p. 124, writes:

> Now we witnessed... a certain extraordinarily complicated looking set, namely the Mandelbrot set [(Figure 1]. Although the rules which provide its definition are surprisingly simple, the set itself exhibits an endless variety of highly elaborate structures.

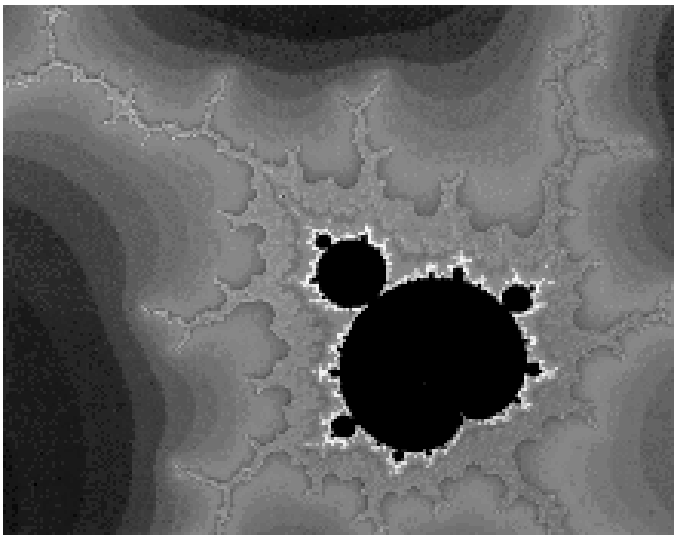

Figure 1 The Mandelbrot set

> Could this be an example of a non-recursive [i.e., undecidable] set, truly exhibited before our mortal eyes?

It is known that the boundary of the Mandelbrot set has a rich and complex structure. (See for example the next figure where a part of this boundary is shown.) Hence Penrose's query seems reasonable.

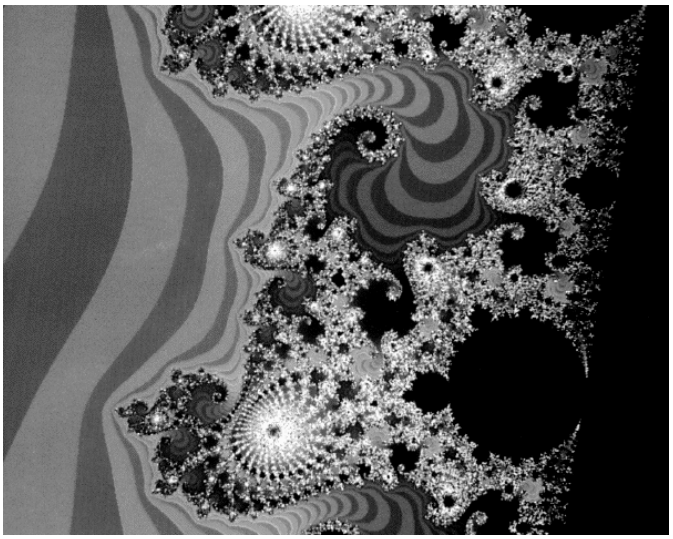

Figure 2 On the boundary: Seahorse Valley

Penrose is motivated to ask this question to make an argument against artificial intelligence. While we find this use of mathematics not compelling, the question of the decidability of the Mandelbrot set has another justification. It can partly answer and give insight to the question: Can one decide if a differential equation is chaotic?

The Mandelbrot set $\mathcal{M}$ is defined as the set of complex numbers c such that the sequence $c, c^2 + c, (c^2 + c)^2 + c, \ldots$ remains bounded.

More formally, for $c \in \mathbb{C}$, the complex numbers, let $p_c(z) = z^2 + c$ and let $p_c^n(z)$ be the nth iterate of p_c applied to z. That is,

$$p_c^n(z) = p_c(\ldots p_c(p_c(p_c(z)))), \; n \text{ times.}$$

So $p_c(0) = c$, $p_c^2(0) = c^2 + c, \ldots$. Then $\mathcal{M}$ is the complement of the set

$$\mathcal{M}' = \{c \in \mathbb{C} \mid p_c^n(0) \to \infty \text{ as } n \to \infty\}.$$

The set $\mathcal{M}$ may also be described as the set of all inputs c which don't halt for the flow chart in Figure 3.

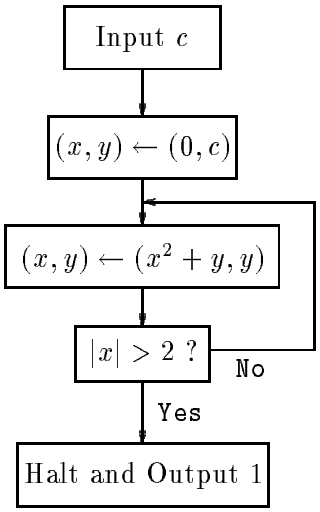

Figure 3 A flow chart associated with the Mandelbrot set

This is because if ever the sequence $c, c^2 + c, (c^2 + c)^2 + c$ escapes the disk of radius 2, it will go off to infinity.[1]

To answer Penrose's query, one needs a "machine" or "algorithm" that, given input c, a complex number, will decide in a finite number of steps whether or not c is in $\mathcal{M}$. (See Figure 4.)

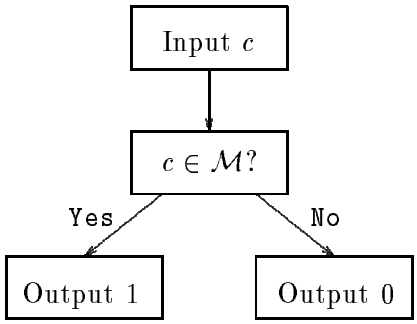

Figure 4 Desired decision machine for $\mathcal{M}$

[1] This fact is utilized in designing computer algorithms for drawing "pictures" of the Mandelbrot set: Let N be a large integer. For given point c, generate up to N elements of the sequence $c, c^2 + c, (c^2 + c)^2 + c, \ldots$ along with their magnitudes. If and when some magnitude is greater than 2, color c white, else color c black. Note that white points are *definitely* in $\mathcal{M}'$ while black points are *possibly* in $\mathcal{M}$ with our confidence level partly dependent on N. (For more sophisticated algorithms, see the book [Peitgen and Saupe 1988]).

After asking his question, Penrose acknowledges being somewhat inexact (p. 125). The classical theory of computation presupposes that all the underlying sets are countable and hence ipso facto cannot handle these questions about subsets which are uncountable.

Next, Penrose seeks ways to bypass this problem. One way is to use computable real numbers (to describe the appropriate complex numbers). This would be the approach of *recursive analysis*, an area originating with early work of Turing. Here one might imagine a Turing machine being input a real number bit by bit.

Using its internal instructions, the machine operates on what it sees, possibly every so often outputting a bit. The resulting sequence, if any, would be considered in the limit the (binary expansion of the) real output.

Problems arise here when one wants to decide if two numbers are equal and so Penrose rejects this approach. As he points out on p. 126, "One implication of this is that even with such a simple set as the unit disc, ..., there would be no algorithm for deciding for sure ... whether the computable number $x^2 + y^2$ is actually equal to 1 or not, this being the criterion for deciding whether or not the computable complex number $x + iy$ lies on the unit circle. ... Clearly that is not what we want."

Another tack might be to consider the rational or algebraic skeleton of the problem. Thus, we could rephrase Penrose's question: given a complex number c whose real and imaginary parts are rational, or algebraic, decide whether or not c is in $\mathcal{M}$. Indeed, this has been a tack used by theoretical computer scientists to deal with problems whose natural underlying spaces are the real numbers (such as the linear programming problem) or the complex numbers. However, this approach is also problematical. For example, the curve $x^3 + y^3 = 1$ has no rational points with both x and y positive. So, the rational skeleton provides no useful information about the given curve.

After exploring several such approaches, Penrose p. 129 concludes: "one is left with the strong feeling that the correct viewpoint has not yet been arrived at."

Thus Penrose's question, "Is the Mandelbrot set decidable?" makes no sense!

Now note that the flow chart of Figure 3 could be interpreted as a machine with "halting set" which is precisely the complement $\mathcal{M}'$ of $\mathcal{M}$. (The set $\mathcal{M}'$ might be said to be "semi-decidable".) This machine has the power to accept complex numbers, perform basic arithmetic operations on complex numbers and to compare magnitudes. It is an example of a machine (to be defined formally later) over the real numbers $\mathbb{R}$ *not* $\mathbb{C}$ since it uses the real comparison $|z| > 2$.

What is not clear but is true, is whether there is not a similar kind of machine with $\mathcal{M}$ as its halting set. The theory of real computation proposed makes precise and formal some of the suggestions here. In this theory, Penrose's question becomes well-defined and we answer it: the Mandelbrot set is *not* decidable over $\mathbb{R}$.

2.2 Example of the Julia Set of $T(z) = z^2 + 4$

We are looking at a polynomial map $T : \mathbb{C} \to \mathbb{C}$ from the point of view of iteration or as a complex dynamical system. Thus we write $T^2(z) = T(T(z))$ and T^k for the composition of T with itself k times. Let us specify $T(z) = z^2 + 4$.

Observe that if $|z| > 2$, $|T^k(z)| \to \infty$ as $k \to \infty$.

Consider the flow chart in Figure 5.

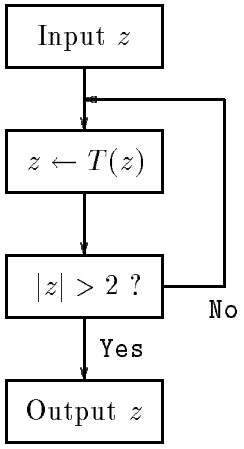

Figure 5 A Julia set flow chart

Call this machine M. In M there are 4 nodes (the boxes) which are called as we descend in the diagram: input node, computation node, branch node and output node. Again we have an example of a machine over the real numbers $\mathbb{R}$ since its branching depends on real inequality comparisons.

The halting set Ω_M of M is the set of inputs $z \in \mathbb{C}$ such that by following the flow of the flow chart, we eventually halt (or output). For example Ω_M contains the set of all z with $|z| > 2$. Moreover $0, \pm 1, \pm 2$ are all in Ω_M. However fixed points of T, (so $T(z) = z$, or $z^2 + 4 = z$) are not in the halting set. In fact any periodic point of T (so $T^k(z) = z$ some $k = 1, 2, 3, \ldots$) is not in Ω_M.

A little thought will show that Ω_M is an open set of complex numbers, so that $J = \mathbb{C} - \Omega_M$ must contain the closure of the set of periodic points of T. J is the Julia set of T in the terminology of complex dynamical systems and it can be proved that J is the closure of the set of periodic points of T and is homeomorphic to a Cantor set.

A question again suggested by the classical theory of computation is: Is J decidable, or equivalently, is there a real machine with halting set J?

To see the equivalence of these questions, note that a decision machine for J can be converted into a machine with halting set J as indicated in Figure 6.

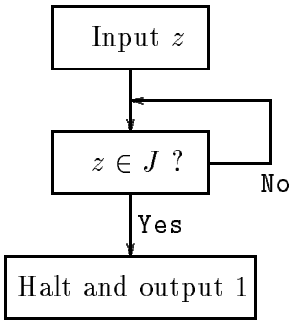

Figure 6 J is a halting set (supposing J is decidable)

On the other hand, suppose J were the halting set of some machine M_J. Recall, the flow chart machine M given in Figure 6 has halting set $\mathbb{C} - J$. We construct a decision machine for J by hooking together M and M_J (see Figure 7):

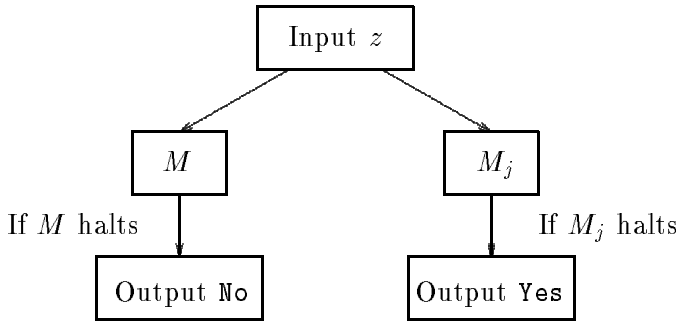

Figure 7 J is decidable (supposing J is a halting set)

To decide if $z \in J$, input z into both M and M_J and run the two machines in tandem. One and only one of these machines will halt, the one that does decides the membership of z. (Schematically, we have indicated a parallel process. This could be turned into a sequential process by alternating in turn between operations of M and M_J.)

With a formal development of machines over $\mathbb{R}$, one can answer the above questions (the answer is, J is not a halting set and hence not decidable).

2.3 Newton's Method

The previous two examples raised questions concerning the existence of machines that would *decide* Yes or No to queries of the form: Given $z \in \mathbb{C}$, is $z \in \mathcal{M}$ (or J)?

On the other hand, often we want algorithms that *search* for solutions to problems of the form: Given a polynomial f, find ζ such that $f(\zeta) = 0$.

Newton's method is the "search algorithm" sine qua non of numerical analysis and scientific computation. Here we briefly recall Newton's method for finding (approximate) zeros of polynomials in one variable.

Given a one variable polynomial $f(z)$ over the complex numbers $\mathbb{C}$, define the *Newton endomorphism* $N_f : \mathbb{C} \to \mathbb{C}$ by:

$$N_f(z) = z - \frac{f(z)}{f'(z)}. \tag{1}$$

This map is defined as long as $f'(z) \neq 0$.

Now for Newton's method: Pick an initial point $z_0 \in \mathbb{C}$ and generate the *orbit*

$$z_0, z_1 = N_f(z_0), z_2 = N_f(z_1), \ldots, z_{k+1} = N_f(z_k) = N_f^{k+1}(z_0), \ldots$$

Some stopping rule such as "stop if $|f(z_k)| < \varepsilon$ and output z_k" is given. In practice, if the procedure has not stopped with an output after a certain number of iterates, or it becomes undefined at some stage, a new initial point is chosen.

We may represent Newton's method schematically as in Figure 8. In this simple machine, we assume that f, N_f and ε are built in. An initial point $z \in \mathbb{C}$ is "input" to the machine. Later we will consider machines which allow f and ε to be input as well.

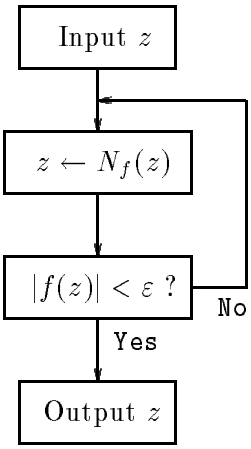

Figure 8 The Newton machine for f

Proposition 1 (a) $f(\zeta) = 0$ *if and only if* $N_f(\zeta) = \zeta$ *and* (b) $N_f(\zeta) = \zeta$ *implies* $|N_f'(\zeta)| < 1$.

Part (a) of the Proposition follows directly from the formula defining N_f (and noting that there are polynomials g and h such that $\frac{f}{f'} = \frac{g}{h}$ and if $f(\zeta) = 0$, then $g(\zeta) = 0$ but $h(\zeta) \neq 0$). To show (b) we observe that, for ζ a zero of f of multiplicity m,

$$|N_f'(\zeta)| = \frac{m-1}{m}.$$

To see this, note that $N_f' = \frac{ff''}{(f')^2}$ and evaluate it using the Taylor expansion of f about ζ,

$$f(z) = a_m(z-\zeta)^m + \text{ higher order terms} \quad a_m \neq 0.$$

Thus the *zeros* of f are the *fixed points* of N_f and the fixed points of N_f are *attracting*. This implies there are local neighborhoods about the zeros of f that contract under (iterates of) N_f. Hence, any point $z \in \mathbb{C}$ that, under the action of N_f, eventually enters one of these contracting neighborhoods will eventually approach a zero of f. This is the basis of Newton's method.

If ζ is a simple zero of f, then $N_f'(\zeta) = 0$, i.e. ζ is a *superattracting* fixed point of N_f. It follows that there is an open set of points whose orbits under the Newton endomorphism eventually converge *quadratically* to ζ. That is, starting at any of the points in this open set, Newton's method will eventually converge to ζ, and moreover, at some stage, the precision of the approximation will double with each successive iteration.

Decidability questions also arise in this context. It is well known that Newton's method is not *generally convergent*. The main obstruction to general convergence is the existence of attracting periodic points of period at least 2. For example, consider the cubic polynomial

$$f(z) = z^3 - 2z + 2.$$

Here $N_f(z) = z - \frac{z^3-2z+2}{3z^2-2}$. So $N_f(0) = 1$ and $N_f(1) = 0$, i.e. 0 is a point of period 2 under the Newton endomorphism. Also, by the chain rule we see, $(N_f^2)'(0) = 0$. So there is a neighborhood of points about 0 whose orbits under the Newton map fail to converge to a zero of f. These points are *super*attracted to the periodic orbit: $0, 1, 0, 1, \ldots$. So, given f, it is natural to ask: Is the set of "good" starting points for Newton's method decidable?

Here we will say that a point $z \in \mathbb{C}$ is good if its orbit under N_f converges to a zero of f. It can be shown that the set of good starting points coincides with the halting set of some Newton machine M (with built in ε that depends on f). So as before we can rephrase our question: Is the set of bad points a halting set? With a theory of real machines doing exact arithmetic it can be shown that the answer is no. We include here a picture indicating the good and bad points for Newton's method applied to the polynomial $f(z) = (z^2-1)(z^2+0.16)$. In this picture (see Figure 9), bad points are colored black and the set of them includes the Julia set for the Newton endomorphism.

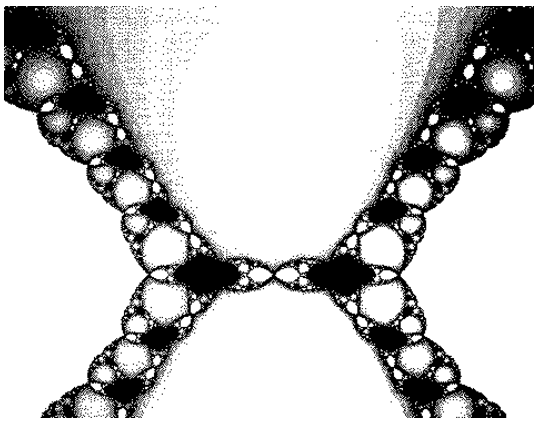

Figure 9 The dynamics of the Newton endomorphism for $f(z) = (z^2 - 1)(z^2 + 0.16)$

2.4 The Knapsack Problem

Now suppose R is a commutative ring with unit. For specificity, one may suppose R is the ring of integers $\mathbb{Z}$, the rationals $\mathbb{Q}$, the reals $\mathbb{R}$, or the complex numbers $\mathbb{C}$. Consider the following problem:

> Given $x_1, \ldots, x_n \in R$ decide if there is a non-empty subset $S \subset \{1, \ldots, n\}$ such that $\sum_{i \in S} x_i = 1$.

We shall call this problem the Knapsack Problem (KP) over R. In the classical theory, it is also known as the Subset Sum Problem:

> Given positive integers $x_1, \ldots, x_n, c$ decide if there is a subset $S \subset \{1, \ldots, n\}$ such that $\sum_{i \in S} x_i = c$.

Here we imagine c to be the capacity of a knapsack, x_i the weights of given items, and the question is: Can one fill the knapsack to capacity with some subset of the items?

Note that in our formulation of the Knapsack Problem, the ring R need not be ordered. Thus, the machines we consider here branch only on equality comparisons over R.

In many ways, the Knapsack Problem is similar to our previous decidability problems. Let K_n be the Knapsack set which we write in the following form:

$$K_n = \{x \in R^n \mid \exists b \in \{0,1\}^n \text{ such that } \sum b_i x_i = 1\}. \tag{2}$$

Now we are seeking a machine that will decide, given $x \in R^n$ if $x \in K_n$.

But unlike our previous problems, KP is easily seen to be decidable: Given $x \in R^n$, successively enumerate the non-zero elements of $\{0,1\}^n$ and evaluate the corresponding $\sum b_i x_i$. If and when an evaluation is 1, halt and output 1 (yes). Otherwise, halt and output 0 (no). (See Figure 10.) Given input $x \in R^n$ this algorithm (or machine) will stop with a correct decision in at most $2^n - 1$ enumerations.

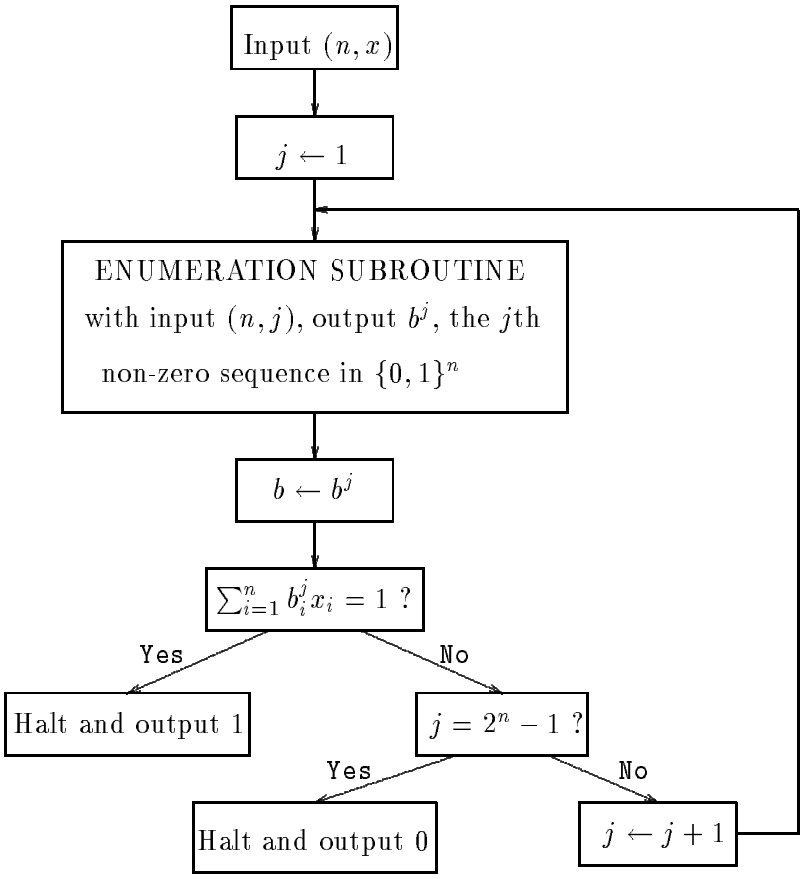

Figure 10 Exhaustive search machine for solving the Knapsack Problem

Fixing n, we can view the input space for our algorithm as the finite dimensional space R^n. However, since the algorithm is "uniform" in n, we can naturally view the input space as R^∞, the infinite direct sum of R (essentially, the space of finite, but unbounded sequences over R). Thus, letting n vary, we have really sketched an algorithm to decide membership in $K = \bigcup K_n$. In the finite dimensional case, the polynomial tests at the branch nodes can be built into the machine. In the "infinite-dimensional" case, the tests at the branch nodes are computed by subroutines.

The exhaustive search algorithm does not at all seem satisfactory. We may very well be unlucky and have to go through an exponential number of iterations before we halt with an answer. The big question is: can we do better? Is there an algorithm for deciding the Knapsack Problem in a polynomial number of "steps"? By this we mean, is there a uniform machine M and a positive integer c such that for each n and $x \in R^n$, M will decide if $x \in K$ in less than n^c steps? We will be more precise in the sequel as to how to define a uniform

machine and how to count steps. For the moment, it is the number of nodes we traverse (counting multiple visits) in a flow chart machine from input to output, given input x.

Notice that the exhaustive search algorithm does not use multiplication. A major unsolved problem is: Can multiplication speed up the process?

Suppose R has no zero divisors. Let $k_n(x) \in R[x_1, \ldots, x_n]$ be the polynomial

$$k_n(x) = \prod_{b \in \{0,1\}^n} (\sum b_i x_i - 1) \tag{3}$$

and $V_{k_n} = \{x \in R^n \mid k_n(x) = 0\}$. Then $V_{k_n} = K_n$. So, for each n we could construct a machine with k_n built in. (See Figure 11.)

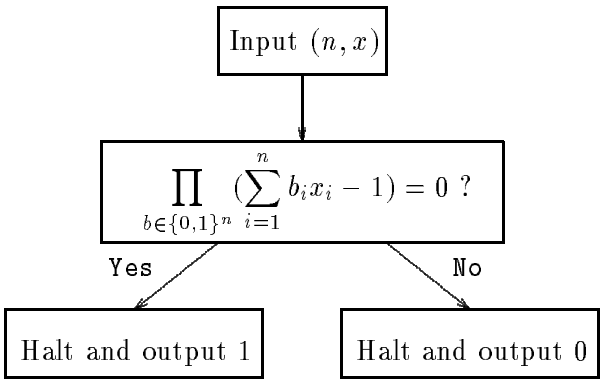

Figure 11 An algebraic machine for solving the Knapsack Problem

The input space of this machine is R^n. It decides if $x \in K$ immediately (in one step) by evaluating the polynomial $k_n(x)$ and checking if the result is 0 or not. The degree of $k_n(x)$ is $2^n - 1$. So we have traded an exponential search for an evaluation of a polynomial whose degree is exponential in n. If R is ordered, a big question remains: Do order comparisons enable significant speed-ups?

While it may be hard to decide in general if $x \in K$ we can quickly verify it is, if we are given a good "witness" $b \in \{0,1\}^n$. Just sum up the corresponding x_i's. We express this property of the Knapsack Problem by saying KP is in the class "NP". Over the integers, KP is a universal problem with this property —it is "NP-complete over $\mathbb{Z}$".

2.5 The Hilbert Nullstellensatz as a Decision Problem

Newton's method tackles the search problem:

> Given a polynomial f over $\mathbb{C}$, *find* a zero of f.

Here we consider the decision problem:

Given a finite set of polynomials f_i in n variables over $\mathbb{C}$, *decide* if the f_i have a common zero.

By the Fundamental Theorem of Algebra, for one polynomial in one variable, the answer is always yes. This is not true for several polynomials in several variables.

We call this decision problem the Hilbert Nullstellensatz over $\mathbb{C}$ and we denote it by HN/$\mathbb{C}$. Thus, one seeks an algorithm, in fact an algebraic algorithm over $\mathbb{C}$, which on input $f = \{f_1, \ldots, f_k\}$ produces `yes` if and only if there is a $\zeta \in \mathbb{C}^n$ such that $f_i(\zeta) = 0$ for all i.

By an algebraic algorithm we have in mind a machine whose computations, like Newton's method, involve the basic arithmetic operations, but whose branching now depends only on equality comparisons and not on order, i.e. a machine now *over* $\mathbb{C}$.

The input f to such a machine can be thought of as the vector of coefficients of the f_i in $\mathbb{C}^N$ where N is given by the formula

$$N = \sum_{i=1}^{k} \left(\begin{array}{c} n + d_i \\ n \end{array} \right), \qquad d_i = \deg f_i, \quad i = 1, \ldots, k$$

Thus, this N represents the size $S(f)$ of the input f.

There are algorithms which accomplish this task. In the linear case, i.e. when $\deg f_i = 1$ for $i = 1, \ldots, k$, this is a simple linear algebra problem. In the general case, Hilbert has shown that the answer is `no` if and only if there exist polynomials $g_1, \ldots, g_k$ in n variables with the property

$$\sum_{i=1}^{k} g_i f_i = 1 \tag{4}$$

where the equality is equality as polynomials. Bounds on the degrees of the g_i have been proved and in fact we may take $\deg g_i \leq D^n$ where $D = \max\{3, d_1, \ldots, d_n\}$. Thus, equation 4 becomes a finite dimensional linear algebra problem; to find the coefficients of the g_i. As one can readily see, the number of Gaussian Elimination steps required is exponential in the size $S(f)$ of the input vector.

This suggests a problem which we formulate as a main conjecture.

Conjecture. The Hilbert Nullstellensatz over $\mathbb{C}$ is intractable.

By this we mean, there is no algorithm which solves HN/$\mathbb{C}$ with the number of arithmetic operations $\mathcal{A}(f)$ satisfying the bound

$$\mathcal{A}(f) \leq S(f)^c$$

where c is a universal constant. But a precise formulation of the conjecture awaits a formal definition of machine over $\mathbb{C}$.

With a formal theory of machines over $\mathbb{C}$ it can be shown that HN/$\mathbb{C}$ is in NP over $\mathbb{C}$ by noting that given f and a test point $\zeta \in \mathbb{C}^n$, we may test ζ for a zero, i.e. whether

$f_i(\zeta) = 0$ for $i = 1, \ldots, k$, in a number of arithmetic operations that is polynomial in $S(f)$. Moreover HN/$\mathbb{C}$ can be shown to be universal with this property so that any problem in NP over $\mathbb{C}$ can be quickly reduced to HN/$\mathbb{C}$. That is to say HN/$\mathbb{C}$ is NP complete over $\mathbb{C}$.

From these considerations it can be shown:

> The Hilbert Nullstellensatz over $\mathbb{C}$ is intractable if and only if P $\neq$ NP over $\mathbb{C}$.

The above sketches our ideas on formulating complexity issues in an algebraic framework. How does this relate to the P $\neq$ NP problem of classical complexity theory?

By abuse of notation write HN/$\mathbb{Z}_2$ for the problem:

> Given a finite set of polynomials f_i in n variables over $\mathbb{Z}_2$, decide if the f_i have a common zero in $(\mathbb{Z}_2)^n$.

By considerations similar to the above, HN/$\mathbb{Z}_2$ can be seen to be intractable if and only if P $\neq$ NP (in the classical sense). Since our algorithms in the above formulation are defined in terms of algebra (characteristic 2 algebra), it could be said that we are placing classical complexity into an algebraic setting in addition to extending it to new problems such as P $\neq$ NP over $\mathbb{C}$. These new problems are interesting in their own right. But also, by posing problems such as P $\neq$ NP within a broader framework, we may be able to employ new mathematical tools to study the classical case as well as gain new insights by analogy or by direct connections.

2.6 Feasibility of Real Polynomials

Let us replace the field of the complex numbers by the real numbers in the problem above and consider polynomials $f_1, \ldots, f_k \in \mathbb{R}[X_1, \ldots, X_n]$. The problem at hand now is to decide if there exists a common root $\xi \in \mathbb{R}^n$. Since the reals have a natural order, it is natural here to consider algorithms that branch on order comparisons.

A particular feature of the real case, not shared by the complex one, enables us to consider the same problem with only one polynomial at the cost of slightly increasing the degree. We associate to the polynomials $f_1, \ldots, f_k$ the single polynomial $g = \sum_{i=1}^{k} f_i^2$. Now g has the property that for every $\xi \in \mathbb{R}^n$, ξ is a common root of all the f_i if and only if $g(\xi) = 0$. Therefore, solving our problem for the f_i turns out to be equivalent to solving it for g. Moreover, if not all the f_i are linear then the degree of g is at least 4. Let us restrict our attention to degree 4 polynomials and consider the following problem denoted by 4-FEAS:

> Given a degree 4 polynomial in n variables with real coefficients, decide whether it has a real zero.

Again, the input g for this problem can be seen as a vector in $\mathbb{R}^N$ where

$$N = \begin{pmatrix} n+4 \\ 4 \end{pmatrix}$$

is the size $S(g)$ of this input.

An algorithm for solving this problem was first given by Tarski in the context of exhibiting a decision procedure for the theory of real numbers. In the context of complexity theory, Tarski's algorithm is highly intractable. The number of arithmetic operations performed by this algorithm grows in the worst case by an exponential tower of n 2's. Later on, Collins devised another algorithm that solved 4-FEAS within a number of arithmetic operations bounded by

$$2^{2^{S(g)}}.$$

More recent algorithms achieve single exponential bounds.These algorithms are quite elaborate.

Again, this suggests a problem which we formulate as another conjecture.

Conjecture. The 4-FEAS problem is intractable.

That is, we conjecture that there is no algorithm which solves 4-FEAS with the number of arithmetic operations $\mathcal{A}(f)$ satisfying the bound

$$\mathcal{A}(f) \leq S(f)^c$$

where c is a universal constant. But again, a precise formulation of the conjecture awaits a formal definition of a machine over $\mathbb{R}$.

As with the Hilbert Nullstellensatz, 4-FEAS is seen to be in NP over $\mathbb{R}$. It is also universal with this property, that is, 4-FEAS is NP complete over $\mathbb{R}$.

2.7 Linear Programming and Integer Programming

Over the reals consider the problem:

> Given a set of m linear inequalities in n variables
>
> $$A_i x \geq b_i \quad i = 1, \ldots, m \quad \text{where } A_i x = \sum_{j=1}^{n} a_{ij} x_j,\ a_{ij} \in \mathbb{R} \text{ and } b_i \in \mathbb{R} \tag{5}$$
>
> decide if there is a point $x \in \mathbb{R}^n$ satisfying (5).

We call this problem the (real) *linear programming feasibility* (LPF) problem.

We now write the system of inequalities (5) as

$$Ax \geq b,$$

where A is the $m \times n$ matrix whose ith row is A_i and b the m-vector whose ith entry is b_i. Then we may rewrite LPF/$\mathbb{R}$ as:

> Given the $m \times n$ real matrix A and $b \in \mathbb{R}^m$, decide if there is an $x \in \mathbb{R}^n$ such that $Ax \geq b$.

This problem is simpler than HN/$\mathbb{R}$ in that the functions are linear, but more complicated by the fact that we consider inequalities. The set of solutions of $Ax \geq b$ is called a *polyhedron*.

The (real) *linear programming optimization* (LPO) problem is:

> With input (A, b, c) *minimize* $c \cdot x$ subject to $Ax \geq b$ where A is an $m \times n$ real matrix, $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$, or decide no minimum exists.

Replacing the reals by the integers or the rationals in LPF we have the *integer programming* (IPF) and the rational linear programming *feasibility* problems:

> Given an $m \times n$ matrix A with integer (respectively rational) entries a_{ij} and $b \in \mathbb{Z}^m$ (respectively $\mathbb{Q}^m$), determine if there is an $x \in \mathbb{Z}^n$ (respectively $\mathbb{Q}^n$) such that $Ax \geq b$.

The corresponding integer programming (IPO) and rational linear programming *optimization* problems are:

$$\begin{array}{c} \text{minimize } c \cdot x \\ \text{subject to } Ax \geq b \end{array}$$

or determine that no minimum exists.

Here A is an $m \times n$ integer (or rational) matrix, $b \in \mathbb{Z}^m$ (or $\mathbb{Q}^m$), $c \in \mathbb{Z}^n$ (or $\mathbb{Q}^n$) and $x \in Z^m$ (or $\mathbb{Q}^m$).

Algorithms are known which solve all six of these problems, but there is a great deal of difference in what is known about the efficiency of algorithms which solve them.

Integer programming is set apart from real or rational linear programming by the fact that the solution of linear equations with integer coefficients are not necessarily integers, e.g. $2x = 1$. But it is also set apart from the reals by the notion of the size of the input. For the reals, the input size $S(A, b)$ or $S(A, b, c)$ of the problem is the number of real variables involved, $mn + m$ for the feasibility problem and $mn + m + n$ for the optimization problem. No algorithm is known for either the feasibility or the optimization problem with total number of arithmetic operations $\mathcal{A}(A, b)$ or $\mathcal{A}(A, b, c)$ satisfying the following complexity bounds:

$$\begin{array}{l} \mathcal{A}(A, b) \leq S(A, b)^d \text{ or} \\ \mathcal{A}(A, b, c) \leq S(A, b, c)^d \end{array}$$

for a universal constant d. Thus an outstanding problem is: Is linear programming tractable over $\mathbb{R}$? Again, to make this precise one needs a formal definition of an algorithm, or machine, over $\mathbb{R}$.

Complexity estimates for algorithms for integer programming traditionally take the binary lengths of the integers used into account both in the input size of a problem "instance" and the "cost" of the algorithm in that instance.

The *height* of an integer x, $ht(x)$, is the first integer greater than or equal to $\log(|x|+1)$.

For the feasibility problem we take input size $S_{ht}(A,b)$ to be equal to be $S(A,b)$ times the maximum height of all the integers a_{ij}, $i=1,\ldots,m$, $j=1,\ldots,n$ and b_i, $i=1,\ldots,m$ where a_{ij} are the entries of the matrix A and b_i the components of the vector b.

For the optimization problem we take input size $S_{ht}(A,b,c)$ to be equal to $S(A,b,c)$ times the maximum height of all the integers a_{ij}, $i=1,\ldots,m$, $j=1,\ldots,n$, b_i, $i=1,\ldots,m$ and c_j, $j=1,\ldots,n$.

The cost of an algorithm for the problem instance (A,b) (respectively (A,b,c)), $C_{ht}(A,b)$ (respectively $C_{ht}(A,b,c)$), is similarly adjusted by multiplying the number of algebraic operations $\mathcal{A}(A,b)$ (respectively $\mathcal{A}(A,b,c)$) by the maximum height of any integer appearing in the computation. The integer programming problems are NP-complete in the classical model. The intractability of either one is equivalent to P $\neq$ NP in the classical case, where now by intractability we mean there is no algorithm and constant $d>0$ for either problem such that

$$C_{ht}(A,b) \leq S_{ht}(A,b)^d \text{ or } C_{ht}(A,b,c) \leq S_{ht}(A,b,c)^d.$$

The situation for the rationals, rational linear programming feasibility and optimization, is dramatically different. The height $ht(x)$ of a rational number $x=\frac{p}{q}$ where p and q are relatively prime is defined as $\max(ht|p|, ht|q|)$. Otherwise the definitions of the input sizes $S_{ht}(A,b)$, $S_{ht}(A,b,c)$, and costs $C_{ht}(A,b)$, $C_{ht}(A,b,c)$ are the same as in integer programming. There are algorithms for both problems and a constant $d>0$ such that

$$C_{ht}(A,b) \leq S_{ht}(A,b)^d \text{ and } C_{ht}(A,b,c) \leq S_{ht}(A,b,c)^d.$$

3 The Classical Theory of Computation

As we have noted, the classical theory of computation had its origins in work of logicians in the 1930's. Of course at that time, there were no computers as we know them. While this work, in particular Turing's (1937), clearly anticipated the development of the modern digital general purpose computer, a primary motivation for the logicians was to formulate and understand the concept of *decidability*, or of a *decidable set*. In particular, the aim was to make sense of such questions: "Is the set of true sentences of arithmetic decidable?" or "Is the set of diophantine equations with integer solutions decidable?" [2]

[2]The latter question is known as Hilbert's Tenth Problem, posed by David Hilbert (along with 22 other seminal problems) at the second International Congress of Mathematicians in Paris on August 8, 1900. It

Intuitively, a set S is decidable if there is an "effective procedure" that given any element u of U (some natural universe containing S) will decide in a finite number of steps whether or not u is in S, i.e. if the characteristic function of S (with respect to U) is "computable." To put the first query in this format, U would be the set of arithmetic sentences, S the true ones. For the second, U would be the set of polynomials with integer coefficients and S the subset of those with integer solutions.

The models of computation designed by these logicians were intended to capture the essence of this concept of effective procedure or computation. The idea was to design theoretical machines with operations, and finitely described rules for proceeding step by step from one operation to the next, so simple and constructive that it would be self-evident that the resulting computations were effective.

A number of distinct formal models of computation were proposed. A primary example is the *Turing machine*. (See Figure 12).

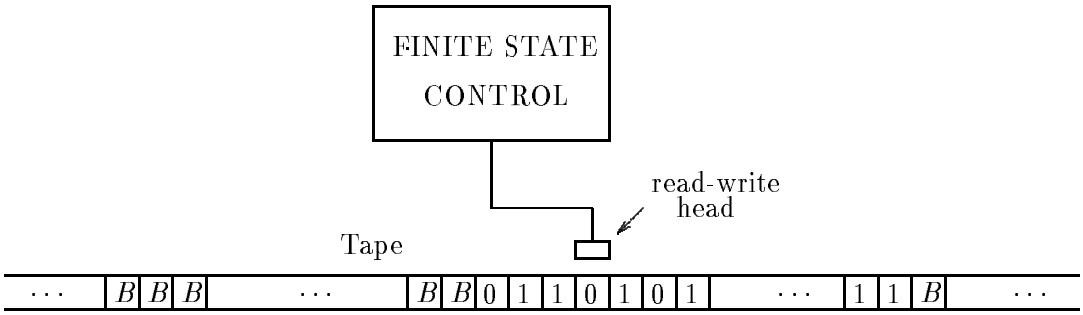

Figure 12 A Turing machine

Here we have a *finite state control* device with a *read-write head* and a two-way infinite *tape* consisting of an infinite number of *cells*. The control device is regulated by a *program* which is a finite set of instructions of the form (q, s, o, q'). Here q and q' belong to a finite set $\{q_0, \ldots, q_N\}$ called the set of *states* of the machine, s is a symbol 0,1, or B (for blank), and o is one of the following *operations*: R (move right one cell), L (move left one cell), 0 (print 0), 1 (print 1), or B (print B). The instruction is interpreted as follows: If the device is in state q with read-write head scanning a cell containing symbol s, then the device performs operation o and goes into state q'. (If o is a print operation then it is implicit that the head erases the current symbol before printing.) We assume the program is *consistent*, i.e. for each q and s there is at most one instruction starting with the pair (q, s).

---

was originally taken for granted, by mathematicians in general, and Hilbert in particular, that the answers to the above questions were both affirmative. The queries were actually posed as tasks: Produce decision procedures for the given sets. The incompleteness/undecidability results of Gödel in 1931 in the first place, and of Matiyasevich in 1971 on the unsolvability of Hilbert's Tenth Problem in the second, show such tasks cannot be carried out.

The machine operates as follows: Given an *input* string x, a finite sequence of 0's and 1's written on consecutive tape cells (with B's everywhere else), the head is placed over the left most symbol of x. (If x is the empty string the head is placed on any cell.) The control device is started in *initial state* q_0 and proceeds according to the program instructions until it can no longer proceed (i.e. it reaches a state q while scanning a symbol s for which there is no instruction starting with the pair q, s). If and when this occurs the *output* is the string of 0's and 1's starting at the current scanned cell (and going right) until the first occurrence of a B (which may be the current cell, in which case the output is the empty string). One might consider input and output strings as natural numbers written in binary (a convention here could be that the empty string is interpreted as 0 and a non-empty input string always has 1 in its left most place).

Here, and in each formalism for computation, a function f from the natural numbers $\mathbb{N}$ to $\mathbb{N}$ is defined to be *computable* if it is the *input-output* map of some such machine. Thus we can now say formally: a set of natural numbers is *decidable* if its characteristic function is computable, in this case by a Turing machine.

A fundamental object of study is the *halting set* of a machine. This is the set of all inputs for which the machine *halts*, i.e. produces an output. It is clear that the halting sets are exactly the *semi-decidable sets*: a set S of natural numbers is semi-decidable if there is a machine which outputs 1 when input an element of S, and otherwise outputs 0 or doesn't halt. A little "programming" shows that S is decidable if and only if both it and its complement are semi-decidable. (Schematically, see Figures 6 and 7.)

This notion of computability can be naturally extended to the integers, $\mathbb{Z}$, the rational numbers, $\mathbb{Q}$, or any domain that can be "effectively encoded" in $\mathbb{N}$. Thus, for example, by "gödel" coding sentences (of a first order language) as natural numbers, one can begin to formally ask (and answer) within the formalism questions about the decidability of the set of true sentences of various mathematical theories.

It is quite remarkable that even though the formalisms we just described and the others proposed were often markedly different, in each case, the resulting class of computable functions —and hence decidable (as well as semi-decidable) sets— was exactly the same. Thus, the class of computable functions appears to be a natural class, independent of any specific model of computation.[3] And consequently, the answers to the basic questions of decidability will be independent of formalism.

This gives one a great deal of confidence in the theoretical foundations of the theory of computation. Indeed, what is known as *Church's thesis* is an assertion of belief that the classical formalisms completely capture our intuitive notion of computable function. Thus for example, in the light of Church's thesis, the negative solution to Hilbert's Tenth Problem can be gotten by showing there is no Turing machine to decide the solvability in integers of

[3] In classical terminology, these functions are often called the *recursive functions*, decidable sets are the *recursive sets* and semi-decidable sets are the *recursively enumerable sets*.

diophantine polynomials. Compelling motivation clearly would be required to justify yet a new model of computation.

4 Toward a Mathematical Foundation of Numerical Analysis

Our perspective is to formulate the laws of computation. Thus we write not from the point of view of the engineer who looks for a good algorithm which solves his problem at hand, or wishes to design a faster computer. The perspective is more like that of a physicist, trying to understand the laws of scientific computation. Idealizations are appropriate, but such idealizations should carry basic truths.

Scientific computation is the domain of computation which is based mainly on the equations of physics. For example, from the equations of fluid mechanics, scientific computation helps understand better design for airplanes, or assists in weather prediction. The theory underlying this side of computation is called numerical analysis.

There is a substantial conflict between theoretical computer science and numerical analysis. These two subjects with common goals have grown apart. For example, computer scientists are uneasy with calculus, while numerical analysis thrives on it. On the other hand numerical analysts see no use for the Turing machine.

The conflict has at its roots another age-old conflict, that between the continuous and the discrete. Computer science is oriented by the digital nature of machines and by its discrete foundations given by Turing machines. For numerical analysis, systems of equations, and differential equations are central and this discipline depends heavily on the continuous nature of the real numbers.

The developments described in the previous section (and next) have given a firm foundation to computer science as a subject in its own right. Use of Turing machines yields a unifying concept of algorithm, well-formalized. Thus this subject has been able to develop a complexity theory which permits discussion of lower bounds of all algorithms without ambiguity.

The situation in numerical analysis is quite the opposite. Algorithms are primarily a means to solve practical problems. There is not even a formal definition of algorithm in the subject. One is reminded of how the development of the definition of differentiable manifold was so important in the history of differentiable topology. The history of algebraic geometry gives us a similar lesson.

Thus we view numerical analysis as an eclectic subject with weak foundations; this certainly in no way denies its great achievements through the centuries.

A major obstacle to reconciling scientific computation and computer science is the present view of the machine, i.e. the digital computer. As long as the computer is seen simply as a finite or discrete object, it will be difficult to systematize numerical analysis.

We believe that the Turing machine as a foundation for real number algorithms can only obscure concepts.

Toward resolving the problem we have posed, we are led to expanding the theoretical model of the machine to allow real numbers as inputs. There has been great hesitation to do this because of the digital nature of the computer. Here, we might learn a lesson from the history of science. In particular, Isaac Newton was faced with an analogous problem in writing his Principia. At the time of Newton, scientists assumed that the world was atomistic, as viewed by the ancient Greek, Democritus. Newton accepted that picture according to which all matter is composed of indivisible particles, a finite number in each bounded region. On the other hand, Newton's mathematics was continuous as was Euclid's. Moreover, the differential equations Newton needed for his theory involved calculus and the continuum, contrasting with the corpuscular view of the universe. It was a substantial problem for Newton to reconcile the discrete world with the continuous mathematics. The resolution was produced by analyzing the effect of replacing an object (e.g. the earth) by a finite number of particles, then making a better approximation with a larger number of particles.

In the limit, the mathematics becomes continuous. *Thomas Kuhn* in "The Copernican Revolution" writes:

> In 1685 [Newton] proved that, whatever the distance to the external corpuscle, all the earth corpuscles could be treated as though they were located at the earth's centre. That surprising discovery, which at last rooted gravity in the individual corpuscles, was the prelude and perhaps the prerequisite to the publication of *Principia*.

And Kuhn adds:

> At last it could be shown that both Kepler's Law and the motion of a projectile could be explained as the result of an innate attraction between the fundamental corpuscles of which the world machine was constructed.

Now our suggestion is that the modern digital computer could be idealized in the same way that Newton idealized his discrete universe. The machine numbers are rational numbers, finite in number, but they fill up a bounded set of real numbers (e.g. between -1000 and 1000) sufficiently densely that viewing the computer as manipulating real numbers is a reasonable idealization, at least in a number of contexts.

Moreover, if one regards computer graphical output such as our picture of the Mandelbrot or Julia sets with their apparently fractal boundaries and asks to describe the machine which made these pictures one is driven to the idealization of machines which work on real or complex numbers in order to give a coherent explanation of these pictures. For a

wide variety of scientific computations the continuous mathematics which the machine is simulating is the correct vehicle for analyzing the operation of the machine itself.

These reasonings give some justification for taking as a model for scientific computation, a machine model which accepts real numbers as inputs. Of course a great many issues such as round-off error must be dealt with. Moreover the ultimate justification is: does the model developed this way give new insights and understanding to the use of the big machines?

5 Classical Complexity Theory and its extension

A cornerstone of classical complexity theory is the theory of NP-completeness and the *fundamental* P $\neq$ NP? problem.

A main goal of this book is to extend this theory to the real and complex numbers, and in particular, to pose and investigate the fundamental problem within a broader mathematical framework.

The foundations for such a theory shall be developed in the next chapters. But here we give some background and briefly and informally introduce some of the classical notions.

Since the 1930's, much work of logicians focused on identifying and classifying decidable and undecidable problems. A prevailing view was that once a problem was known to be decidable (or solvable), then by and large, it was not terribly deep or interesting. In contrast, there was a great deal of interest and activity designed to untangle and understand the rich hierarchy amongst the undecidable problems (the "degrees of unsolvability").

To relate the notion of solvable problem to our earlier discussion of decidability (in Section 3), we can view a *decision problem* as a pair (X, X_{yes}). Here X is the set of *problem instances*, and X_{yes} the subset of *yes-instances*. Thus X plays the role of the universe U and X_{yes} of the subset S. The problem is *decidable* (or *solvable*) if X_{yes} is decidable (by a machine that on input $x \in X$ will output 1 if x is in X_{yes} and 0 if not). So, for example, for Hilbert's Tenth Problem, X would be the set of diophantine equations (polynomial equations with integer coefficients) and X_{yes} the subset of those with integer solutions.

With the advent of the digital computer, and its promise of solving hither-to intractable problems, interest perked in the realm of the solvable with the quest for efficient algorithms. Although there were many successes, it soon became apparent that a number of problems (such as the famous Traveling Salesman Problem) while solvable in principle, defied efficient solution. These problems seemed in essence intractable. Thus, amongst the solvable, there appeared to be yet another rich and natural hierarchy, with the dichotomy of tractability/intractability mirroring the earlier dichotomy of decidability/undecidability. And so, the theory and field of *computational complexity* was born.[4]

[4] Again in his Hixon Symposium lecture, von Neumann voiced the need for such a theory:

> Throughout all modern logic, the only thing that is important is whether a result can be achieved in a finite number of elementary steps or not. The size of the number of steps which

The foundation of this theory was developed in the 1960's, primarily by researchers originally trained in mathematics and logic but who found more hospitable environments for these interests in the newly emerging computer science departments. The theory began in an abstract setting with the formulation of axiomatic complexity measures yielding surprising speed-up theorems, and then became more concrete with the NP-completeness results in the early 1970's. It is primarily this latter work, showing the equivalence of literally thousands of often seemingly unrelated difficult problems, that has captured the attention of researchers from many fields. These problems have the property that an efficient solution to any one can be easily converted to an efficient solution to any other.

The formalisms of classical complexity theory are founded on the models and formalisms of classical computation theory. Formal measures of complexity are intended to indicate various degrees of difficulty inherent in problems. These difficulties could be measured by the amount of information necessary to describe a problem (*descriptional* or *informational complexity*), the power of the language needed (*descriptive complexity*), or the amount of resources, such as time or space, required to *solve* the problem. In this book we will primarily follow the tradition of *computational complexity* which studies the cost of computation with regard to time, or number of steps, for solution. The complexity of a problem is then measured in terms of the complexity of machines for solving it. Paramount here is that *complexity* is given as a function of *input word size* L, classically measured in bits.

A machine M is said to be in *class* P if there are positive integers c and q such that for all inputs x,

$$\text{cost}_M(x) < c(\text{size}(x))^q.$$

Here $\text{cost}_M(x)$ denotes the number of *basic* operations performed by machine M from input x to output. A decision problem (X, X_{yes}) is in *class* P, or solvable in *polynomial time*, if it is decidable by a machine in class P.

Polynomial-time is an attempt to capture a notion of tractability and is what is meant in this discussion when we use qualifiers such as "quick", "efficient", "short" and "fast".

Note that to give upper bounds on complexity or to show a problem is tractable it is sufficient to demonstrate one appropriate machine. On the other hand, to claim a lower

---

are required, on the other hand, is hardly ever a concern of formal logic. Any finite sequence of correct steps is, as a matter of principle, as good as any other. It is a matter of no consequence whether the number is small or large, or even so large that it couldn't possibly be carried out in a lifetime, or in the presumptive lifetime of the stellar universe as we know it.... [On the other hand] in the case of an automaton the thing which matters is not only whether it can reach a certain result in a finite number of steps at all but also how many such steps are needed.

A primary concern here for von Neumann was his conviction that the cumulative effect of the small but non-zero probability of component failure "may (if unchecked) reach the order of magnitude of unity— at which point it produces, in effect, complete unreliability." In fact, to the contrary, the phenomenon of error build-up due to computer failure has not posed difficulties anywhere near the magnitude posed by the (apparent) intractability phenomenon.

bound g for complexity or that a problem is not in *class* P (and hence intractable) is more problematic. For now we must demonstrate that *every* machine for solving it has complexity function that grows faster than g or, in the latter case, faster than any polynomial.

The more subtle concept of *class* NP is meant to capture the notion that some problems have the property that for each yes-instance there exists a quick verification, or short proof, of this fact.

Since a quick decision also serves as a quick verification, we see that class P is contained in class NP. It is natural to ask the converse: If a yes-instance has a short proof, can we find some such proof quickly? This is the essence of the fundamental P=NP? problem.

Again, as in the case of decidability and computability, for all this to be reasonable and natural, we must have some degree of assurance that these notions and classes are independent of most "reasonable" formalisms.

To illustrate some of these ideas, we consider probably the most well known problem of classical complexity theory, the Traveling Salesman Problem (TSP):

> Given n cities, the distances (a_{ij}) between them and a positive number k, does there exist a tour through all the cities with total distance less than or equal to k?

and the related Shortest Path Problem (SPP):

> Given n cities, the distances (a_{ij}) between them, two specified cities l and m, and a positive number k, does there exist a path from l to m with total distance less than or equal to k?

The SPP is solvable in order n^2 operations.[5] On the other hand, the TSP appears not at all to be tractable. All known solutions essentially require us to enumerate the $(n-1)!$ possible tours. By Sterling's formula, $n!$ is asymptotically equal to $(n/2)^n\sqrt{2\pi n}$ which is exponential in n.

Let's look at these problems a bit more formally. First, we can easily pose them as decision problems in the above form. For example, for the TSP let

$$X = \{(A,k) \mid A = (a_{ij}) \text{ is an } n \times n \text{ matrix of } \textit{distances}, k > 0\} \text{ and}$$
$$X_{\text{yes}} = \{(A,k) \in X \mid \text{there is a } \textit{tour } \tau \text{ with } \text{Dist}(A,\tau) \leq k\}$$

Here τ is a cyclic permutation of $\{1,2,\ldots,n\}$ and $\text{Dist}(A,\tau) = \sum_{i=1}^{n-1} a_{\tau_i\tau_{i+1}} + a_{\tau_n\tau_1}$.

[5] We indicate a solution to the special case when the distances between distinct cities are either 1 or $k+1$: At stage 0, label city l with the number 0. At stage $s+1$, label all unlabeled cities that are distance 1 from the cities labeled s by the number $s+1$. If no such cities exist, terminate process and answer "yes" if city m is labeled by a number $\leq k$, otherwise answer "no."

Notice that X is the set of *all* problem instances, for all n. This reflects the fact that we are interested in solving problems uniformly.

Notice also that, in stating these particular problems, we have made no assumption that the distances are integers; it makes perfectly good sense to talk about these particular problems over the reals or any ring with order.

Over $\mathbb{R}$, a natural measure of the size of a TSP or SPP instance would be n^2 (the number of entries in the matrix A of distances). Over $\mathbb{Z}$, a more natural measure would be n^2b where b is the maximum of the heights (or binary lengths) of the distances (a_{ij}) and k. This size roughly reflects the number of symbols needed to describe the instance (or its bit length) and is essentially the classical measure.

The classical measure of cost, the bit cost, is the number of Turing machine operations for solution. Thus the bit costs of the above solutions for SPP and TSP are of order n^2b and $(n-1)!b$ respectively. Over the reals, a natural measure of cost could be the number of arithmetic computations and comparisons, and so for the above solutions to SPP and TSP, of order n^2 and $(n-1)!$ respectively. Thus, over $\mathbb{R}$ or over $\mathbb{Z}$, the cost of these solutions (as a function of size of instance) is linear in the case of SPP and exponential in the case of the TSP.

Now the TSP, while not known to be in class P over any ordered ring, is seen to be in class NP in any reasonable sense. Although we may not be able to easily tell if a TSP instance has a "good" tour (i.e. one of total distance bounded by k), if we are handed a good one we can quickly check it out: First check that it is indeed a tour and then sum up the n distances along the tour and compare with k.

The TSP is NP-*complete* over $\mathbb{Z}$, i.e. it is universal for NP problems over $\mathbb{Z}$: If (X, X_{yes}) is a problem in NP over $\mathbb{Z}$, then *problem instances* $x \in X$ can be efficiently encoded as Travelling Salesman instances T_x such that $x \in X_{\text{yes}}$ if and only if T_x has a good tour. Thus an efficient solution to the TSP will yield an efficient solution to any other NP problem. Hence the importance of the TSP in classical complexity theory —not only because it is one of the ubiquitous problems of discrete optimization, but also because of its NP-completeness!

The Knapsack Problem (KP) introduced in Section 2.4 can also be posed as a decision problem in the above form. Let

$$\begin{aligned} X &= R^\infty = \bigcup_{n \geq 0} R^n \\ X_{\text{yes}} &= \{x \in X \mid \exists b \in \{0,1\}^n \text{ such that } \textstyle\sum b_i x_i = 1\} \end{aligned} \tag{6}$$

Over any ring R, KP is in class NP in any reasonable sense. Moreover, the Knapsack Problem is also NP-complete over $\mathbb{Z}$ and hence equivalent to the TSP with respect to complexity.

We propose a theory of NP and NP-completeness over an arbitrary ring. In such a framework one can obtain both old and new NP-completeness results. The classical satisfiability problem is NP-complete over the field $\mathbb{Z}_2$. The integer programming problem

is naturally NP-complete over $\mathbb{Z}$. In non-classical domains the Hilbert Nullstellensatz is NP-complete over $\mathbb{C}$ or $\mathbb{R}$. Hierarchies of complexity classes can be developed over the real numbers.

6 Complexity Theory in Numerical Analysis

It is natural to be skeptical about machines using exact arithmetic in numerical analysis. Most numerical problems can only be solved to within an accuracy of ε. Round-off error is an important fact in the use of actual machines for solving scientific problems. Does it make sense to try to extend the complexity theory of computer science to numerical analysis?

We recall that computer scientists say that an algorithm defined by a machine M is *tractable* (or polynomial time, or in P) if the computation time $T(x)$ associated to input x satisfies the bound

$$T(x) \leq c(\text{size } (x))^q \quad \text{all inputs } x \tag{7}$$

where the constants c and q depend only on M. Here time is the number of Turing machine operations and size is the number of bits. A *problem is tractable* if there is a tractable algorithm solving it.

Some of the algorithms of numerical analysis are quite immediately tractable in a natural extension of this definition. Consider the problem of solving a linear system of equations $Ax = b$. The input of the problem is a non-singular $n \times n$ matrix A and a vector $b \in \mathbb{R}^n$. Gaussian elimination produces an output x, solving this problem in less than cn^3 arithmetic operations. Therefore one can speak of the "tractability" of Gaussian elimination where Turing operators are replaced by arithmetic operations (and comparisons). Also the size of the input now becomes naturally the number of input variables. Thus

$$T(A, b) \leq c(\text{size}(A, B))^{3/2}$$

for Gaussian elimination.

More generally in numerical analysis it is important to take into account the desired accuracy ε of an approximate solution. This is because most problems cannot be solved exactly, even using exact arithmetic. Thus one must modify the concept of tractable and one way to do this is consider $\varepsilon < 1$ as an additional (special) input to the problem. Then one demands that the time T of computation satisfy

$$T(\varepsilon, x) \leq (|\log \varepsilon| + \text{ size}(x))^q, \ \varepsilon < 1. \tag{8}$$

Much recent work on solving non-linear equations fits into this framework, where even sometimes $|\log \varepsilon|$ is replaced by $\log|\log \varepsilon|$ in (8).

Frequently among the set of inputs to a problem, there is a subset of "ill-posed" problems where the main algorithms fail and may even fail in principle. In general as an input gets closer to this ill-posed set the time of computation becomes larger.

A "condition number", a function on the input, has been defined traditionally to deal with this phenomenon. If the condition number of a certain input is large, then the time of computation can be expected to become large and the effects of round off error to become substantial. It has often been shown that there is a relation between the condition number of an input and the reciprocal of the distance to the ill-posed set. Then the desired complexity results have the form

$$T(\varepsilon, x) \leq (|\log \varepsilon| + \log \mu(x) + \text{size}(x))^q$$

where μ is the condition number of x.

7 Summary

As we have said, our proposal is to develop a theory of machines which will take real numbers as inputs.

Generally speaking, mathematical theories are built on plausible abstractions and simplifications intended to capture the essence of, rather than precisely describe, phenomena they are attempting to model. We hope that the basic assumptions reflect fundamental underlying principles, and that the results inferred from these assumptions reveal new truths. Justification for our proposal will ultimately depend on how well this last task is accomplished.

The basic arithmetic operations $(+,-,\times,/)$ are to be taken as primary in the structures of computation. This point of view bestows an algebraic emphasis so that it becomes natural to suppose that the inputs and states of the machines are numbers (or finite sequences of numbers) in a field (mathematical sense of the word). In the main case this is the field of real numbers. But certainly the field of complex numbers is also important.

There are natural situations where division can't be done as within the integers $\mathbb{Z}$. So to cover those cases we propose a model of computation of machines over a ring.

Now formulating a theory of computation in this manner, i.e., over a field K, one can include and extend the classical theory by taking $K = \mathbb{Z}_2$ (the field of 2 elements). In this way, the classical theory takes on an algebraic setting. By choosing K to be the real numbers $\mathbb{R}$, we are able to obtain a setting which provides a foundation of numerical analysis. The notion of an algorithm over $\mathbb{R}$, becomes well-defined as a mathematical object in its own right. So we will have developed an *extension* of the classical theory to a new theory which can be specialized to the study of real number algorithms. This theory by the nature of our development is primarily algebraic. More precisely, when the field K is an ordered field as is the case of $\mathbb{R}$, the comparisons include $\leq$, and the geometry becomes what is called semi-algebraic. The classical algorithms of mathematics and of computer science naturally fit into this framework.

It is important to remark that a fundamental property of classical computation is that the machines are finite objects, even though they operate on inputs which have no a priori

bound on size. This property is satisfied by the machines suggested here.

8 Brief History

The ideas we have presented are at the confluence of different traditions in mathematics and computer science.

On the one hand, there is the work of classical computability and complexity. The initial motivating force here was —as we have already remarked— the question of the decidability of the arithmetic, and also the tenth problem posed by Hilbert at the second International Congress of Mathematicians in 1900. A common characteristic of these problems is the possibility of expressing their underlying objects (arithmetic sentences and diophantine equations) in a language over a finite alphabet. Not surprisingly, the host of theoretical computational models that were subsequently proposed to formalize the notion of decidability were designed to act on finite strings over a finite alphabet.[6]

This was the case with the general recursive functions of Kleene [1936], the λ-computable functions of Church [1936], the computable functions of Turing [1936] and the canonical systems of Post [1943], to mention just the most influential models. Perhaps less expectedly, all models were equivalent in the sense that they defined the same class of computable functions. This gave rise to *Church's thesis* discussed in Section 3.

On the other hand, there is a long standing tradition of decidability results in algebra and analysis that we refer to as the *numerical tradition.* This theory lead to algorithms —like Newton's method discussed in Section 2 and Gaussian elimination for solving linear systems of equations— as well as to several undecidability results. A paradigm here is Galois' result on the non solvability by radicals of polynomial equations of degree 5 or more. It is important to notice that these algorithms manipulate real numbers in much the way proposed here.

With the arrival of the digital computer, attention shifted from decidability to complexity issues, and the first of the traditions described above produced a sophisticated theory of complexity of which the P vs. NP question described in Section 5 became central. We owe to this research concepts and tools that enable us to classify computational problems into complexity classes reflecting different resource requirements, and then to discover structural relations among these classes.

During the 1960's, Rabin [1960b], Hartmanis and Stearn [1965] and Blum [1967] developed the notion of measuring the complexity of a problem in terms of the number of steps required to solve it with an algorithm. This lead in a natural way to the association by

[6] Furthermore, at the turn of the century, with recent discoveries of paradoxes in the foundation of mathematics well in mind, there was an understandable preoccupation with questions of consistency. This surely was a factor in stipulating, in the early computational models, that the simplest operations were to be performed on the simplest objects.

Cobham [1964], Edmonds [1965] and Rabin [1966] of the concept of "feasible" or "tractable" problems to the class P. Simultaneously, it was observed that a large class of search problems seemed to defy the existence of algorithms significantly better than brute force. Independently Cook [1971] and Levin [1973] characterized this class (Cook named it NP) and proved the existence of complete problems for it. Cook exhibited the first NP-complete problem, the Satisfiability Problem of propositional logic. Shortly afterwards, Karp [1972] showed that a series of familiar problems from different areas of discrete mathematics were also NP-complete. This gave strong impetus to the subject that was reflected, on the one hand, in work exhibiting hundreds of NP-complete problems and, on the other hand, in attempts to prove the inequality $\mathrm{P} \neq \mathrm{NP}$ leading to results on the structure of the class NP.

A lively exposition on the P vs. NP question (containing a large list of NP-complete problems) can be found in the already classic book by Garey and Johnson [1979]. A survey of the state of the art of this question is given in [Sipser 1992]. In this latter article, a recently discovered letter of Gödel to von Neumann dated 1956 is reproduced in which Gödel stated the P vs. NP question in the form of the time required by a Turing machine to test whether a formula of the predicate calculus has a proof of a given length.

The rise of complexity issues in the numerical tradition is less attached to the advent of the digital computer. Early in 1937, in a short note of Scholz [1937], complexity questions arose under the form of the number of additions needed to produce a given integer starting from 1. Seventeen years later Ostrowski [1954] conjectured the optimality of Horner's rule for evaluating univariate polynomials. In order to do so, he defined a formal model of computation and associated to it an idea of cost. This was followed by a flow of results concerning lower bounds (including the proof of Ostrowski's conjecture by Pan [1966]) for computational models with the following two characteristics:

(i) they take their inputs from R^n where R is a ring, and

(ii) their basic operations are arithmetic and complexity is measured by how many such operations are performed.

In most cases, the ring R was chosen to be the field of real numbers $\mathbb{R}$ and this choice, together with the second characteristic above, reflected the kind of computations done in numerical analysis. However, these models were essentially non uniform. This fact, useful for the search of lower bounds, becomes an obstruction to developing a theory of complexity for general purpose algorithms. Two very influential papers at the end of the 1960's were those of Winograd [1967] and Straßen [1969]. They helped to make this search for lower bounds in algebraic problems an independent subject of study, now known as *algebraic complexity*. Some central examples of algebraic computational models along with lower bounds for them are given by Steele and Yao [1982], Ben-Or [1983] and Smale [1987]. Two early books on algebraic complexity are the ones by Borodin and Munro [1975] and by Winograd [1980]. A recent survey of the subject can be found in [Straßen 1990].

Complexity issues are at the forefront of current research related to designing algorithms for finding zeros of polynomials and determining the solvability of polynomial systems. Amongst the major references here are: Collins [1975] Shub and Smale [1993a, 1993b, 1993c, 1993d, 1994], Schönhage [1982], Ben-Or, Kozen and Reif [1986], Renegar [1987, 1992], Pan [1987, 1995], Grigor'ev and Vorobjov [1988], Canny [1988], and Heintz, Roy and Solerno [1990]. This work may be considered the modern counterpart to algorithmic investigations begun earlier in the century by Hermann [1926], Van der Waerden [1949], and Tarski [1951], in particular related to elimination theory for real closed fields. While the history of numerical analysis provides us with a great motivating force towards our efforts here, we will only give thje reference [Goldstine 1977].

The computational model proposed in this article has a candidate in [Blum, Shub, and Smale 1989]. This candidate lies on the traditions of both computer science and numerical analysis since it incorporates the universality of universal machines and of NP-complete problems, while keeping the assumptions of the numerical one (real numbers given as an entity and unit cost of arithmetical operations) that make it suitable for modelling continuous algorithms. There is a growing body of work —by Cucker [1993, 1992b, 1992a], Koiran [1993], Meer [1990, 1992, 1993, 1994], Michaux [1989, 1991], and Poizat [1995] among others— giving a broad development to this point of view.

In addition to the work already described, there are many more contributions by mathematicians and computer scientists which predate the aforementioned model. We proceed now to review some of them.

Close to the classical approach, Rabin [1960a] developed a theory of computable algebra and fields in which the underlying domains can be effectively coded by natural numbers and are thus, necessarily countable.

On the other hand, the theories of computation over abstract structures, are quite general. See e.g., Friedman [1971] (or as discussed by Shepherdson in Harrington *et al.* [1985]), Tiuryn [1979], and Moschovakis [1986]. These general approaches both exploit and explore the logical properties of procedures. But, when applied to specific structures such as the reals, they do not yield the concrete mathematical results (such as the undecidability of the Mandelbrot set, NP-completeness of the Hilbert Nullstellensatz) that will quite naturally follow from the model we propose here.

There is yet another possible approach to the complexity of real valued problems known as recursive analysis, originating with Turing's seminal paper [Turing 1936]. Indeed, in this paper Turing introduced the notion of computable real numbers *before*, and as a means to, defining computation over the integers.[7] Here the machine model is the classical Turing

[7]This fact seems not well known, so it is of considerable historical interest to examine the very first paragraph of Turing's paper:

> The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable numbers, it is almost equally easy to define and investigate computable functions

machine and one deals with real numbers that, roughly speaking, are fed to the machine bit by bit. This contrasts with the numerical tradition where real numbers are viewed not as their decimal (or binary) expansion, but rather as mathematical entities. Text references for recursive analysis are [Ko 1991] for complexity matters and [Weihrauch 1987] for computability issues. Other references are Friedman and Ko [1982], Pour-El and Richards [1983], Hoover [1987] and Kreitz and Weihrauch [1982].

More closely related to our perspective are the register machines of Shepherdson and Sturgis [1963] and the RAM's or random access machines. This were originally defined over the integers (see [Aho, Hopcroft, and Ullman 1974] for a definition) but with an algebraic character in their ground operations. An extension of the RAM to the real numbers is suggested in the book of Preparata and Shamos [1985]. The goal of the model is primarily to describe algorithms in computational geometry and the formal development of a theory of computability or complexity is not pursued. Also, in the book mentioned above by Borodin and Munro [1975], the authors state that their underlying model of computation will be the RAM. However, immediately afterwards they say that this "code can be 'unwound' and separate programs can be written for each 'degree' of the desired class of functions", justifying therefore the subsequent use of models of fixed dimension. Again, the formal development of a theory of computability and complexity over fields is not pursued.

Perhaps closest to our approach is the work of Herman and Isard [1970] on computability over arbitrary fields. Here, some finite dimensional problems over the reals are shown to be undecidable in a manner similar to our proof of the undecidability of the Mandelbrot set. Also close is the work of Tucker [1980] and Tucker and Zucker [1992] who employ the theory of computing over abstract structures to obtain computability and non-computability results in line with ours. Friedman and Mansfield [1992] have also specialized the abstract theory to specific structures to good avail.

Another model, again close in spirit, is a theory of real Turing machines outlined by Abramson [1971]. The machine model developed here can operate on arbitrarily long vectors of real numbers. The main thrust of the article is to develop a hierarchy of non-computable functions according to their use of a greater-lower-bound operation.

Yet another approach is information-based complexity, developed in Traub *et al.* [1988]. A paradigm problem whose complexity is analyzed here is: given a function f of class $\mathcal{C}^p$ in $[0,1]^n$, compute $\int_{[0,1]^n} f$. As one notes, inputs for this problem can not be given in general

---

of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

by a finite vector of real numbers. So one must assume the existence of a routine that given $x \in [0,1]^n$ returns $f(x)$. The complexity is evaluated in terms of the operations done as well as in terms of the number of times this routine is used. Again we are in the realm of the numerical tradition since the arithmetic is performed on real numbers at a constant cost and the main issue is the search for lower and upper bounds.

We close this section with some general references to the topics introduced in this chapter.

A good reference for the Mandelbrot and Julia sets is Devaney [1989]. For the undecidability of the Mandelbrot set see Blum and Smale [1993]. A major reference for Hilbert's Tenth Problem is Matiyasevich [1993].

For the Nullstellensatz, see Lang [1993] or Kendig [1977]. More advanced books in algebraic geometry are those of Hartshorne [1977] or Shafarevich [1977]. These references however, only deal with the qualitative aspect of the Nullstellensatz. Exponential bounds for the degrees in the Nullstellensatz were proved by Bronawell [1987] and refined by Kollar [1988] and Caniglia, Galligo and Heintz [1988]. Real polynomials, real algebraic sets and semi-algebraic sets are exposed in the monographs of Benedetti and Risler [1990] and by Bochnak, Coste and Roy [1987].

For Newton's method see Smale's survey article [Smale 1985]. A classical reference for linear and integer programming is the book of Schrijver [1986].

For the classical theory of computability and Turing machines see the books by Davis [1965], Rogers [1967] and Cutland [1980]. Classical complexity theory is a younger subject. The books by Balcázar, Díaz and Gabarró [1988] and [1990] or by Papadimitriou [1994] offer very good introductions to its achievements. Other references for this chapter are [Blum 1991; Blum 1990], [Smale 1988; Smale 1990] and [Hirsch, Marsden, and Shub 1993].

The quotations from von Neumann, Penrose and Kuhn are taken from [von Neumann 1963; Penrose 1991; Kuhn 1957].

The reference list, while extensive, is not meant to be exhaustive.

References

Abramson, F. (1971). Effective computation over the real numbers. In *12th annual IEEE Symp. on Switching and Automata Theory*, pp. 33–37.

Aho, A., J. Hopcroft, and J. Ullman (1974). *The design and analysis of computer algorithms*. Addison-Wesley.

Balcázar, J., J. Díaz, and J. Gabarró (1988). *Structural Complexity I*. EATCS Monographs on Theoretical Computer Science, 11. Springer-Verlag.

Balcázar, J., J. Díaz, and J. Gabarró (1990). *Structural Complexity II*. EATCS Monographs on Theoretical Computer Science, 22. Springer-Verlag.

Ben-Or, M. (1983). Lower bounds for algebraic computation trees. In *15th annual ACM Symp. on the Theory of Computing*, pp. 80–86.

Ben-Or, M., D. Kozen, and J. Reif (1986). The complexity of elementary algebra and geometry. *J. of Computer and Systems Sciences 18*, 251–264.

Benedetti, R. and J.-J. Risler (1990). *Real algebraic and semi-algebraic sets.* Hermann.

Blum, L. (1990). Lectures on a theory of computation and complexity over the reals (or an arbitrary ring). In E. Jen (Ed.), *Lectures in the Sciences of Complexity II*, pp. 1–47. Addison-Wesley.

Blum, L. (1991). A theory of computation and complexity over the real numbers. In *Proceedings of the International Congress of Mathematicians*, pp. 1491–1507. Springer-Verlag.

Blum, L., M. Shub, and S. Smale (1989). On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin of the Amer. Math. Soc. 21*, 1–46.

Blum, L. and S. Smale (1993). The Gödel incompleteness theorem and decidability over a ring. In M. Hirsch, J. Marsden, and M. Shub (Eds.), *From Topology to Computation: Proceedings of the Smalefest*, pp. 321–339. Springer-Verlag.

Blum, M. (1967). A machine-independent theory of the complexity of recursive functions. *Journal of the ACM 14*, 322–336.

Bochnak, J., M. Coste, and M.-F. Roy (1987). *Géométrie algébrique réelle.* Springer-Verlag.

Borodin, A. and I. Munro (1975). *The Computational Complexity of Algebraic and Numeric Problems.* American Elsevier.

Brownawell, W. (1987). Bounds for the degrees in the Nullstellensatz. *Annals of Math. 126*, 577–591.

Caniglia, L., A. Galligo, and J. Heintz (1988). Borne simple exponentielle pour les degrés dans les théorèmes de zéros sur un corps de caractéristique quelconque. *C. R. Acad. Sci. Paris 307*, 255–258.

Canny, J. (1988). Some algebraic and geometric computations in PSPACE. In *20th annual ACM Symp. on the Theory of Computing*, pp. 460–467.

Church, A. (1936). An unsolvable problem of elementary number theory. *Amer. J. of Math. 58*, 354–363.

Cobham, A. (1964). The intrinsic computational difficulty of problems. In *International Congress for Logic, Methodology, and the Philosophy of Science,* edited by Y. Bar-Hillel, North-Holland, pp. 24–30.

Collins, G. (1975). *Quantifier elimination for real closed fields by cylindrical algebraic deccomposition*, Volume 33 of *Lect. Notes in Comp. Sci.*, pp. 134–183. Springer-Verlag.

Cook, S. (1971). The complexity of theorem proving procedures. In *3rd annual ACM Symp. on the Theory of Computing*, pp. 151–158.

Cucker, F. (1992a). The arithmetical hierarchy over the reals. *Journal of Logic and Computation 2*, 375–395.

Cucker, F. (1992b). $\mathrm{P}_{\mathbb{R}} \neq \mathrm{NC}_{\mathbb{R}}$. *Journal of Complexity 8*, 230–238.

Cucker, F. (1993). On the complexity of quantifier elimination: the structural approach. *The Computer Journal 36*, 400–408.

Cutland, N. (1980). *Computability.* Cambridge University Press.

Davis, M. (1965). *The Undecidable.* Raven Press.

Devaney, R. (1989). *Chaotic Dynamical Systems.* Addison-Wesley.

Edmonds, J. (1965). Paths, trees, and flowers. *Canadian Journal of Mathematics 17*, 449–467.

Friedman, H. (1971). Algorithmic procedures, generalized turing algorithms, and elementary recursion theory. In R. Gandy and Yates (Eds.), *Logic Colloquium 1969*, C.M.E., pp. 361–390. North-Holland.

Friedman, H. and K. Ko (1982). Computational complexity of real functions. *Theoretical Computer Science 20*, 323–352.

Friedman, H. and R. Mansfield (1992). Algorithmic procedures. *Transactions of the Amer. Math. Soc. 332*, 297–312.

Garey, M. and D. Johnson (1979). *Computers and Intractability: a guide to the theory of NP-completeness.* Freeman.

Goldstine, H. (1977). *A History of Numerical Analysis from the 16th through the 19th Century.* Springer-Verlag.

Grigoriev, D. and N. Vorobjov (1988). Solving systems of polynomial inequalities in subexponential time. *Journal of Symbolic Computation 5*, 37–64.

Harrington, L., M. Morley, A. Seedrov, and S. Simpson (Eds.) (1985). *Harvey Friedman's Research on the Foundations of Mathematics.* North-Holland.

Hartmanis, J. and R. Stearns (1965). On the computational complexity of algorithms. *Transactions of the Amer. Math. Soc. 117*, 285–306.

Hartshorne, R. (1977). *Algebraic Geometry.* Springer-Verlag.

Heintz, J., M.-F. Roy, and P. Solerno (1990). Sur la complexité du principe de Tarski-Seidenberg. *Bulletin de la Société Mathématique de France 118*, 101–126.

Herman, G. and S. Isard (1970). Computability over arbitrary fields. *J. London Math. Soc. 2*, 73–79.

Hermann, G. (1926). Die Frage der endlich vielen Schritte in der Theorie der Polynomideale. *Math. Ann. 95*, 736–788.

Hirsch, M., J. Marsden, and M. Shub (Eds.) (1993). *From Topology to Computation: Proceedings of the Smalefest.* Springer-Verlag.

Hoover, H. (1987). *Feasibly constructive analysis.* Ph. D. Thesis, Dept. of Comp. Sci., Univ. of Toronto.

Karp, R. (1972). Reducibility among combinatorial problems. In R. Miller and J. Thatcher (Eds.), *Complexity of Computer Computations*, pp. 85–103. Plenum Press.

Kendig, K. (1977). *Elementary Algebraic Geometry.* Springer-Verlag.

Kleene, S. (1936). General recursive functions of natural numbers. *Math. Annalen 112*, 727–742.

Ko, K. (1991). *Complexity theory of real functions.* Birkhäuser.

Koiran, P. (1993). A weak version of the Blum, Shub & Smale model. In *34th annual IEEE Symp. on Foundations of Computer Science*, pp. 486–495.

Kollár, J. (1988). Sharp effective Nullstellensatz. *Journal of Amer. Math. Soc. 1*, 963–975.

Kreitz, C. and K. Weihrauch (1982). Complexity theory of real numbers and functions. In A. Cremers and H. Kreigel (Eds.), *Theoretical Computer Science*, Volume 145 of *Lect. Notes in Comp. Sci.*, pp. 165–174. Springer-Verlag.

Kuhn, T. (1957). *The Copernican revolution: planetary astronomy in the development of the Western thought.* Harvard University Press.

Lang, S. (1993). *Algebra, 3rd edition.* Addison-Wesley.

Levin, L. (1973). Universal sequential search problems. *Probl. Pered. Inform. IX 3*, 265–266. (In Russian, English translation in *Problems of Information Trans.* 9,3; corrected translation in [Trakhtenbrot 1984]).

Matiyasevich, Y. (1993). *Hilbert's Tenth Problem*. The MIT Press.

Meer, K. (1990). Computations over $\mathbb{Z}$ and $\mathbb{R}$: a comparison. *Journal of Complexity 6*, 256–263.

Meer, K. (1992). A note on a P $\neq$ NP result for a restricted class of real machines. *Journal of Complexity 8*, 451–453.

Meer, K. (1993). Real number models under various sets of operations. *Journal of Complexity 9*, 366–372.

Meer, K. (1994). On the complexity of quadratic programming in real number models of computation. *Theoretical Computer Science 133*, 85–94.

Michaux, C. (1989). Une remarque à propos des machines sur $\mathbb{R}$ introduites par Blum, Shub et Smale. *C. R. Acad. Sci. Paris 309, Série I*, 435–437.

Michaux, C. (1991). Ordered rings over which output sets are recursively enumerable. *Proceedings of the Amer. Math. Soc. 112*, 569–575.

Moschovakis, Y. (1986). Foundations of the theory of algorithms. Draft.

Ostrowski, A. (1954). On two problems in abstract algebra connected with Horner's rule. In *Studies in Mathematics and Mechanics presented to Richard von Mises*, pp. 40–48. Academic Press.

Pan, V. (1966). Methods of computing values of polynomials. *Russian Math. Surveys 21*, 105–136.

Pan, V. (1987). Sequential and parallel complexity of approximate evaluation of polynomial zeros. *Comput. Math. Appl. 14*, 591–622.

Pan, V. (1995). Optimal (up to polylog factors) sequential and parallel algorithms for approximating complex polynomial zeros. In *27th annual ACM Symp. on the Theory of Computing*, pp. 741–750.

Papadimitriou, C. (1994). *Computational Complexity*. Addison-Wesley.

Peitgen, H.-O. and D. Saupe (Eds.) (1988). *The Science of Fractal Images*. Springer-Verlag.

Penrose, R. (1991). *The Emperor's New Mind*. Penguin Books.

Poizat, B. (1995). *Les Petits Cailloux*. Aléa.

Post, E. (1943). Formal reductions of the general combinatorial decision problem. *Amer. Journal of Math. 65*, 197–268.

Pour-El, M. and I. Richards (1983). Computability and noncomputability in classical analysis. *Transactions of the Amer. Math. Soc. 275*, 539–560.

Preparata, F. and M. Shamos (1985). *Computational Geometry: an introduction*. Texts and Monographs in Computer Science, Springer-Verlag.

Rabin, M. (1960a). Computable algebra, general theory and theory of computable fields. *Transactions of the Amer. Math. Soc. 95*, 341–360.

Rabin, M. (1960b). Degree of difficulty of computing a function and a partial ordering of recursive sets. Technical Report 2, Hebrew University of Jerusalem.

Rabin, M. (1966). Mathematical theory of automata. In *19th ACM Symp. in Applied Mathematics*, pp. 153–175.

Renegar, J. (1987). On the efficiency of Newton's method in approximating all zeros of systems of complex polynomials. *Math. of Oper. Research 12*, 121–148.

Renegar, J. (1992). On the computational complexity and geometry of the first-order theory of the reals. Part I. *Journal of Symbolic Computation 13*, 255–299.

Rogers, H. (1967). *Theory of Recursive Functions and Effective Computability*. McGraw-Hill.

Scholz, A. (1937). Aufgabe 253. *Jahresber. Deutsch. Math.-Verein. 47*, 41–42.

Schönhage, A. (1982). The fundamental theorem of algebra in terms of computational complexity. Technical report, Math. Institut der. Univ. Tübingen.

Schrijver, A. (1986). *Theory of Linear and Integer Programming*. John Wiley & Sons.

Shafarevich, I. (1977). *Basic Algebraic Geometry*. Springer-Verlag.

Shepherdson, J. and H. Sturgis (1963). Computability of recursive functions. *Journal of the ACM 10*, 217–255.

Shub, M. and S. Smale (1993a). Complexity of Bezout's theorem I: geometric aspects. *Journal of the Amer. Math. Soc. 6*, 459–501.

Shub, M. and S. Smale (1993b). Complexity of Bezout's theorem II: volumes and probabilities. In F. Eyssette and A. Galligo (Eds.), *Computational Algebraic Geometry*, Volume 109 of *Progress in Mathematics*, pp. 267–285. Birkhäuser.

Shub, M. and S. Smale (1993c). Complexity of Bezout's theorem III: condition number and packing. *Journal of Complexity 9*, 4–14.

Shub, M. and S. Smale (1993d). Complexity of Bezout's theorem IV: probability of success, extensions. To appear at *SIAM J. of Numer. Anal.*

Shub, M. and S. Smale (1994). Complexity of Bezout's theorem V: polynomial time. *Theoretical Computer Science 133*, 141–164.

Sipser, M. (1992). The History and Status of the P versus NP Question. In *24th annual ACM Symp. on the Theory of Computing*, pp. 603–618.

Smale, S. (1985). On the efficiency of algorithms of analysis. *Bulletin of the Amer. Math. Soc. 13*, 87–121.

Smale, S. (1987). On the topology of algorithms I. *Journal of Complexity 3*, 81–89.

Smale, S. (1988). The Newtonian contribution to our understanding of the computer. In M. Stayer (Ed.), *Newton's Dream*. McGill-Queens University Press.

Smale, S. (1990). Some remarks on the foundations of numerical analysis. *SIAM Review 32*, 211–220.

Steele, J. and A. Yao (1982). Lower bounds for algebraic decision trees. *Journal of Algorithms 3*, 1–8.

Straßen, V. (1969). Gaussian elimination is not optimal. *Numer. Math. 13*, 354–356.

Straßen, V. (1990). Algebraic complexity theory. In J. van Leeuwen (Ed.), *Handbook of Theoretical Computer Science*, Volume A, pp. 633–672. The MIT Press/Elsevier.

Tarski, A. (1951). *A Decision Method for Elementary Algebra and Geometry*. University of California Press.

Trakhtenbrot, B. (1984). A survey of russian approaches to perebor (brute-force search) algorithms. *Annals of the History of Computing 6*, 384–400.

Traub, J., G. Wasilkowski, and H. Wozniakowski (1988). *Information-Based Complexity*. Academic Press.

Tucker, J. (1980). Computing in algebraic systems. In F. Drake and S. Wainer (Eds.), *Recursion Theory, its Generalizations and Applications*, London Math. Soc. Cambridge University Press.

Tucker, J. and J. Zucker (1992). Examples of semicomputable sets of real and complex numbers. In M. O'Donnell and J. Myers Jr. (Eds.), *Constructivity in Computer Science*, Volume 613 of *Lect. Notes in Comp. Sci.*, pp. 179–198. Springer-Verlag.

Turing, A. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc., Ser.2 42*, 230–265.

Tyurin, J. (1979). A survey of the logic of effective definitions. In E. Engeler (Ed.), *Logic of Programs*, Volume 125 of *Lect. Notes in Comp. Sci.*, pp. 198–245. Springer-Verlag.

Van der Waerden, B. (1949). *Modern Algebra*. F. Ungar Publishing Co.

von Neumann, J. (1963). *Collected Works,* V, A. Taub, editor. MacMillan.

Weihrauch, K. (1987). *Computability*. EATCS Monographs on Theoretical Computer Science, 9. Springer-Verlag.

Winograd, S. (1967). On the number of multiplications required to compute certain functions. *Proc. National Acad. Sci. 58*, 1840–1842.

Winograd, S. (1980). *Arithmetic complexity of computations*. SIAM Regional Conf. Ser. Appl. Math. 33.