Computational complexity theory and the philosophy of mathematics*

Walter Dean
Department of Philosophy
University of Warwick
Coventry, CV4 7AL UK
W.H.Dean@warwick.ac.uk

Abstract

Computational complexity theory is a subfield of computer science originating in computability theory and the study of algorithms for solving practical mathematical problems. Amongst its aims is classifying problems by their *degree of difficulty* – i.e. how hard they are to solve computationally. This paper highlights the significance of complexity theory relative to questions traditionally asked by philosophers of mathematics while also attempting to isolate some new ones – e.g. about the notion of *feasibility* in mathematics, the $\mathbf{P} \neq \mathbf{NP}$ problem and why it has proven hard to resolve, and the role of non-classical modes of computation and proof.

1 Introduction

Computational complexity theory is a subfield of theoretical computer science whose origins lie jointly in computability theory and in the much older study of algorithms for solving mathematical problems of practical import. One of its central aims is that of classifying mathematical problems according to their *degree of intrinsic difficulty* – i.e. according to how hard they are to solve using computational means which can be carried out in practice.

The fundamental objects which the theory seeks to classify are so-called *decision problems* which ask us to decide whether the members of a specified class of combinatorial objects possess a given property. Some examples are as follows:

PRIMES Given a natural number n, is n prime?

FACTORS Given natural numbers n and m, does there exist a divisor $1 < d \leq m$ such that $d \mid n$?

PERFECT MATCHING Given a bipartite graph G, does there exist a perfect matching in G?[1]

SAT Given a formula φ of propositional logic, does there exist a satisfying assignment for φ?

CHESS Given a board configuration in a generalized game of chess played on an $n \times n$ board (with one king per side), does there exist a winning strategy for white?

*Fothcoming in *Philosophica Mathematica*. Please do not cite without permission.

[1] A graph $G = \langle V, E \rangle$ is *bipartite* just in case its vertices can be partitioned into two disjoint sets U_1 and U_2 such that all of its edges E connect a vertex in U_1 to one in U_2. A *matching* is a subset of edges $M \subseteq E$ such that no two members share a common vertex. M is *perfect* if it matches all vertices.

In this context, a problem is understood to be "computationally complex" to the extent that its instances are difficult to decide algorithmically. But it is easy to see that all of these problems are *effectively decidable* – i.e. there exists an algorithm which can be implemented by a conventional Turing machine which always halts and returns a correct answer after a finite number of steps. They are hence equally difficult from the standpoint of classical computability theory. On the other hand, many of the problems studied in complexity theory arise in contexts in which we are interested not just in knowing that they may be effectively solved *in principle* but also whether specific instances can be solved *in practice* by current or foreseeable computing technology. For example, instances of PERFECT MATCHING arise when we wish to determine if it is possible to pair all the members of one set U_1 (e.g. job candidates, organ donors) with compatible members of another set U_2 (jobs, transplant recipients), as well as in a number of other contexts which arise in logistics, scheduling, and organic chemistry. On the other hand, the security of many common cryptographic protocols (including the well-known RSA system) depends on the hypothesis that the problem of integer factorization is difficult to solve in the general case.

It is thus of considerable practical importance that there exists a *polynomial time algorithm* for solving PERFECT MATCHING – i.e. one which runs in a number of steps proportional to a polynomial function of the number of edges and vertices in G. This has been traditionally taken as a touchstone of *feasible decidablity* – i.e. of decidability in practice rather than merely in principle. On the other hand, it is easy to see that the existence of such an algorithm for solving the decision problem FACTORS would also entail our ability to solve the general factorization problem efficiently. And it is thus also of practical significance that not only is there no known polynomial time algorithm for FACTORS but it is conjectured that no such algorithm can exist. One consequence of this is that there are concretely inscribable natural numbers (say of 1000 decimal digits) which can be safely employed as cryptographic keys in virtue of the fact that there is no current (or even foreseeable) combination of algorithms and computing machinery which would allow us to factor them.

Complexity theory is now a well-developed subject in computer science which draws on a rich array of techniques and results from logic, combinatorics, graph theory, abstract algebra, number theory, probability, game theory, and mathematical physics. It also has given rise to several high-profile open questions, of which the $\mathbf{P} \neq \mathbf{NP}$ conjecture is the best-known. One simple formulation of this statement is that there does not exist an algorithm for deciding SAT which is asymptotically more efficient than the "brute force" method of calculating the truth value of a propositional formula with respect to all possible evaluations of its propositional variables (i.e. the method of truth tables).

During the 1950s Kurt Gödel and John Nash independently formulated versions of $\mathbf{P} \neq \mathbf{NP}$ while also observing that its resolution would have far-reaching consequences not just for practical applications like cryptography, but also for the foundations of mathematics at large. But despite the considerable attention which this and related questions have attracted within computer science, there has been little philosophical engagement with computational complexity to date. This is true not only within philosophy of mathematics but also in subjects like epistemology, philosophy of language, philosophy of mind and cognitive science, social choice theory, and philosophy of physics wherein questions of computational difficulty also arise.

On potential reason for this is that complexity theory originated as a subject only in the early 1970s with the discovery of so-called **NP**-*complete problems* – i.e. ones which are complete for search problems similar to SAT in a manner analogous to how the classical Halting Problem is complete for recursively enumerable languages. This was more than 40 years removed from the foundational debates of the 1900s-1930s which inspired the developments in mathematical logic

with which philosophers are still most familiar. During the intervening period, computer science has indeed been somewhat less circumspect about its foundations than was mathematics during the early twentieth century. It is thus not surprising that there are currently few ready-made dialectics about computational complexity with which philosophers have elected to engage.

This diagnosis of the current relationship between philosophy and complexity theory will provide the frame for the rest of this paper. As I have already mentioned, one of the features which makes complexity theory interesting is its abundance of open technical questions. I will enumerate some of these below as $\mathrm{T1}, \mathrm{T2}, \ldots$ Both these questions and the definitions which lead to them are often accompanied by foundational questions to which it might reasonably be expected that a general theory of computational complexity should be responsive. These will be enumerated in the sequence $\mathrm{P1}, \mathrm{P2}, \ldots$ I will not attempt to provide systematic answers to many of these questions here. Rather they are intended to call attention to points where the standard technical development may appear to call for additional scrutiny or at which philosophers might hope to contribute either by applying complexity theoretic concepts and results within other subjects or by deepening our understanding of computational complexity itself.[2]

In service of this aim, the rest of this paper is structured as follows. In §2 I will seek to clarify the notions of *mathematical difficulty* and *computational feasibility* which complexity theory seeks to analyze while also surveying basic definitions and results. In §3, I will offer an account of the origins of complexity theory which illustrates how it may be understood as a natural outgrowth of Hilbert's program in the foundations of mathematics alongside proof theory and computability theory. In this context I will also discuss how the origins of complexity theory are historically linked to the related foundational view known as *strict finitism* and thereby also to broader philosophical debates about vagueness and the sorites paradox. In §4, I will return to discuss the current status of open separation problems such as $\mathbf{P} \neq \mathbf{NP}$ and how Gödel and Nash's predictions about their significance have borne out. In §5 I will explore a class of technical results collectively known as *barriers* which seek to explain why separation results between complexity classes have proven difficult to resolve. Finally in §6, I will attempt to position the topics which are treated in the prior sections – which pertain largely to modes of computation similar to those studied in classical computability theory – with respect to work on various "non-classical" models (e.g. randomized and quantum computation or probabilistic and interactive proof) which also play a significant role in contemporary theorizing about computational complexity.

2 Basic notions in complexity theory

Starting with (Hopcroft and Ulman 1979), introductory computability textbooks have typically included brief expositions of complexity theory. But as such treatments were originally targeted at a computer science audience, philosophers could for a long time be forgiven for neglecting the

[2]Since this survey is targeted at philosophers of *mathematics*, the questions in this sequence have thus been selected to engage with topics and debates within its extant literature (as well as that of philosophy of computer science). But as I have already noted, there are also potentially profitable connections to explore between complexity theory and topics in epistemology (e.g. the characterization of logical omniscience at its relation to *a priori* knowledge and rationality – see Cherniak 1986 and Dean 2016a, §4.7), philosophy of language (e.g. the semantics of vague predicates – see Dean 2018), philosophy of mind and cognitive science (e.g. the performance/competence distinction – see Pylyshyn 1984 and Pantsar 2019), social choice theory (e.g. the computational feasibility of strategic voting – see Bartholdi et al. 1989 and Brandt et al. 2016), and philosophy of physics (e.g. the role of computation in the interpretation of quantum mechanics – see Aaronson 2013b and Cuffaro 2018).

development of complexity theory. This situation has changed considerably with the publication of more accessible complexity theory textbooks (e.g. Papadimitriou 1994; Arora and Barak 2009; Goldreich 2010), popularizations (e.g. Harel 2006; Moore and Mertens 2011; Fortnow 2013), as well as recent surveys targeted specifically at philosophers (e.g. Urquhart 1998; Aaronson 2013a; **?**). Readers are directed to these sources for more complete definitions, a wider range of examples, and more extensive references than can be feasibly provided here.

2.1 Problems and mathematical difficulty

Perhaps the most fundamental notion of computational complexity theory is that of a *decision problem*. Such problems are treated similarly to the way they are introduced in computability theory – i.e. a problem X is regarded as a set in which we wish to decide membership, prototypically specified as the members of some superset Y comprising a class of finite combinatorial objects such as natural numbers, logical formulas, or graphs which satisfy a given predicate. There are, however, several differences in conventions adopted in the two subjects which should be stressed from the outset.

Consider, for instance, the problem SAT. This may be informally described as the subset of the set PROP consisting of all propositional formulas φ satisfying the predicate *there exists a satisfying valuation for* φ. It is conventionally assumed in complexity theory that problems are encoded as finite binary strings. This is accomplished in the general case by defining for the relevant type of object an injective coding function $\ulcorner\cdot\urcorner : Y \to \{0,1\}^*$ (i.e. the set of finite-lenth strings composed of 0s and 1s) such that both the encoding $\ulcorner x \urcorner$ of a problem instance and its decoding $\ulcorner x \urcorner^{-1}$ can be "easily computable" in a sense which will be clarified below (see also note 9). In the case of SAT, for instance, we can begin by defining $\ulcorner\varphi\urcorner$ to be a Gödel number for φ written in binary notation and then defining $\text{SAT} = \{\ulcorner\varphi\urcorner : \varphi \in \text{PROP} \;\&\; [\![\varphi]\!]_v = 1 \text{ for some propositional valuation } v\}$.[3]

In this way, decision problems are identified with sets of finite combinatorial objects, which are in turn understood to be encoded as sets of binary strings often referred to as *languages*. It is such languages which complexity theory then takes to be the primary bearers of both absolute and relative computational difficulty. There are programmatic reasons for each of these decisions which I will further discuss in a moment. But before doing so, it should also be acknowledged that the conventions in complexity theory reflect answers to certain basic questions about the meaning of the term "difficulty" – as well as cognates such as "hardness" or "complexity" itself – in mathematics. Among these are the following:

P1: To what sorts of objects do we ascribe difficulty (hardness, complexity, etc.) in the practice of mathematics?

[3]Here $[\![\varphi]\!]_v$ denotes the truth value of φ with respect to a function $v : \text{At} \to \{0,1\}$ which assigns a truth value $v(P_i)$ to each atomic formula $P_i \in \text{At}$. To make this initial example as concrete as possible, observe that a propositional formula φ may be considered as a finite string of symbols $\sigma_1, \ldots, \sigma_n$ over the alphabet $\Sigma = \{\wedge, \vee, \rightarrow, \neg\} \cup \{P_i : i \in \mathbb{N}\}$ in which each symbol is additionally separated by the symbol , (comma). Following Buss (1986), we may now define $\ulcorner\varphi\urcorner$ as follows: i) assign $\sigma \in \Sigma$ a code $c(\sigma) \in \{0,1\}^*$ via $c(\wedge) = 0, c(\vee) = 1, c(\rightarrow) = 10, c(\neg) = 11$ and $c(P_i) =$ *the binary numeral for* $i + 4$; ii) write the string $c(\sigma_1), \ldots, c(\sigma_n)$ in reverse order to obtain a string τ containing the symbols 0, 1, and ,; iii) let $\ulcorner\varphi\urcorner \in \{0,1\}^*$ be the result of replacing each instance of 0 in τ by 10, each 1 by 11 and each , by 00. If we assume that the subscripts of propositional letters appearing in φ are themselves written in binary notation, then it is easy to see that the length of $\ulcorner\varphi\urcorner$ will be linearly proportional to the number of symbols in φ and also that such a code may be computed in a number of steps proportional to the number of symbols in φ.

P2: Is it meaningful to speak of *degrees of difficulty* or to *compare* the difficulty of the relevant sorts of objects across different mathematical domains or subjects?

In regard to P1, perhaps the most familiar attributions of difficulty in mathematics are to *propositions* (or the sentences we take to express them) corresponding to theorems or conjectures. This is reflected by the fact that we often speak of propositions as "open problems" in their own right. We also occasionally grade the difficulty of problems understood in this manner – e.g. when we say that the Twin Primes Conjecture is a "difficult" open problem, while acknowledging that it is most likely "less difficult" than the Riemann Hypothesis. Such usage appears to extend to established theorems – e.g. when we say that the Infinitude of Primes is a "relatively easy" problem in number theory while the Four Squares Theorem, the Prime Number Theorem, and Fermat's Last Theorem are more "difficult", perhaps in increasing order.[4]

Much can be said about the basis of such judgements. But if we were pushed to identify one dimension which stands behind such attributions, a common answer would likely be that a statement ψ is "easy" or "hard" in proportion to how difficult it is was for the mathematical community to find a proof or refutation of ψ (in the case where ψ has been settled) or how hard the community expects it to be find such a demonstration (in the case that ψ is still open). Such an account takes a step towards an analysis of mathematical difficulty in terms of *proof search*

This in turn might raise the hope that the relevant notion of difficulty can be analyzed using tools from mathematical logic such as those of proof theory or Reverse Mathematics. But at least at present, there is no well-developed proposal of the form such a theory might take. One evident reason why this is so that it is largely foreign to mathematical practice to formalize statements and proofs (e.g. with respect to fixed logical signatures or axiom systems). And even when we do engage in formalization, it is not at all clear how the sorts of formal metrics studied in logic – e.g. the logical complexity of statements, the length or cut rank of derivations, the consistency strength or degree of constructivity of axioms – contribute to judgements of mathematical difficulty. In fact, it is often possible to marshal examples which suggest that such measures crosscut the sort of informal attributions of difficulty just canvased.[5]

These considerations by no means rule out the possibility that an account of what we might call *propositional difficulty* – i.e. the hardness of proving or refuting individual mathematical propositions – might eventually be developed. But they also serve to underscore several senses

[4]It is also notable that complexity theory has itself given rise to theorems and conjectures which are commonly described as "difficult" in the same sense – e.g. $\mathbf{P} \neq \mathbf{NP}$ was famously included on the list of Millennium Problems along with the Poincaré Conjecture and the Riemann Hypothesis (Carlson et al. 2006). In fact $\mathbf{P} \neq \mathbf{NP}$ may be understood as making a claim about the notion of mathematical difficulty itself – i.e. that the problem of settling open questions is hard because there is no *general* method which performs better than exhaustive search for determining whether a statement is provable or refutable from a given set of axioms by a proof which is short enough for us to comprehend. As I will discuss further in §4 and §5, $\mathbf{P} \neq \mathbf{NP}$ thus itself expresses a claim about *problem difficulty* (as defined below). But it is perhaps for this very reason why it has proven so hard to resolve in the sense of *propositional difficulty*.

[5]To take just one example, consider Dirichlet's Theorem on arithmetical progressions – i.e. if m and k are relatively prime, then the sequence $m, m+k, m+2k, \ldots$ contains infinitely many primes. There are several senses in which this statement was (and presumably still is) "hard" – e.g. after being conjectured by Legendre in 1798 it stayed open for 40 years (eluding Gauss, amongst others) and its resolution required the development of new methods in analytic number theory (see, e.g. Avigad and Morris 2016). On the other hand, the statement of the theorem itself is equivalent to a Π^0_2-sentence of first-order arithmetic and hence (relatively) "simple" in terms of logical complexity. Cegielski (1992) also showed that that the proof of Dirichlet's theorem is formalizable in RCA_0 (one of the weakest systems studied in Reverse Mathematics) and thus also in Primitive Recursive Arithmetic (by the Π^0_2-conservativity of RCA_0 over PRA). This suggests that Dirichlet's theorem is (relatively) "easy" in the verificational sense of not requiring "strong" or "non-constructive" axioms.

which distinguish this notion from what I will call *problem difficulty* which is studied in complexity theory.[6] This notion applies to problems X (as defined above) and comes with a more established gloss: a problem X is complex (or hard) to the extent that it is *difficult to solve by algorithmic means*. Making sense of this initial account still requires that we provide an account of how we can quantify over algorithms and measure how difficult they are to carry out. But however these tasks are accomplished, it is then this understanding of a problem as a typically *infinite* set of instances which makes possible a further analysis of how problem difficulty can be measured and compared. For presuming that we are able to additionally to give an account of the *size* $|x|$ of a problem instance $x \in Y$, then we can analyze the efficiency of an algorithm A for deciding $X \subseteq Y$ as a *function* $f : \mathbb{N} \to \mathbb{N}$ which relates $|x|$ to some metric of the *computational resources* needed to carry out A on x. It is then the *order of growth* of $f(|x|)$ (see §2.2) which is traditionally referred to as the *computational complexity* of X.

These considerations also help to explain why binary encodings of problem instances are employed in complexity theory. For note that some uniform representation of problem instances must be employed so that different forms of mathematical objects can serve as inputs to a formal model of computation, which are in turn used to implement decision algorithms. Once this step is undertaken, it is then possible (at least in principle) to provide a positive answer to question P2 – e.g. formal results can be cited to define a precise sense to the claim that Primes and Perfect Matching are of the same degree of difficulty while they are both easier than Sat (presuming $\mathbf{P} \neq \mathbf{NP}$) and Chess (even without the need for such an assumption).[7]

The adoption of binary codes also makes it possible to represent a finite mathematical object as a string whose length is proportional to the parameters which are often most useful in gauging the efficiency of decision algorithms. A canonical example is provided by the familiar practice of using *positional* (e.g. binary or decimal) *numerals* to denote natural numbers. Such numerals correspond to finite sequences of symbols $d_{i_0} \dots d_{i_{k-1}}$ where each symbol (or *digit*) is drawn from a finite set of distinct symbols $\{d_0, \dots, d_{b-1}\}$ where $b \geq 2$ is the *base* and the denotation $[\![d_i]\!]$ of d_i is understood to be the natural number i. While the *length* of a positional numeral $d_{i_0} \dots d_{i_{k-1}}$ is equal to the number k of digits it contains, its *value* is given by the sum $\sum_{j=0}^{k-1} [\![d_{i_j}]\!] \cdot b^{k-j-1}$ (where we additionally

[6] What is here called propositional difficulty is akin to what Detlefsen has referred to as the *inventional* or *discovermental complexity* of a given mathematical statement – i.e. that which is "encountered in coming up with a proof in the first place" (1990, p. 376, see also Detlefsen 1996, p. 87). However Detlefsen contrasts this notion not with problem difficulty (as defined here) but rather with "verificational complexity" – i.e. "the type of complexity that is encountered in determining of a given syntactical entity whether or not it is a proof in a given system of proofs". Understood in this sense, it is often easy (in the sense of problem difficulty) to decide whether a given (putative) derivation $\mathcal{D}$ is a correct proof of a statement φ from the axioms of a given mathematical theory T – i.e. it suffices to check if each formula in $\mathcal{D}$ is either an axiom of T or follows from prior statements by logical rules. This can typically be accomplished in polynomial time in relative to size of $\mathcal{D}$ provided that the problem of deciding axiomhood in T may itself be decided in polynomial time (a feature which is satisfied by familiar recursively axiomatizable theories such as PA or ZF). But of course one might additionally attempt to locate other dimensions of either verificational or discovermental complexity – e.g. that of finding axioms T which allow an elucidatory (or otherwise non-trivial) proof in the first place, various measures of the "ideality" of its axioms (e.g. logical complexity, possibly understood as a proxy for the difficulty of determining whether they *true* of a relevant domain), the novelty of the methods which are employed either employed in initial proof of φ from T (or are subsequently determined to be required), whether such a proof requires "impure" methods, whether such methods can be shown to be eliminable, etc. See, e.g., (Detlefsen and Arana 2011) for further discussion of some of these issues.

[7] Note that this is so despite the fact that these problems are respectively defined on natural numbers, graphs, formulas, and board configurations. This is again in apparent contradistinction to attributions of propositional difficulty where our intuitions of relative difficulty appear to become even more tenuous when we attempt to compare the relative difficulty across subfields of mathematics.

assume that the leading digit does not denote 0 – i.e. $[\![d_{i_0}]\!] \neq 0$). It thus follows that the length of a positional numeral will be exponentially shorter than the value it denotes.

The properties of positional notations are also exploited by many of the numerical algorithms which we employ in practice – e.g. carry addition or long division. These procedures operate in a digit-by-digit manner requiring a number of steps which is proportional to the length of their inputs (typically at worst quadratically) rather than than their values. It is for this reason that we are able to compute in practice the values of sums and product of various "large" natural numbers – e.g. 25,211,713,952,371,115,192,327 and 75,557,863,725,914,323,419,121 – despite the fact that it is infeasible to count up to them (by ones) or to inscribe co-denoting unary numerals of the form $|, ||, |||, \ldots$ in practice.

The distinction at issue can also also be vividly illustrated by considering the so-called *trial division algorithm* for deciding PRIMES – i.e. on input $x \in \mathbb{N}$, check if $2 \mid x, 3 \mid x, \ldots, (x-1) \mid x$, returning "no" if a factor is found and "yes" otherwise. Since each of the divisibility checks can be performed efficiently (e.g. by long division) and there are only on the order of x checks to be made, it might at first appear that this is not a difficult task to perform relative to the value x itself. But since we typically denote natural numbers using positional notation, it follows that there will be many numbers which we can concretely inscribe in practice and hence might hope to check for primality – e.g. the ones displayed in the prior paragraph – but for which we have no hope of carrying out this procedure in practice. For note that although trial division operates in a number of steps linearly proportional to x, its performance is only *exponentially* proportional to the length of x's representation in positional notation.[8] As we will see, this sort of contrast is typical of those encountered in complexity theory.

Analogous observations apply to our computational practices involving (e.g.) logical formulas, graphs, matrices, or groups in terms of which other problems studied in complexity theory are defined. We often do not distinguish such objects as sharply from their syntactic representations as we do in the case of natural numbers. Nonetheless, it is also typically possible to construct binary representations which are easy to compute and which transparently encode the relevant structural properties of the objects which are most relevant to measuring the performance of algorithms which operate on them. This in turn provides further practical justification for the convention of identifying problems with sets of binary strings.

As I will discuss further in §3, reflection on the role of different notation systems played an important role in the early history of complexity theory. But the considerations just described also draw attention to the extent to which apparently arbitrary decisions must be made to encode finite mathematical objects as binary strings – a point which is vividly illustrated by example in note 3. This in turn prompts questions such as the following:

P3: Is it possible to alter what we might take to be the *intrinsic* difficulty of a problem defined directly on such objects by using different computable encodings?[9] In the face of this concern

[8] This is the hallmark of a so-called *psuedo-polynomial time algorithm* – i.e. one whose time complexity is proportional to a polynomial function of x rather than to $|x|$ (see §2.4). There are, of course, many obvious improvements to trial division – e.g. we only need to check for divisors up to $\sqrt{x}$ rather than $x-1$, by using sieve methods, etc. But such refinements only appear capable of reducing the complexity of trial division to $O(2^{\sqrt{n}})$ which is still considered an infeasible rate of growth. It was thus of considerable import when a polynomial time algorithm for PRIMES – i.e. the so-called *AKS primality algorithm* – was finally discovered by Agrawal et al. (2004).

[9] The possibility of employing even more radically "deviant" encodings of inputs and outputs which transform an undecidable problem on (say) natural numbers into a decidable one on (say) binary strings has long been a topic of philosophical debate in regard to the need to interpret the operation of models of computation which operate on

is it possible to provide a general account of the role of notation in discrete mathematics – e.g. one that subsumes symbolisms for denoting objects like numbers, formulas, proofs, graphs, matrices, groups, etc. – which accounts for why we find certain schemes or conventions for representing such objects more useful in practice than others? How is this related to the recent discussion of our apparent ability to form *de re* beliefs (or other propositional attitudes) about individual natural numbers – e.g. in regards to Kripke's notion of referential "buck-stoppers"?[10]

It should finally be noted that although decision problems are most central in complexity theory, its methods can also be applied to study the computational difficulty of other sorts of mathematical problems. Amongst these are *function problems* – e.g. find an algorithm to compute a given function $f : X \to Y$ – *search problems* – e.g. given $\varphi \in \text{PROP}$, find a satisfying assignment $v : 2^n \to \{0, 1\}$ (if one exists) and output "no" otherwise – and *optimization problems* – e.g. find $x \in Y$ which maximizes (or minimizes) a given cost function $c(x)$. These sorts of problems are often related to decision problems – e.g. given an optimization problem of the form just stated we can also consider the decision problem "is there an $x \in Y$ such that $c(x) \geq k$?". There are, however, cases where a decision problem is known to be trivially solvable, but the corresponding search or optimization problem is believed to be difficult in the general case. And it is often the case that the question "is there an $x \in Y$ such that $c(x) \geq k$?" is apparently easier to resolve than "does $\max_{x \in X} c(x) = k$?' (as no single witness to the former problem is sufficient to provide an affirmative answer to the latter).[11]

symbols (such as Turing machines) in order to argue for Church's Thesis understood as an hypothesis about which *numerical* functions can be effectively computed (e.g. Shapiro 1982; Rescorla 2015; Shapiro 2017). Nonetheless, this concern has been isolated within philosophical work and has had little influence on the practice of computability theory. Three observations help to explain why complexity theorists have been similarly oblivious to such skeptical concerns: 1) as we have just seen, the theory itself helps to explain why we often adopt an "inequalitarian attitude" about the relative merits of different means of representing mathematical objects – e.g. by offering an account of why it is more useful in our computing practices to represent natural numbers using positional rather than unary numerals; 2) for this reason, it seems that complexity theorists will have little reason to object to the claim that the objects which are manipulated in our most fundamental account of computation are "syntactic" entities (like strings) rather than "mathematical" ones (like natural numbers); 3) in instances where it is necessary to explicitly encode inputs and decode outputs in order to make use of the results of a computation, complexity theorists are also in a position to make use of practical considerations to rule out the envisioned sort of complexity-distorting encodings – e.g. by demonstrating that they are infeasible to compute relative to a hybrid model of computation which operates on (say) both strings and numbers or propositional formulas simultaneously. See (Matthews and Dresner 2017) for a recent articulation of a similar point contra related skeptical concerns about the interpretation of physical implementations of computers of the sort originally introduced by Putnam (1988) and **?**.

[10] According to Kripke, a given term τ is a "buck-stopper" for an agent if it functions as a sort of canonical name in the sense that when the agent is presented with τ they will no longer be disposed to ask "What object does τ denote?" This notion is introduced in Kripke's 1992 Whitehead Lectures wherein he also makes the additional claims: 1) decimal numerals – e.g. 17 as opposed to (say) unary numerals like |||||||||||||||| or polynomial expressions like $7^4 - 13^3 + 6^2 - 223$ – provide such notations for most agents; 2) an agent's ability to form a *de re* belief about a natural number is mediated by the availability of such representations; 3) absent an account of mathematical reference which is responsive to the phenomenaon of "buck-stopping" – and is presumably compatible more generally with a non-trivial answer to the second question raised by P3 – it is difficult to explain how in the practice of computability and complexity theory we typically find statements of the form $f(n) = k$ (where n and k are decimal numerals and $f(x)$ is, e.g., a notation for a recursive function – e.g. factorial(7) = 5040) more informative than syntactic identity statements of the form $f(n) = f(n)$. (Although Kripke's lectures remain unpublished, a partial account of these considerations is given by Steiner (2011) and, in passing, by Kripke 2011, p. 344.)

[11] An example of the former sort is that of finding a Nash equilibrium in a non-cooperative game represented as a payoff matrix: although Nash's Theorem guarantees that equilibria always exist, Daskalakis et al. (2009) provide evidence that there is no polynomial time algorithm for finding one in the general case. One the other hand, if **P** = **NP** then it is not difficult to show that for all languages $X \in \mathbf{NP}$, there is a polynomial time algorithm which

It is thus often more convenient to take the decision version of a problem as basic for purposes of complexity classification and then consider the function or optimization variants as special cases.

2.2 Complexity measures

Once the notion of a decision problem is in place, the next steps in the technical development of complexity theory are to introduce a formal *model of computation* $\mathfrak{M}$ by which informally specified decision algorithms can be implemented and to identify various parameters of this model as formal representations of *computational resources*. Although we will see below there is some flexibility in this regard, it is conventional to assume that $\mathfrak{M}$ corresponds to the familiar multi-tape Turing machine model $\mathfrak{T}$ with the additional proviso that it is assumed that the tape alphabet includes at least the symbols $0, 1$ (in addition to the blank symbol $-$) to account for the need to supply binary strings as inputs.[12]

The two most basic computational resources which are studied in complexity theory are *running time* and *tape* (or *memory*) *space*. These are defined for a given Turing machine $T \in \mathfrak{T}$ in three stages. First, the functions $\text{time}_T, \text{space}_T : \{0,1\}^* \to \mathbb{N}$ are defined which respectively return the number of steps taken and the number of tape cells visited on an input $x \in \{0,1\}^*$ if T halts on x and are undefined otherwise. Second, a *size measure* $|\cdot| : \{0,1\}^* \to \mathbb{N}$ is defined which returns a parameter of problem instances which determines the efficiency of algorithms for deciding X with which we are concerned in practice – e.g. for PRIMES or FACTORS, $|\cdot|$ might be the length of a number's representation in binary notation (i.e. $|x| = \lceil \log_2(x) \rceil$) or in the case of SAT $|\cdot|$ might be the number of propositional variables in a formula φ. Third, the measures $\text{time}_T(x)$ and $\text{space}_T(x)$ are transformed into functions of type $\mathbb{N} \to \mathbb{N}$ by defining $t_M(n) = \max\{\text{time}_T(x) : |x| = n\}$ and $s_M(n) = \max\{\text{space}_T(x) : |x| = n\}$ – i.e. the *worst case* time and space complexity of T defined as the maximum number of steps or tape cells visited during T's computation for all inputs x of size n. These measures are then used to classify problems into *complexity classes* as described in §2.3.

The choice of $\mathfrak{T}$ as a reference model for measuring computational complexity may at first appear arbitrary if our ultimate aim is to provide an analysis of a notion of difficulty which is *intrinsic* to a problem rather than to a particular means of solving it. The explanation which is traditionally given in favor of the adequacy of $\mathfrak{T}$ for this task can also be reconstructed in three steps:

1) An account is given to justify the fact that in adopting $\mathfrak{T}$ as a reference model, we have not been *overly liberal* in characterizing what can be computed in a single step.

2) A complementary account is given that we have not been *overly conservative* in measuring complexity in terms of basic Turing machine operations.

3) A comparison of $\mathfrak{T}$ with other models of computation is undertaken to provide further confirmation of the arguments given for 1) and 2).

With respect to the first step, an obvious concern is that by using a particular model to measure time and space complexity we are assuming its basic operations are computable in a primitive, molar

not only decides X but also finds the appropriate sort of object to witness this fact – e.g. a satisfying assignment for φ if $X = \text{SAT}$.

[12] See (van Emde Boas 1990) for a precise specification of $\mathfrak{T}$ and the other models and results mentioned in this section.

sense – i.e. without further computational decomposition.[13] But for reasons which Turing already made clear in his original (1936) paper, $\mathfrak{T}$ only faces this problem to a minimal extent. For not only are its basic read/write/move operations clearly effective in both the intuitive and concrete senses (i.e. we can build mechanical devices which can implement them), Turing also provides an account of why these operations are best understood as primitives (e.g. in terms of the finite divisibility of the tape or distinguishability of symbols). One benefit of using an elementary model such as an $\mathfrak{T}$ as a benchmark for time and space complexity is that there is little risk that it sweeps the costs of concretely embodied computation under the carpet.

The second step builds on part of the evidence which is commonly adduced for *Church's Thesis* – i.e. the claim that the effectively computable functions on the natural numbers coincide with the recursive ones, and thus also those computable by machines in $\mathfrak{T}$. One sort of argument often given in favor of this hypothesis is the observation that it is generally possible to transform the informal descriptions of algorithms which are generally given in mathematical practice into specifications of Turing machines. This is true despite the fact that algorithms are typically specified in textbooks or journal articles in terms of operations which cannot be carried out by a Turing machine in a single step – e.g. computing products or quotients (e.g. Euclid's algorithm), manipulating terms in polynomials (e.g. Sturm's algorithm), or adding or removing edges from a graph (e.g. Kruskal's algorithm). But in each such case, experience has borne out that with sufficient ingenuity it is always possible to mimic using the primitives made available by the Turing machine model not only these "high level" objects and operations, but also the data structures (e.g. lists, stacks, queues) and flow control devices (e.g. loops and recursion) which are used in informal specifications of algorithms.

For reasons discussed in (Dean 2016b), the process of representing such "high level" structures and processes relative to a fixed model of computation raises substantial concerns as to whether the identity conditions of informally specified algorithms are preserved under the process of implementation. This in turn suggests that there may be difficulties in regarding algorithms – as opposed to their implementations – as mathematical objects in their own right. But such issues are not of immediate concern to complexity theory. For although the initial analysis given in §2.1 of "the computational complexity of problem X" does quantify over all possible algorithms for deciding X, what matters for classifying of X with respect to the complexity classes which will be introduced in §2.4 is merely that the process of implementation preserves asymptotic time and space complexity.

Accounting for the particular complexity costs of $\mathfrak{T}$ in relation to other models corresponds to the third step in the rationale for its adoption as a reference model. In order to explain how this is accomplished, it will be useful to also recall *order of growth notation*. Given a function $f : \mathbb{N} \to \mathbb{N}$ we define its order of growth to be

$$O(f(n)) = \{g(n) : \exists c \exists n_0 \forall n \geq n_0 (g(n) < c \cdot f(n))\}$$

– i.e. the set of all functions which are *asymptotically bounded* by $f(n)$ ignoring constant scalar and additive factors.[14] It is a convention of informal algorithmic analysis that worst case running time or space complexity are reported using such notation and also that efficiency comparisons are reported in these terms – e.g. the Mergesort algorithm is taken to be more efficient that the "naive"

[13] For instance as Cobham (1965) observed *en route* to providing his original definition of polynomial time, it would be of little use to employ a model of computation which already took multiplication as such a basic operation if our goal was to analyze the difficulty of computing products relative to that of computing sums.

[14] By way of illustration, the functions $\log_2(n), n$ and $10^{100} \cdot n^2 + 10^{100}$ are all in $O(n^2)$ but $10^{-100} \cdot n^3 \notin O(n^2)$. Note also that any polynomial in the single variable n of order k is in the class $O(n^k)$.

sorting method known as Insertionsort because the time complexity of the former is $O(n \log_2(n))$ while that of the latter is $O(n^2)$ (where n is the length of the list being sorted).

The sort of analysis just alluded to is typically carried out by counting the number of "high level" operations performed by the algorithms in question relative to their informal specifications – e.g. comparison between list elements in the case of sorting algorithms or calculation of moduli in the case of Euclid's algorithm. But although Turing machines cannot perform such operations in a single step, experience also bears out that the sorts of implementations of algorithms described above are typically only a polynomial factor less efficient in terms of time or space complexity than their informally specified counterparts.[15] A related discovery of the 1960s is that the Turing machine model is also capable of efficiently simulating a wide class of other models of computation $\mathfrak{M}$ in the sense that if a function $f(x)$ is computable in time $t_M(n) \in O(f_1(n))$ and space $s_M(n) \in O(f_2(n))$ relative to some machine $M \in \mathfrak{M}$, it will also be computable by a machine $T \in \mathfrak{T}$ with $t_T(n) \in O(f_1(n)^{k_1})$ and $s_T(n) \in O(f_2(n)^{k_2})$ for fixed $k_1, k_2 \in \mathbb{N}$ – in fact typically $k_1, k_2 \leq 3$. This includes not only a variety of generalizations of $\mathfrak{T}$ itself – e.g. with multi-dimensional tapes or additional structures like stacks or queues – but also as the familiar *Random Access Machine* [RAM] model $\mathfrak{R}$ which allows for addition and subtraction to be performed on registers whose values can be retrieved and stored in a single step.

It is often possible to implement informally specified algorithms more directly using $\mathfrak{R}$ which in turn more closely resembles the architecture of contemporary digital computers. This motivates the following refinement of the general notion of a model of computation employed in computability theory due to van Emde Boas (1990): a model $\mathfrak{M}$ is in the *first machine class* just in case it can be simulated by $\mathfrak{T}$ with polynomial time and space overheads as just defined. As we will see in §2.3, the notion of *polynomial time computability* has proven to be central to the development of complexity theory. And what has thus turned out to be of most importance for its development is that time and space complexity functions are defined relative to *some* model in this class rather than the particular details of the Turing machine model itself.

The final stage in the traditional argument that it suffices to take $\mathfrak{T}$ as a reference model is thus the observation that the time and space complexity of models drawn from the first machine class provide an accurate gauge on the exigencies which we experience in computational practice. One means by which this can be accomplished is to consider models which further liberalize the definitions of $\mathfrak{T}$ or $\mathfrak{R}$ but which are still effective in the sense that they do not allow for the computation of non-recursive functions. Paradigmatic of such formalisms are models of parallel computation such as the so-called *PRAM* model $\mathfrak{P}$. During the course of a single step in its computation, a machine $P \in \mathfrak{P}$ may recruit a fixed number $k \geq 2$ of additional RAM-like machines which subsequently operate in parallel on a shared memory. Models of this sort comprise the so-called *second machine class*. A striking feature of such models is that they contain machines which solve problems we find infeasible to decide in practice in polynomial time – e.g. the obvious parallel algorithms for Factors or Sat have a polynomial time implementation with respect to $\mathfrak{P}$. And it is for this reason that

[15] It is, of course, also possible to specify intuitively effective procedures for solving the sorts of problems exemplified above in linear or even in constant time in cases where we suspect that there is no corresponding Turing machine with polynomial running time – e.g. by assuming that it is possible to determine whether a number is prime or a formula is satisfiable in a single unmediated step. But there is a strong tendency to refrain from doing this in the practice of algorithmic analysis. As is also discussed further in (Dean 2016b), this in turns suggests that there is a more delicate balance between the sorts of operations on discrete mathematical structures which we are willing to treat as primitive in our informal practices and the details of the model $\mathfrak{T}$ and similar models than it might at first appear.

members of the second machine class have traditionally been excluded from the class of so-called *reasonable* models of computation on which time and space measures intended to accurately reflect computational practice are based.[16]

The prior observations are at least suggestive that $\mathfrak{T}$ should be regarded as a reasonable model while $\mathfrak{P}$ should not. But they also point towards another foundational question which can be posed independently of the technical development of complexity theory:

P4: Do our practices provide an independent characterization of a *reasonable* computational operation or procedure which refines that of an *effective* operation or procedure? Can such considerations be used to provide a model-independent characterization of the first machine class?[17]

As I will discuss further in §2.5, these question point towards the goal of articulating and defending an analog of Church's Thesis for the notion of feasible computability. But it should also be noted that our intuitions about "reasonableness" already bring us into contact with an open technical question in complexity theory:

T1: Is $\mathfrak{T}$ already contained in the second machine class?

Relative to the formal definition of parallelism give by van Emde Boas (1990), the expected negative answer to T1 is equivalent to the hypothesis $\mathbf{P} \neq \mathbf{PSPACE}$. But as will be discussed below, while weaker than the $\mathbf{P} \neq \mathbf{NP}$ conjecture, this question is also currently unresolved.

2.3 Non-determinism

The next step in the standard development of complexity theory is the introduction of so-called *non-deterministic models of computation*. The motivation for such models can be illustrated by considering the following procedure for deciding membership in the problem SAT, which I will refer to as NSat:

1) On input $\varphi \in$ PROP, make a list of the propositional variables $X_0, \ldots, X_{k-1}$ which it contains.

2) Choose a finite valuation function $v : \{X_0, \ldots, X_{k-1}\} \to \{0, 1\}$.

3) Using the method of truth tables, compute the valuation $[\![\varphi]\!]_v$ and output "yes" if $[\![\varphi]\!]_v = 1$ and "no" otherwise.

NSat is what is known as a *non-deterministic algorithm* – i.e. rather than specifying at step 2) how the valuation v is determined, it is left to the agent (or device) carrying out the procedure to make a decision (or "guess") as to the truth values it assigns. As there are only finitely many functions of type $\{X_0, \ldots, X_{k-1}\} \to \{0, 1\}$, such a determination can be effectively constructed in

[16] A further rationale for this convention is provided by various results which suggest that it is not possible to physically realize models in the second machine class in a manner such that a single physical "step" in a concretely embodied computation corresponds to a single abstract state transition – see, e.g. (Chazelle and Monier 1983), (Schorr 1983), (Vitányi 1988). But as I will discuss further in §6, however, the question of whether the notion of "reasonableness" most relevant to complexity theory is best regarded as a conceptual one (similar to that *effectivity*) or an empirical one (similar to that of *physical possibility*) is perhaps best regarded as currently open in virtue of (e.g.) current debate about the status of models of quantum computation.

[17] See (Dean 2016c) for an attempt to do so via a variant of Kreisel's (1967a) "squeezing argument".

time proportional to k. Once such a choice for v has been made, the procedure then continues in a conventional deterministic manner at step 3). And as should be evident, NSat is a *correct* algorithm for SAT in the sense that if φ is satisfiable, then there will exist an appropriate choice such that it outputs "yes" and if φ is unsatisfiable all choices will lead to the output "no".

One way in which NSat can be employed to test φ for satisfiability is thus to enumerate all of the (finitely many) functions $v_0, v_1, \ldots$ of type $\{X_0, \ldots, X_{k-1}\} \to \{0,1\}$ and check for each if $v_i(\varphi) = 1$. Such a satisfying valuation is what is known as a *certificate* for φ's satisfiability – i.e. an object which is small relative to the size of φ and such that it can be used to easily confirm φ's membership in SAT (as computing $[\![\varphi]\!]_v$ given v can also be accomplished efficiently via truth tables). But since there are 2^k functions of the relevant type which must be checked, this sort of procedure has traditionally been referred to as a "brute force search". For instance it is evident that we cannot employ this method in practice to uniformly decide the satisfiability of formulas even in the case where k is relatively small (say on the order of 50).

As I will discuss further in §2.6, an observation which was important in the early development of complexity theory is that there are many problems of practical import for which brute force search seems to be unavoidable. This in turn raises the question:

P5: Does the notion *brute force search* occupy an independent status in our practices distinct from the traditional notion of an effective deterministic procedure?[18]

The study of brute force search is one of the considerations which motivate the introduction *non-deterministic* models of computation in complexity theory. The standard formalism employed for this purpose is the non-deterministic Turing machine model $\mathfrak{N}$. This model is derived from $\mathfrak{T}$ by relaxing the requirement that the transition mapping Δ of a machine is a (partial) function to merely require that it is a relation – i.e. instead of requiring that Δ relates each state-symbol pair $\langle q, \sigma \rangle$ with at most one state-action pair $\langle q, \alpha \rangle$, it is now allowed that Δ may relate $\langle q, \sigma \rangle$ to two or more distinct state-action pairs. Recall also that if $T \in \mathfrak{T}$ is a deterministic machine, its operation on a given input x is uniquely determined by a (finite or infinite) sequence of configurations $C_0(x), C_1(x), C_2(x), \ldots$ determined by iterating its transition relation. However if $N \in \mathfrak{N}$, there may be more than one configuration which is related to the current configuration $C_i(x)$ by Δ. In this case, a sequence $C_0(x), C_1(x), C_2(x), \ldots$ is said to be a computation sequence for N just in case for all $i \geq 0$, $C_{i+1}(x)$ is among the configurations which are related by Δ to $C_i(x)$.

With these conventions in place, we now also define what it means for a non-deterministic machine to N to *decide a language* X:

i) N always halts – i.e. for all initial configurations $C_0(x)$, the computation sequence beginning with $C_0(x)$ is of finite length.

ii) If $x \in X$ then *there exists* a computation sequence $C_0(x), C_1(x), \ldots, C_n(x)$ of N such that $C_n(x)$ is an accepting state.

iii) If $x \notin X$, then *all* computation sequences $C_0(x), C_1(x), \ldots, C_n(x)$ of N are such that $C_n(x)$ is a rejecting state.

[18] Some evidence to this effect is provided by the fact that a systematic study of *perebor* (which is typically translated as "brute force search") was undertaken in the Soviet Union, largely in isolation from the other developments surveyed here. Starting with the work of Yablonskii in the 1950s on circuit complexity, this would lead to Levin's independent formulation of the notion of **NP**-completeness (see Trakhtenbrot 1984).

We now also define the time and space complexities $t_N(n), s_N(n)$ of a non-deterministic machine N to be the *maximum* length of any possible computation for N or number of tape cells visited for all inputs x such that $|x| = n$.

Note that the prior definition treats accepting and rejecting computations *asymmetrically*. For if $x \in X$, some of N's computation sequences starting from $C_0(x)$ may still lead to rejecting states as long as at least one leads to an accepting state. On the other hand, if $x \notin X$, then *all* of N's computations from $C_0(x)$ are required to lead to rejecting states. It is, of course, possible to transform a non-deterministic machine into a deterministic one deciding the same language either by using a deterministic machine $T \in \mathfrak{T}$ to implement the brute force search strategy described above or a parallel machine $P \in \mathfrak{P}$ to test all of the relevant alternatives at once. But for reasons which should now be clear, in neither case is the resulting implementation of much use in deciding problems like SAT in practice.

This in turn might lead one to expect that $\mathfrak{N}$ is an "unreasonable" model of computation in the sense discussed above. But because of the asymmetry of the acceptance/rejection conditions in its definition, it is not entirely straightforward to locate $\mathfrak{N}$ with respect to the distinction between the first and second machine classes. Rather, its primary role has been that of serving as a model by which *non-deterministic complexity classes* can be defined. And it is to the definition of such classes which we now turn.

2.4 Complexity classes

With the foregoing definitions in place, we can now define the general notion of a *complexity class*. First recall that we say that an instance of a model of computation M *decides* a language X just in case it computes its characteristic function. If $t(n)$ and $s(n)$ are functions of type $\mathbb{N} \to \mathbb{N}$, we first define the classes $\mathbf{TIME}(t(n))$ and $\mathbf{SPACE}(s(n))$ relative to the Turing machine model to be the set of languages which are decided by some $T \in \mathfrak{T}$ respectively in time $t(n)$ and space $s(n)$ – i.e.

$$\mathbf{TIME}(t(n)) = \{X \subseteq \{0,1\}^* \ : \ \exists T \in \mathfrak{T} \forall n (t_T(n) \leq t(n)) \ \text{ and } \ T \text{ decides } X\}$$

$$\mathbf{SPACE}(s(n)) = \{X \subseteq \{0,1\}^* \ : \ \exists T \in \mathfrak{T} \forall n (s_T(n) \leq s(n)) \ \text{ and } \ T \text{ decides } X\}$$

Since any polynomial in n is in the class $O(n^k)$ for some k, we also define the classes *polynomial time* and *polynomial space* are respectively as $\mathbf{P} = \bigcup_{k \in \mathbb{N}} \mathbf{TIME}(n^k)$ and $\mathbf{PSPACE} = \bigcup_{k \in \mathbb{N}} \mathbf{SPACE}(n^k)$. It is also standard to introduce names for the classes $\mathbf{EXP} = \bigcup_{k \in \mathbb{N}} \mathbf{TIME}(2^{n^k})$ (*exponential time*), $\mathbf{EXPSPACE} = \bigcup_{k \in \mathbb{N}} \mathbf{SPACE}(2^{n^k})$ (*exponential space*) and $\mathbf{L} = \mathbf{SPACE}(\log(n))$ (*logarithmic space*).

We can also define analogous non-deterministic complexity classes based on the acceptance conventions for the model $\mathfrak{N}$ given in §2.3 as follows:

$$\mathbf{NTIME}(t(n)) = \{X \subseteq \{0,1\}^* \ : \ \exists N \in \mathfrak{N} \forall n (t_N(n) \leq t(n)) \ \text{ and } \ N \text{ decides } X\}$$

$$\mathbf{NSPACE}(s(n)) = \{X \subseteq \{0,1\}^* \ : \ \exists N \in \mathfrak{N} \forall n (s_N(n) \leq s(n)) \ \text{ and } \ N \text{ decides } X\}$$

The classes **NP** (*non-deterministic polynomial time*), **NPSPACE** (*non-deterministic polynomial space*), **NEXP** (*non-deterministic exponential time*), and **NL** (*non-deterministic logarithmic space*) are defined analogously to **P**, **NP**, **EXP** and **L** – e.g. $\mathbf{NP} = \bigcup_{k \in \mathbb{N}} \mathbf{NTIME}(n^k)$.

Many results and open questions in complexity theory concern the inclusion relationships among these classes. Central among the former are the so-called *Hierarchy Theorems* which demonstrate

that the classes $\mathbf{TIME}(t(n))$ form a proper hierarchy in the sense that if $t_2(n)$ grows sufficiently faster than $t_1(n)$, then $\mathbf{TIME}(t_2(n))$ is a proper superset of $\mathbf{TIME}(t_1(n))$, and similarly for $\mathbf{SPACE}(s(n))$ and $\mathbf{NTIME}(t(n))$. For instance, a modified form the diagonal argument by which Turing originally showed the undecidability of the classical Halting Problem for $\mathfrak{T}$ can be used to show that for $k \geq 1$, $\mathbf{TIME}(n^k)$ is always a proper subset of $\mathbf{TIME}(n^{k+1})$ and $\mathbf{SPACE}(n^k)$ is always a proper subset of $\mathbf{SPACE}(n^{k+1})$. This in turn can be used to show that $\mathbf{P} \subsetneq \mathbf{EXP}$ and $\mathbf{L} \subsetneq \mathbf{PSPACE}$.[19]

Note that since every deterministic Turing machine is, by definition, a non-deterministic machine, we also clearly have $\mathbf{P} \subseteq \mathbf{NP}$ and $\mathbf{PSPACE} \subseteq \mathbf{NPSPACE}$. This leads to the following inclusions among complexity classes:

$$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXP} \subseteq \mathbf{NEXP} \subseteq \mathbf{EXPSPACE}$$

It thus follows that at least one of the first four displayed inclusions must be proper and also at least one of the third, fourth, or fifth. At present, however, this is *all* that is known – i.e. although various heuristic considerations can be cited in favor of the properness of each of the displayed inclusions, none of them are currently known to be true. Demonstrating these claims remains a major unfulfilled goal of complexity theory. For instance, the following is often described as the single most important open question in all of theoretical computer science:

T2: Is $\mathbf{P}$ properly contained in $\mathbf{NP}$?

T2 is an example of what is often referred to as a *separation question* involving complexity classes – i.e. a conjecture about whether complexity classes $\mathbf{C}_1$ and $\mathbf{C}_2$ with different definitions are in fact distinct. Such questions can be answered by demonstrating the existence of a language $X \in \mathbf{C}_2$ for which $X \notin \mathbf{C}_1$ (from which it typically follows that $\mathbf{C}_1 \subsetneq \mathbf{C}_2$ given the definitions of classes).

However T2 – which corresponds to what was referred to as *the* $\mathbf{P} \neq \mathbf{NP}$ *problem* above – is by no means the only separation question which is currently unresolved in complexity theory. For instance the following apparently weaker statement – which we saw in §2.2 to be related to the distinction between the first- and second machine classes – is also famously open:

T3: Is $\mathbf{P}$ properly contained in $\mathbf{PSPACE}$?

It is also possible to formulate a variety of other similar separation question classes intermediate between $\mathbf{P}$ and $\mathbf{PSPACE}$ (e.g. $\mathbf{P} \neq \mathbf{NP} \cap \mathbf{coNP}$, $\mathbf{NP} \neq \mathbf{coNP}$, $\Sigma_i^{\mathrm{P}} \subsetneq \Sigma_{i+1}^{\mathrm{P}}$,$\mathbf{PH} \neq \mathbf{PSPACE}$) as well as for classes involving models of parallel computation (e.g. $\mathbf{NC} \neq \mathbf{P}$), probabilistic computation (e.g. $\mathbf{P} = \mathbf{BPP}$), quantum computation (e.g. $\mathbf{NP} \subsetneq \mathbf{BPQ}$), and non-uniform computation (e.g. $\mathbf{NP} \subsetneq \mathbf{P/poly}$). Although not all of these classes can be defined here, each of the relations between classes just state is currently open (although believed to be true). Finding an unconditional proof of any of these statements would also currently constitute a result of the first magnitude in complexity theory.

[19] The time hierarchy theorem was originally presented in a paper of Hartmanis and Stearns (1965) entitled "On the computational complexity of algorithms" while the space hierarchy theorem is due to Cook (1973). In more precise form, the Time Hierarchy Theorem states that $\mathbf{TIME}(f(n)) \subsetneq \mathbf{TIME}(g(n))$ just in case the limit of the ratio of $f(n)\log(f(n))$ to $g(n)$ goes to 0 as n goes to infinity and that $f(n)$ and $g(n)$ are both *time constructible* – i.e. that there is a Turing machine which on input 1^n outputs $1^{f(n)}$ and $1^{g(n)}$.

2.5 Feasibility and P

In order to understand the practical significance of separation questions, it is useful to reflect further on the definition of the class **P** as well as the traditional argument that it characterizes the class of so-called *feasibly decidable* problems – i.e. those which can be solved "in practice" rather than only in the "in principle" sense of classical computability theory. Recall that under relevant historical conception, $f : \mathbb{N}^k \to \mathbb{N}$ is characterized as *effectively computable* just in case its values may be computed by an algorithm which is finitary in the sense that it halts after a finite number of steps each of which can be carried out mechanistically, but without concern for time or space resources. By analogy, $f : \mathbb{N}^k \to \mathbb{N}$ is characterized as *feasibly computable* just in case its values may be computed by an algorithm which is not only finitary but also such that it may be practically executed both with respect to the number of steps required to carry it out for a given input but also the resources required to perform each step individually.

Church's Thesis (CT) is traditionally taken to provide an adequate analysis of the former notion by equating effective computability with recursiveness. An analogous hypothesis for complexity theory was first proposed by Alan Cobham in a paper entitled "The intrinsic computational difficulty of problems" (1965) and is often referred as the *Cobham-Edmonds Thesis*:[20]

CET: A function $f(x)$ is feasibly computable if and only if $f(x)$ is computed by a Turing machine $T \in \mathfrak{T}$ such that $t_T(n) \in O(n^k)$ for some fixed k.

This proposal is extended to problems in the obvious way by asserting that X is feasibly decidable just in case its characteristic function is feasibly computable. CET thus has the effect of proposing that a necessary and sufficient condition for X to be feasible is that X is a member of **P**.

CT is traditionally thought to derive support from several different kinds of argument – e.g. quasi-inductive confirmation on the basis of a large class of cases, converging mathematical definitions, or perhaps even a successful conceptual analysis relative to a single model.[21] Although I will discuss the possibility of providing arguments of the second and third types for CET in §3, the consensus which has grown up in favor of this hypothesis is largely underpinned by considerations of the first sort. In particular, in almost all cases where we are able to uniformly solve a problem in practice, this is because we are in possession of a decision algorithm which can be implemented as a Turing machine with polynomial time complexity. And as I will discuss further in §2.6, in instances where we cannot solve in practice a problem which is decidable in principle, it can often be shown that the problem is *hard* for **NP** (or a more expansive class) and thus is unlikely to be in **P**. In this sense, CET is currently regarded as a useful guide to the large-scale contours of our computational practices.

Nonetheless, there are at least two sorts of considerations which complicate this picture. First, before we accept CET on inductive grounds, we should also confirm that problems which we find intractible in practice do not possess such algorithms. But although **NP**-hard problems appear to provide evidence for this, we cannot decide conclusively that these problems are not in **P** without resolving question T2 (i.e. $\mathbf{P} \neq \mathbf{NP}$). Second, although the concept of feasibility itself seeks to draw a distinction between "computability in principle" and "computability in practice", CET encompasses several idealizations which are undertaken in the standard development of complexity theory – e.g. in its use of order of growth analysis in the definition of **P**. And this raises questions such as the following:

[20] The contribution of Edmonds (1965a) will be discussed in §2.6.

[21] A classical source for such arguments is (Kleene 1952, §62) while a more modern one is (Sieg 2009, §3).

P6: To what sort of objects do we ascribe feasibility? Should feasibility be regarded as an intrinsic property of the items to which it is ascribed or is it relative to other parameters? How (if at all) is the notion of "computability in practice" related to various forms or grades of possibility (e.g. logical, mathematical, metaphysical, nominological, ...)?

As an initial answer to the first of these questions, note that we have already relied on intuitive judgements about feasibility several times – e.g. in observing that it is feasible to compute the sum of 25,211,713,952,371,115,192,327 and 75,557,863,725,914,323,419,121 via the carry addition algorithm, but infeasible to test them for primality via the trial division method. Such examples suggest that feasibility is a property which we attribute to what might be called a *task*, akin (e.g.) to everyday actions such as baking a cake, climbing a mountain, or walking from one location to another. For instance, it is in this sense that we say that the task of walking from Washington Square to Herald Square is feasible while that of walking from New York to Los Angeles is infeasible.

In assigning feasibility to problems (i.e. sets) or functions rather than tasks, complexity theory can be understood as abstracting from such everyday examples along two dimensions. First, it identifies parameters of a given task which can be understood as inputs to a general method for carrying it out – e.g. numbers to be checked for primality or geographic coordinates to be traveled between. And second, it quantifies over different methods by which the the task can be carried out – e.g. trial division vs the Sieve of Eratosthenes vs ... or walking vs driving vs ....

On the resulting analysis, statements of the form "problem X is feasible" are thus assigned the following interpretation: *there exists a decision algorithm A for X which is such that for all inputs x, if it is possible to construct an inscription of x in practice, then it is possible in practice to execute A on x to yield an output.*[22]

On this reconstruction, the pretheoretical notion of feasibility is clearly related to that of problem difficulty discussed in §2.1. But according to CET, it follows that the relevant pretheoretical notion of feasibility is unlike that of difficulty in that it is regarded as *bivalent* rather than graded – i.e. a problem is taken to be feasible if it is in $\mathbf{P}$ and infeasible otherwise. In particular, CET can be seen as drawing a line in the middle of the sequence

$$O(1), O(\log_2(n)), O(\sqrt{n}), O(n), O(n^2), O(n^3), \ldots, O(2^{\sqrt[3]{n}}), O(2^{\sqrt{n}}), O(2^n), O(n!), O(2^{2^n}), \ldots$$

which separates *polynomial* orders of growth (i.e. those approximated from below by $O(n), O(n^2), O(n^3), \ldots$) and so-called *super-polynomial* ones (i.e. those approximated from above by $O(2^n), O(2^{\sqrt{n}}), O(2^{\sqrt[3]{n}}), \ldots$). According to CET, the former are asserted to measure the running time of algorithms by which feasible problems can be solved while the latter are asserted to measure the best possible running times by which infeasible problems can be solved.

[22] Such an analysis suggests that there is a *counterfactual* component in the notion of feasibility which complexity theory seeks to analyze. For note that such an account suggests that it is only relative to the *assumption* that we can construct an input x of a given size $|x|$ – say by inscribing it on a blackboard or storing it in a computer's memory – that it becomes of practical interest whether we can apply a given algorithm A to x to yield an output. But if this is correct, then it appears that complexity theorists are under no obligation to provide an independent account of what it means for it to be "possible in practice" to perform a given task (thus also freeing them of the obligation of contrasting this notion with various other grades of possibility which enter into traditional philosophical discussions). Rather they may accept this notion as an unanalyzed primitive and then seek to provide an account of "relative possibility in practice" for parameterized tasks of the form "*if* we can perform a task requiring n steps in practice, *then* we can perform a task requiring $O(f(n))$ steps in practice". But note that it is exactly this sort of criterion the Cobham-Edmonds Thesis suggests.

Like many exercises in drawing boundaries, it might initially seem that this is an arbitrary distinction which need be not underwritten by a either a similarly bivalent difference in our practices or an antecedently recognized concept of "computability in practice". It is, for instance, a consequence of CET that a problem whose most efficient decision algorithm has running time $t_1(n) = \lceil 2^{10^{-12} \cdot n} \rceil \notin O(n^k)$ (for any k) is categorized as infeasible, whereas a problem whose most efficient algorithm has running time $t_2(n) = 10^{12} \cdot n^{10^{12}} + 10^{12}$ is categorized as feasible. This is so despite the fact that we could apply the former algorithm in practice to inputs of size 10^{12} (in fact trivially so), whereas we would have no hope of applying the latter in practice even to inputs of size 10.

Such examples draw attention to the role of *constant factors* – i.e. the degrees of polynomials (k) as well as additive and scalar factors (c) – and the use of order of growth notation in the definition of the class **P**. According to CET these factors are precluded from accounting for the basis of judgements about feasibility and infeasibility. A commonly cited response to this objection is that the sorts of algorithms which we discover and apply in practice typically have small constant factors (say $k \leq 6$ and $c < 1000$). A related phenomenon is that instances where the membership of a problem X in **P** has been shown have previously been open or that X is known to be in **P** only in virtue of a polynomial time algorithm with "large" constant factors, additional work has often yielded more efficient algorithms with smaller constants.[23]

On the other hand, for many problems we currently believe to be infeasible it is possible to provide a heuristic argument that their most efficient algorithms have super-polynomial time complexity - e.g. $O(2^n)$ or even $O(n!)$. As I will discuss further in §2.6, this suggests that CET may do a better job as a *negative* criterion for feasibility than a positive one. These considerations notwithstanding, further reflection on the thesis suggests several conceptual questions which refine those posed by P6:

P7: Should an adequate analysis of feasibility take account of the role of constant factors in computational difficulty? Should such an analysis take account of the computational costs of executing procedures for *all* inputs in their domains, or only those which we can construct in practice? Are other notions of a *solution* to a problem (e.g. approximative or probabilistic ones) relevant to our background conception of feasible computability or decidability?

I will say a bit more about the first and second of these questions in §3 and the third in §6.

2.6 Intractability and NP-completeness

As with the case for equating feasibility with polynomial time decidability, a complementary set of considerations can be adduced for identifying the notion of a *computationally intractable problem* with one which is *hard* for the class **NP**. A traditional example is provided by the well-known *Traveling Salesman Problem*:

TSP Given a list of cities and distances between them represented as a weighted graph $G = \langle V, E, d \rangle$ (where $d(u, v)$ is the distance between u and v), and a budget $b \in \mathbb{N}$, is there a tour visiting each city exactly once and returning to the starting city of total distance $\leq b$?

[23] For instance, although the problem PRIMES was shown by Agrawal et al. (2004) to be in **P** relative to an $O(n^{12})$ algorithm, this has subsequently been reduced to $O(n^6)$ by Lenstra et al. (2019). Such examples thus suggest that it is the distinction between polynomial and exponential time which provides the most robust distinction between feasibility and infeasibility rather than, say, between time $\mathbf{TIME}(n^2)$ and $\mathbf{TIME}(n^{17})$.

TSP is also evidently decidable by the following brute force algorithm: enumerate all possible tours $T_1, \ldots, T_k$ in G (where $|V| = n$, $T_i = \langle u_1, \ldots, u_n \rangle$, and $u_1 = u_n$ is a tour), checking in each case if $\Sigma_{i=1}^{n-1} d(u_i, u_{i+1}) \leq b$. But since there are $O(n!)$ possible tours it is evidently not possible to carry out this procedure in practice, even for graphs which may otherwise arise in concrete cases – e.g. with $n = 50$.

TSP has many practical applications – e.g. in planning, logistics, and manufacturing. As such, a significant amount of effort has been invested in finding an efficient solution since the problem was formulated in the 19th century.[24] A promising development in this regard was Bellman's (1962) discovery that the naive $O(n!)$ algorithm for TSP can be improved to $O(2^n)$ via a technique known as *dynamic programming*. But both TSP and a number of other similar problems which were being investigated at this time – e.g. determining whether a graph has a Hamiltonian path (i.e. a tour which visits every vertex exactly once), or whether it is has a clique, independent set, or vertex cover of size greater than k – resisted attempts to find algorithms which were asymptotically more efficient than brute force search. Starting in the 1950s, such problems started to be classified together as *intractable*.

Many of the problems in the relevant class are decision variants of search problems whose witnesses provide certificates for membership and are thus easily seen to be in **NP**. Edmonds (1965b) (in effect) proposed that the characterizability of a problem in this manner together with the lack of an efficient decision algorithm provides a minimal criterion of intractability. Such a proposal not only suggests that intractability enjoyed a pretheoretical status similar to that of feasibility, but also that the two concepts are in fact dual to one another – i.e. *a problem X is intractable just in case it is not feasible*. This in turn implies that CET should be understood as providing not only a necessary criterion of feasibility but a sufficient one as well – i.e. that the *failure* of a problem to possess a polynomial time algorithm should be understood as entailing its infeasibility or intractibility. But this also raises the question of when we are justified in ascribing intractability to a given problem – i.e. when we say that X is intractable do we mean merely that we do not know of an efficient decision algorithm for deciding X (despite our best efforts to find one) or that no such algorithm can exist in principle? For note that *proving* that there exists a problem $X \in \mathbf{NP}$ which does not have a polynomial time algorithm is sufficient to demonstrate $\mathbf{P} \neq \mathbf{NP}$. And thus if we adopt Edmonds' understanding of intractability, we would presently have to look beyond the sorts of problems with which he was originally concerned to find demonstrable examples of intractability.

Two major advances in our understanding of intractability came in quick succession with the papers "The complexity of theorem-proving procedures" by Stephen Cook (1971) and "Reducibility among combinatorial problems" by Richard Karp (1972). In addition to formally introducing the class **NP**, Cook defined the notions of polynomial time reducibility, **NP**-hardness, and **NP**-completeness. A problem X is said to be *polynomial time many-one reducible* to Y just in case there exists a polynomial time computable function $f(x)$ such that

$$\text{for all } x \in \{0,1\}^*,\ x \in X \text{ if and only if } f(x) \in Y$$

In this case we write $X \leq^P Y$ and say that $f(x)$ is a *polynomial time reduction* of X to Y. Y is additionally said to be **NP**-*hard* if $X \leq^P Y$ for all $X \in \mathbf{NP}$. Finally, Y is said to be **NP**-*complete*

[24] In fact the original formulation of TSP (by W.R. Hamilton and others) provided part of the motivation for the subject now known as *combinatorial optimization* – see (Schrijver 2005). See (Gutin and Punnen 2002) for further discussion of contemporary applications.

if it is both **NP**-hard and also a member of **NP** itself.[25]

It is easy to see that a bounded version of the classical Halting Problem for non-deterministic Turing machines – i.e. does machine N halt on input x in t steps (where t is represented as a unary numeral) – is **NP**-complete. But Cook (and independently Levin 1973) showed that not only is SAT **NP**-complete but so is the special case known as 3-SAT consisting of the class of satisfiable formulas in conjunctive normal form where each conjunct is the disjunction of exactly three negated or unnegated propositional variables. This result is already surprising in the sense that 3-SAT is a logical problem apparently unrelated to the operation of Turing machines. What is more surprising, however, is that Karp then showed that TSP, HAMILTONIAN PATH, CLIQUE, INDEPENDENT SET, and VERTEX COVER as well as a number of other problems which had previously been studied in graph theory and combinatorics are also **NP**-complete (21 problems in total). Subsequently many other problems from a diverse range of subjects such as algebra, number theory, formal language theory, and game theory have also been shown to be **NP**-complete.[26]

Note that if X is polynomial time reducible to Y via $f(x)$, then an efficient algorithm A for deciding Y would yield an efficient algorithm for deciding X as follows: i) on input x, compute $f(x)$; ii) use A to decide if $f(x) \in Y$, accepting if so, and rejecting if not. Part of the significance of the existence of **NP**-complete problems is thus that the discovery of a polynomial time algorithm for *any one* of them would yield polynomial time algorithms for *all* problems in **NP**. This in turn reflects the consensus that we are justified in attributing intractability to specific **NP**-complete problems like TSP. For although we cannot currently *prove* that this problem not does have a polynomial time algorithm, the fact that the existence of such an algorithm would also yield efficient decision algorithms for a large array of other problems which have been intensively studied (sometimes for more than 50 years) lends credence to the conjecture that no such algorithm can exist.

Hardness and completeness for other classes extending **P** – such as **coNP** (the class containing the problems which are the complements of those in **NP**), **PSPACE**, and **EXP** – are defined similarly. These too have natural complete problems – e.g. deciding whether a propositional formula is a tautology is **coNP**-complete, while many problems like CHESS about games in which the players make moves in alternation are complete for either **PSPACE** or **EXP**.[27] The latter class is known

[25]The foregoing definitions of reduction, hardness, and completeness can be understood as feasible variants of the traditional computability theoretic concepts of *many-one reducibility* and *many-one hardness* and *completeness* (see, e.g. Rogers 1987, §7). These definitions were originally introduced by Karp (1972). On the other hand, Cook (1971) originally defined **NP**-completeness in terms of the polynomial time analogue of *Turing reducibility* (Rogers 1987, §9) – i.e. $X \leq_T^P Y$ just in case membership in X can be decided in polynomial time by a Turing machine which has access to Y as an oracle. It is possible to construct examples showing the distinctness of the degree structures induced by $X \leq^P Y$ and $X \leq_T^P Y$. But as the two definitions typically coincide in practice, Karp's definitions are now more widely employed.

[26]The classical reference for **NP**-complete problems is (Garey and Johnson 1979) which already contains over 300 examples. Current online compilations often contain more than 1000.

[27]For instance, in order to see that the problem TAUT consisting of all propositional tautologies is complete for **coNP** observe that $\varphi \in$ TAUT if and only if $\neg\varphi \notin$ SAT – i.e. prefixing a negation to a formula provides a reduction of TAUT to the complement of the known **NP**-complete problem SAT. But although it intuitively seems harder to check whether a formula is a tautology (i.e. true in *all* rows of its truth table) than whether it is satisfiable (i.e. true in *at least one* row of its truth table), the statement **NP** $\neq$ **coNP** is also famously open. In fact **NP** and **coNP** form the first level of what is called the *polynomial hierarchy* **PH** which is believed to properly stratify problems between **P** and **PSPACE**. **PH** is defined in a manner which is structurally similar to the arithmetical hierarchy **AH** in computability theory (Rogers 1987, §14), wherein **P**, **NP** and **coNP** play roles which are respectively analogous to recursive (Δ_1^0), recursively enumerable (Σ_1^0), and co-recursively enumerable (Π_1^0) sets of natural numbers. But although a straightforward diagonal argument suffices to shows that **AH** is a proper hierarchy, it is yet again open whether **PH** collapses to a fixed level.

to properly extend **P** as a consequence of the Time Hierarchy Theorem. Thus **EXP**-complete problems can currently be regarded as *demonstrably* intractable relative to CET.

The ubiquity of complete problems in complexity theory also raises a number of questions about mathematical difficulty which refine those posed in §2.1 and §2.4, both technically and conceptually. Some initial examples are as follows:

T4: Presuming that $\mathbf{P} \neq \mathbf{NP}$, are there problems $X \in (\mathbf{NP} - \mathbf{P})$ which are *not* **NP**-complete?[28] If so, are particular problems which are known to be in **NP** but currently believed not to be in **P** – e.g. Factors or Graph Isomorphism – **NP**-complete? More generally, what does the internal reducibility structure of **NP** and other classes tell us about absolute and relative problem difficulty?

It is also notable that **NP**-complete problems can appear to be about very different topics – e.g. Sat concerns propositional formulas, Tsp concerns weighted graphs, etc. On the other hand, we have already observed that any pair X, Y of such problems is such that an algorithm solution to X can be efficiently transformed into one for Y and conversely. This in turn provides a natural context in which to pose questions about problem identity such as the following:

P8: How finely are mathematical problems individuated in practice? For instance, if we can prove that $X \leq^P Y$ and $Y \leq^P X$ (or similarly with respect to other reducibility notions) is there a salient pretheoretical sense in which we have discovered that X and Y are *same* problem?

3 Feasibility and the foundations of mathematics

One reason why philosophers appear to have not engaged more actively with complexity theory to date is that many contemporary sources present the subject in a manner which makes it appear disconnected from the development of mathematical logic and the attendant foundational debates of the late nineteenth and early twentieth centuries. The goal of this section is to illustrate why such a characterization is misleading by illustrating how complexity theory grew out of developments within the traditional Hilbert program alongside proof theory and computability theory in the 1930s-1950s. In fact, this stream of events can be understood as eventuating in Cobham's (1965) original definition of polynomial time and the proposal that it aligns with feasible computability.

A reasonable place to begin is with the following passage from Paul Bernays's (1935) well-known paper "On platonism in mathematics":

> Intuitionism makes no allowance for the possibility that, for very large numbers, the operations required by the recursive method of constructing numbers can cease to have a concrete meaning. From two integers k, l one passes immediately to k^l; this process leads in a few steps to numbers which are far larger than any occurring in experience, e.g. $67^{(257^{729})}$.
>
> Intuitionism, like ordinary mathematics, claims that this number can be represented by an Arabic numeral. Could not one press further the criticism which intuitionism makes of existential assertions and raise the question: What does it mean to claim the existence of an Arabic numeral for the foregoing number, since in practice we are not in a position to obtain it? ...¶...

[28] This question was partially answered by Ladner (1975) who demonstrated that if $\mathbf{P} \neq \mathbf{NP}$, then then **NP** has a rich degree structure similar to that of the recursively enumerable many-one degrees in computability theory. However since the methods involved in this proof are diagonal constructions, this result on its own is not usually taken to demonstrate that there are "natural" non-complete problems in $\mathbf{NP} - \mathbf{P}$ (even if this class turns out to be non-empty).

> [T]here is no precise boundary between the numbers which are accessible and those which are not. One could introduce the notion of a "practicable" procedure and implicitly restrict the import of recursive definitions to practicable operations. To avoid contradictions, it would suffice to abstain from applying the principle of the excluded middle to the notion of practicability. (Bernays 1935, pp. 61-62)

By the mid-1930s much of the technical and philosophical work which we now think of as comprising the Hilbert program and the attendant foundational characterization of what Hilbert and Bernays referred as the *finitary standpoint* – i.e. the view which has come to called *finitism* in subsequent sources – had already been completed. As most readers will be aware, a central aspect of the program was the use of recursive definitions to provide an account of how certain modes of inference (such as quantifier-free mathematical induction) could be reduced to numerical computation. Central amongst the schemes which were considered for this purpose what is now called *primitive recursion* which, in the case of binary functions, takes the familiar form

$$(1) \quad \begin{aligned} f(x,0) &= g(x) \\ f(x',y) &= h(x,f(x,y)) \end{aligned}$$

Here the auxiliary functions $g(x)$ and $h(x,y)$ are assumed to be defined from the basis functions (i.e. the successor function $'$, 0, and projections) via this scheme and composition.[29]

The primitive recursion scheme allows for the definitions of addition, multiplication, exponentiation, ... to be built up in the familiar manner – e.g. as $\mathrm{add}(x,0) = x, \mathrm{add}(x',0) = \mathrm{add}(x,y)'$. Such definitions provide examples of how primitive recursion is conventionally understood to be a quasi-concrete mode of computation on numbers represented in unary notation as $0, 0', 0'', \ldots$ For instance, computing the sum $2+3$ is reduced to the sort of computation

$$\mathrm{add}(0'',0''') = \mathrm{add}(0'',0'')' = \mathrm{add}(0'',0')'' = \mathrm{add}(0'',0)''' = 0''''$$

which Hilbert (1922, p. 1123) famously suggested should be understood in terms of the "construction and deconstruction of number signs". This is illustrative of the way in which finitists have subsequently sought to reduce the verification of statements of number theory to "intuitively evident" calculations.

On the other hand, it is also possible to regard the primitive recursive functions as a model of computation $\mathfrak{PR}$ in their own right – i.e. by taking "machines" to be function definitions which are "executed" in the manner just illustrated. The precise definition of this model by Gödel (1931) historically preceded the Turing machine model $\mathfrak{T}$ while also serving as the basis for the subsequent definition of the general recursive functions by Gödel (1934) and the μ-recursive functions by Kleene (1936). These models determine the same class of function of as $\mathfrak{T}$ as "computable". It is of some note, however, the original textbooks in computability theory (e.g. Kleene 1952; Rogers 1987) take not $\mathfrak{T}$ but rather the μ-recursive functions – together with corresponding conception of operating on *unary* numerals – as a basic model of computation .

In light of the equivalence results between models which underpin the conventional argument for Church's Thesis, the choice between different Turing-complete models may appear like an arbitrary decision in regard to developing a general theory of computability and non-computability. But we have also seen that it is the ability of "reasonable" models like $\mathfrak{T}$ to operate directly on *binary* strings

[29] Although Hilbert first considered recursive definitions in (Hilbert 1922), the finitary basis of primitive recursion in particular is presented in greatest detail in (Hilbert and Bernays 1934, §1, §2, §7).

which allows them to directly implement algorithms like carry addition or multiplication. These algorithms operate efficiently on positional notations – i.e. respectively in time $O(n)$ and $O(n^2)$ where n is the maximum *length* of the binary representations of the inputs. On the other hand, primitive recursive procedures of the sort just described operate on unary notations in a manner which typically requires that the number of steps in their executions is proportional to the *value* of the function which is being computed – e.g. it takes $O(x)$ steps to compute the sum $x+y$ via the prior definition of $\text{add}(x, y)$, $O(x \times y)$ steps to compute $O(x \times y)$ via $\text{mult}(x, y)$, etc. These procedures thus run in time which at best *exponentially* proportional to the size of the positional numerals by which we would most naturally denote their inputs and outputs, precisely because they operate in a manner which requires that the corresponding unary representations are "deconstructed" completely. This in turn suggests that they are often infeasible to carry out by the standards discussed in §2.[30]

Bernays signaled his awareness of this issue by drawing attention to both the ability of short Arabic (i.e. decimal) notation to represent large numbers and also how such notations together with exponentiation could be used to construct succinct expressions denoting far larger numbers still. For instance we have already observed that we are not in practice able to obtain a unary numeral denoting the value of (say) a 50 digit binary numeral. But since $\log_{10}(n^m) = m \log_{10}(n)$, the length of the base 10 numeral corresponding to $67^{(257^{729})}$ is on the order of 1.27×10^{1757} digits. And thus it would also be an infeasible task to concretely inscribe even the decimal representation of this number.

In light of such observations, the question arises as to how the 'intuitability" of a number should be understood, not only in regard to the characterization of finitism but also, as Bernays notes, for intuitionism as well.[31] But what matters most in the present context are three other observations which appear implicit in Bernays's remarks: i) even when attention is limited to the domain of quasi-concrete objects like numerals, there is a salient sense in which the boundary between the "intuitable" and the "unintuitiable" may be vague or otherwise underdetermined; ii) *exponentiation* (as applied to unary numerals) or *iterated exponentiation* (as applied to positional numerals) provides an apparent means of stepping over this boundary; and iii) there is at least the possibility for developing a foundational standpoint distinct from finitism which takes i) and ii) into account by making the notion of *feasible procedure* itself an object of study.[32]

Although Bernays demurred from developing the third possibility further himself, this task was taken on by Hao Wang (1958) and more systematically by Alexander Yessenin-Volpin (1961; 1970; 1981). Wang described in more detail a restriction of finitism which he labels "anthropologism" and locates it at the bottom of a hierarchy of increasingly permissive foundational standpoints. Taking Bernays's paper as his point of departure, he proposes making operations on notation an explicit object of study alongside that of feasibility. Although Wang also did not propose a formalism

[30] Although the examples just cited pertain to the running time of specific primitive recursive algorithms, it is also possible to obtain lower bound result for $\mathfrak{PR}$ once certain assumptions are made about how primitive recursion is implemented computationally. For instance, Colson (1991) showed that there is no primitive recursive call-by-name procedure for computing $\min(x, y)$ in fewer than $O(\min(x, y))$ steps. This illustrates a way in which models like $\mathfrak{PR}$ appear "unreasonable" not in the sense that they allow *too much* computation to be performed in a single step (e.g. like $\mathfrak{P}$) but because they allow *too little*.

[31] Such questions have, of course, been the subject of substantial debate (e.g. Parsons 1979; Tait 1981; Dummett 2000). But the scholarly details are only of concern here insofar as Bernays's argument makes it incumbent on proponents of finitism or intuitionism to distinguish their views from those of the strict finitists as described below.

[32] In citing the passage reproduced above, Wang (1958, p.473) translates Bernays's original phrase "processus «effectuable»" as "feasible process". This appears to be the point at which the term *feasible* entered the literature out of which complexity theory would eventually developed.

for carrying this out, further reflection on anthropologism would later play an explicit role in the analysis of feasbility in the 1960s and 1970s.[33]

It was also during this period when hierarchies contained within the primitive recursive functions were first being explored as potential measures of computational difficulty. This most notably included the so-called *Grzegorczyk hierarchy* (1953) which properly stratifies the class of primitive recursive functions into levels $\mathcal{E}^0 \subseteq \mathcal{E}^1 \subseteq \mathcal{E}^2 \subseteq \ldots$ which can be (roughly) described as classifying primitive recursive functions according to the number of applications of the recursion scheme (1) are needed in their definitions – e.g. $x + k \in \mathcal{E}^0, x + y \in \mathcal{E}^1, x \cdot y \in \mathcal{E}^2, \ldots$ The fourth class in this hierarchy $\mathcal{E}^3$ corresponds to the so-called *elementary functions* which include those whose order of growth is proportional to the finitely iterated exponential function $2^{2^{\cdot^{\cdot^{2}}}}$. It is now known that $\mathcal{E}^3$ is a functional variant of a complexity class $\mathbf{ELEM} = \mathbf{TIME}(2^n) \cup \mathbf{TIME}(2^{2^n}) \cup \mathbf{TIME}(2^{2^{2^n}}) \cup \ldots$ As **EXP** forms the first level in the definition of this class, **ELEM** far exceeds **P** (or even **PSPACE**). But as was originally observed by Ritchie (1963), the region between $\mathcal{E}^2$ and $\mathcal{E}^3$ is still a natural territory to explore in order to account for the complexity of algorithms which are useful in practice.

These developments served as the immediate context of Cobham's (1965) influential paper "The intrinsic difficulty of functions". Cobham's goal was to provide an account of computational difficulty which distinguishes the complexity of functions within $\mathcal{E}^3$ – e.g. addition versus multiplication – so as to obtain an analysis "intrinsic to the functions themselves and not with properties of particular related algorithms" (p. 24) or the "type of machine used in the computation" (p. 27). To this end, he proposed modifying the primitive recursion scheme so that a single application reduces the length of a base b *positional* representation of the input x by one (and thus its value by $\lfloor x/b \rfloor$). This leads to the scheme of so-called *limited recursion on notation* which in the case for $b = 2$ is exemplified by the following schema for binary functions:

$$
\begin{aligned}
(2) \quad & f(x, 0) = g(x) \\
& f(2 \cdot x, y) = h_0(x, f(x, y)) \\
& f(2 \cdot x + 1, y) = h_1(x, f(x, y))
\end{aligned}
$$

The second clause in this definition handles the case for even values in the series defined by $x_0 = x$ and $x_{i+1} = \lfloor x_i/2 \rfloor$ while the third clause handles the case for odd values. It thus follows that a recursive computation defined in this manner will have length proportional to $\log_2(x)$ – i.e. the length of x's binary representation – rather than x itself (as in the case of traditional primitive recursion).

It can be shown that the class **C** of functions obtained as the minimal closure of a small set of initial functions under composition and a generalized form of (2) corresponds to the class **FP** of functions computable in polynomial time by a Turing machine. This equivalence is of note in part because it provides a characterization of **P** – i.e. as the class of languages whose characteristic functions are definable by limited recursion on notation – which is independent of any "machine-like"

[33] It should be noted more generally that Wang played an important role in cross-fertilizing ideas originating in philosophy of mathematics and mathematical logic within the early development of computer science in a manner which facilitated the development of complexity theory. A student of Quine at Harvard and protégé of Bernays in Zurich, he would later become a colleague of Dummett at Oxford (to whom he would relate "Wang's paradox"). After returning to Harvard, he did early work on automatic theorem proving and combinatorial decision problems (e.g. "Wang tiles"), as well as supervising Stephen Cook's dissertation on the complexity of arithmetical functions. See (Wang 1990) for Wang's own account of these developments.

model of computation. This further testifies to the mathematical robustness of polynomial time, thereby lending credence to CET.

But of equal importance is that **C** is obtained by closing a class of "obviously feasible" functions under operations which can be argued to preserve feasibility. In fact, Cobham's motivates limited recursion on notation in a manner which makes explicit reference to the digit-by-digit operation of efficient algorithms like carry addition and multiplication for operating on positional notations similar to those described by Wang (1958).[34] As anticipated by Bernays and Wang, Cobham's analysis of feasible computability can thus be understood as proceeding in the same foundational spirit as Turing's (1936) and Church's (1936) analyses of effective computability. Although this progression has been largely overlooked by philosophers, it is also related to concerns which inspired the better known *strict finitist* program of Yessenin-Volpin.

Yessenin-Volpin is best known for his (1961) critique of "traditional mathematics" as well as his (1970) proposal to introduce the notion of feasibility into the language of mathematics in the service of a consistency proof for Zermelo-Fraenkel set theory.[35] In particular, Yessenin-Volpin went further than Wang or Bernays, in introducing an object language predicate $F(x)$ such that $F(n)$ is intended to express that n is a "feasible number" which he further glosses as one "up to which it is possible to count". In order to formalize reasoning about this notion, he also proposed axiomatic principles such as

F_0: $F(0)$

F_s: $\forall x(F(x) \to F(x+1))$

which respectively express that it is possible to count up to 0 and if we can count up to x, then we can up to $x+1$ by adjoining an $'$ to its unary representation. Citing authors such as van Dantzig, Borel, and Mannoury as precedent, Yessenin-Volpin also asserted that 10^{12} is evidently an infeasible number.[36] This leads to the additional principle

F_i: $\neg F(10^{12})$

But of course the three preceding statements are mutually inconsistent. For F_0 and F_s allow us to derive $F_i(n)$ for any $n \in \mathbb{N}$, either quickly by first using an instance of the first-order induction scheme for $F(x)$ with F_0 and F_s as premises to derive $\forall x F(x)$ followed by $F(10^{12})$ or by an apparently more arduous derivation consisting of repeated application of *modus ponens* to the sequence of statements beginning with $F(0), F(0) \to F(1), F(1) \to F(2), \ldots$ derived from F_s via universal instantiation.

[34] In addition to the traditional primitive recursive basis functions, the relevant initial functions are the operations $2 \cdot x$ and $2 \cdot x + 1$, the length function $|x| = \log_2(x)$, and the so-called *smash function* $x\#y = 2^{|x| \cdot |y|}$. (See Rose (1984) for a precise definition and proof that **C** = **FP**.) The first two of these are included to formulate the scheme (2) in the context of which they can be understood as taking the value of a binary numeral $d_k d_{k-1} \ldots d_1 d_0$ on the left and "deconstructing" it into $d_k d_{k-1} \ldots d_1$ on the right. The smash function is included so that it is possible to define terms which grow at the rate $O(2^{|p(x)|})$ for any polynomial $p(x)$ so as to formulate a side condition bounding the functions $h_0(x, y)$ and $h_1(x, y)$ by one which is known to be in the class. This formalizes the constraint that the auxiliary computations performed by a feasible procedure at each stage during the sort of length-bounded recursion described above are themselves feasibly computable. Similar characterizations of **FP** which avoid explicit mention of the smash function are provided by Bellantoni and Cook (1992) and Leivant (1994).

[35] Although the details of the latter remain murky, see (Gandy 1982) for a partial reconstruction.

[36] "Let us consider the series F of feasible numbers, i.e. of those up to which it is possible to count. The number 0 is feasible and if n is feasible then then $n < 10^{12}$ and so n' also is feasible. And each feasible number can be obtained from 0 by adding $'$; so F forms a natural number series. But 10^{12} does not belong to F." (Yessenin-Volpin 1970, p. 5)

The *locus classicus* for this complaint is Michael Dummett's well-known (1975) paper "Wang's Paradox". Therein Dummett observes that the derivations just described match the format of two forms of the well-known *sorites paradox* (as more traditionally formulated for natural language predicates like *bald* or *heap*). He thus concludes that Yessenin-Volpin's notion of feasibility is vague and thus potentially paradoxical.[37] This concern stands behind the apparent consensus among philosophers that even if there is no intrinsic incoherence in the concept of feasibility itself, such a concept is at least vague in a manner which places it outside the purview of mathematics as traditionally conceived.[38]

But although this elucidates to some extent why philosophers have been reluctant to engage with the notion of feasibility (and hence perhaps with complexity theory more generally), it is also evident that Yessenin-Volpin's treatment of this notion departs in important ways from how it is now understood. A central discrepancy pertains to the objects to which he takes the predicate "feasible" to apply. For as we have just seen, Yessenin-Volpin takes feasibility to a property of *individual natural numbers* which holds he takes to hold of 0, be preserved under successor, but which fails to hold (e.g.) of 10^{12}. Extending his analysis one step, such attributions can be understood to also apply to the *tasks* of counting (in unary) up to the numbers in question. But as we saw in §2.3, complexity theorists understand feasibility not as a notion which applies to individual tasks, but rather to problems (or functions). Moreover, what determines whether a problem X is feasible in the contemporary sense is not the behavior of a particular algorithm on a particular input, but rather the order of growth of the running time complexity of the most efficient algorithm for deciding X.

On the other hand, we have seen that the Cobham-Edmonds Thesis does indeed make a distinction among orders of growth of the sort which might at first seem similar to that between Yessenin-Voplin's distinction between feasible and infeasible numbers. Some consequences of this analysis are as follows:

O_0: $O(1)$ (i.e. constant time) is a feasible rate of growth.

O_s: If $O(n^k)$ is feasible, then so is $O(n^{k+1})$.[39]

O_i: Super-polynomial orders of growth such as $O(2^n)$ are not feasible.

On this basis one might fear that the intuitions about feasible computability codified by the Cobham-Edmonds Thesis exhibit the same kind of instability as Dummett suggests applies to Yessenin-Volpin's account of feasible numbers.

To see that this is not the case, it suffices to observe that the ordering $\prec$ conventionally used to compare the orders of growth of $f, g : \mathbb{N} \to \mathbb{N}$ is given by that of *eventual domination*:

$$O(f(n)) \prec O(g(n)) \text{ if and only if there exists } n_0, c \in \mathbb{N} \text{ such that for all } n > n_0, f(n) < c \cdot g(n)$$

It is a consequence of this definition that $O(1) \prec O(n) \prec O(n^2) \prec \ldots$ – i.e. the polynomial orders of growth form an ω-sequence with respect to $\prec$. However, it also follows that $O(n^k) \prec O(2^n)$ for

[37] It is also evident that Bernays (1935) had the same worry about the consistency of a formalized theory for reasoning about feasibility much earlier. In particular, it is presumably for this reason that he suggested that the law of the excluded middle not be applied to the notion of feasibility since, for instance, the potential existence of "borderline cases" of feasible numbers make us reluctant to assert $\forall x(F(x) \vee \neg F(x))$.

[38] See, e.g. (Kreisel 1967b, p. 10), (Kreisel 1970, pp. 507-508), (Lavine 1994, p. 248), (Troelstra 2011, p. 153), (Feferman et al. 2000, p. 410), (Gaifman 2004, p. 16). See also (Dean 2018) for more on the historical and technical connections between attempts to analyze feasibility, the sorites paradox, and natural language vagueness.

[39] This follows since CET assert that all polynomial orders of growth are feasible.

all $k \in \mathbb{N}$ – i.e. $O(2^n)$ (as well as other super-polynomial orders of growth) are "points at infinity" sitting above each polynomial order of growth with respect to this ordering. From this it follows that such orders cannot be reached from below by a sorties-like sequence of polynomial orders of growth.

This suggests that by equating feasibility with polynomial-time decidability we are not embracing a conception of feasibility which is vague in a manner which renders it potentially paradoxical. As we have seen, such a conception does idealize away from the role of constant factors. But by refusing to provide an account of such factors, we can now also see how complexity theory avoids becoming embroiled in the debates which historically inhibited the development of foundational standpoints such as those described by Bernays, Wang, and Yessenin-Volpin. Nonetheless, a number of questions about their proposals remain:

P9: Is it possible to motivate strict finitism on the basis of concerns which refine the traditional characterization of finitism – e.g. in terms of novel antinomies, the justification of mathematical induction, or the role of notation systems in mathematical practice? What epistemic gains might accompany a strict finitist reconstruction of a given portion of mathematics – e.g. in terms of the avoidance of infinitary assumptions, or the size, comprehensibility, or other computational aspects of proofs? Does strict finitism admit to an axiomatic presentation? How might such a formal theory relate to **P**, other complexity classes, or open separation problems?

It should finally be noted that reflection on the issues summarized in this section has also inspired a considerable amount of technical work on theories collectively known as *bounded* (or *feasible*) *arithmetics*. Such systems are exemplified by Buss's (1986) S^i_2 and T^i_2 which enrich the familiar language of first-order arithmetic with operations which facilitate interpreting natural numbers as (the denotations of) binary strings while also restricting the scope of the first-order induction scheme. A hallmark of such theories is that their provably total functions can be used to characterize **P** and other complexity classes in a manner which allows for logical reformulations of separation questions in terms of metalogical properties such as finite axiomatizability. But although the introduction of such theories reforged a link between complexity theory and mathematical logic in the 1980s, work in this tradition has also gone largely unnoticed by philosophers.[40]

4 On P versus NP

As we have seen in §3, the development of complexity theory through the mid-1960s can be understood as cashing out observations about the exigencies of numerical computation many of which could have been framed in the 1930s. But it was not until the discovery of the phenomenon of **NP**-completeness in the early 1970s that complexity theory became a subject in its own right. At the forefront of these developments was the framing of the $\mathbf{P} \neq \mathbf{NP}$ problem itself, followed by the gradual accrual of evidence which suggest that it is itself a *hard* problem in the sense of propositional difficulty sense discussed in §2.1.

In light of its notoriety, $\mathbf{P} \neq \mathbf{NP}$ has also frequently been the subject of survey in its own right – e.g. (Cook 2006), (Fortnow 2009), (Aaronson 2016). Due to both the scrutiny it has received and

[40] This is so despite the fact that two of the earliest forms of bounded arithmetic – i.e. the theories PB of Parikh (1971) (which is now known as $\mathrm{I}\Delta_0$) and PV of Cook (1975) – were explicitly introduced as attempts to formalize Wang's anthropological standpoint. Some other examples of technical work in this spirit with similar foundational goals includes (Vopěnka 1979), (Nelson 1986), (Ferreira 1994), and (Pudlak 1996).

its unsettled status, the following remarks can do little more than underscore the relation of the problem to some standing issues about mathematical knowledge and proof. Some initial questions here are as follows:

P10: What is the general significance of the **P** ≠ **NP** problem to mathematics and its practice?

P11: What would the implications be if **P** ≠ **NP** were either proven, refuted, or shown to be formally undecidable?

P12: Why has **P** ≠ **NP** proven so hard to settle? Does this tell us anything about the relationship between propositional and problem difficulty?

Questions P10 and P11 are foreshadowed in remarks made by Kurt Gödel in what is now a well-known letter he wrote to John von Neumann in 1956:[41]

> Obviously, it is easy to construct a Turing machine that allows us to decide, for each formula F of the restricted functional calculus and every natural number n, whether F has a proof of length n [length = number of symbols]. Let $\psi(F, n)$ be the number of steps required for the machine to do that, and let $\varphi(n) = \max_F \psi(F, n)$. The question is, how rapidly does $\varphi(n)$ grow for an optimal machine? It is possible to show that $\varphi(n) > Kn$. If there really were a machine with $\varphi(n) \sim Kn$ (or even just $\sim Kn^2$) then that would have consequences of the greatest significance. Namely, this would clearly mean that the thinking of a mathematician in the case of yes-or-no questions could be completely[42] replaced by machines, in spite of the unsolvability of the *Entscheidungsproblem*. n would merely have to be chosen so large that, when the machine does not provide a result, it also does not make any sense to think about the problem. Now it seems to me to be quite within the realm of possibility that $\varphi(n)$ grows that slowly. For 1) $\varphi(n) > Kn$ seems to be the only estimate obtainable by generalizing the proof of the unsolvability of the *Entscheidungsproblem*; 2) $\varphi(n) \sim Kn$ (or $\sim Kn^2$) just means that the number of steps when compared to pure trial and error [*dem blossen Probieren*] can be reduced from N to $\log N$ (or $\log N^2$). Such significant reductions are definitely involved in the case of other finitist problems, e.g. when computing the quadratic remainder symbol by repeated application of the law of reciprocity. It would be interesting to know what the case would be, e.g., in determining whether a number is prime, and how significantly *in general* for finitist combinatorial problems the number of steps can be reduced when compared to pure trial and error.

In order to see the relation of Gödel's remarks to P10 and P11, it is useful to contrast the following two decision problems:

VALID Given a formula F of first-order logic, is F valid (i.e. true in all models)?

PROVABLE Given a first-order formula F and a natural number n, does there exist a derivation $\mathcal{D}$ of F from Hilbert's axioms for first-order logic such that $\mathcal{D}$ contains $\leq n$ symbols?

The first of these is the *Entscheidungsproblem* for first-order logic first posed by Hilbert and Ackermann (1928, §III.11) (and from which Gödel takes the name "restricted functional calculus" to refer to first-order logic). This is, of course, undecidable in virtue of Church's Theorem (1936). But

[41] Gödel's letter was not discovered until 1988, long after the basic concepts of complexity theory had been defined. For more on its historical context, see Sieg's introductory remarks to (Gödel 1956), (Sipser 1992), (Hartmanis 1993), and (Urquhart 2010).

[42] Here Gödel adds the footnote: "Except for the formulation of axioms".

due to the first-order soundness and completeness theorems, the question $F \in$ VALID is equivalent to the derivability of F from Hilbert's axioms. It thus follows that n-PROVABLE and VALID are related to one another in the sense that $F \in$ VALID just in case $\langle F, m\rangle \in n$-PROVABLE for some m. It is also easy to see on the basis of Turing's original undecidability proof (1936, §11) that VALID is complete for computably enumerable languages with respect to many-one reductions. On the other hand, n-PROVABLE is clearly decidable. For in order to decide $\langle F, m\rangle \in n$-PROVABLE, it suffices to enumerate all strings containing $\leq m$ symbols in the language of F and check if they are well-formed proofs with conclusion F. As each such check can clearly be performed in time polynomial in n, such a proof serves as a certificate for the membership of $\langle F, m\rangle$ in n-PROVABLE thus showing that n-PROVABLE $\in \mathbf{NP}$. But it is also not difficult to construct a reduction of SAT to n-PROVABLE which shows that n-PROVABLE is also $\mathbf{NP}$-complete.[43]

The definitions needed to state this result precisely would not be framed until 15 years after Gödel's letter. Nonetheless, his remark that the efficient solvability of n-PROVABLE would be of the "greatest significance" does indeed flag a point at which the $\mathbf{P} \neq \mathbf{NP}$ question comes into contact with traditional foundational concerns in mathematics. For if there were a polynomial time algorithm for n-PROVABLE, then we would be able to feasibly check whether a given first-order formula F follows from a given set of sentences Γ membership in which is itself decidable in polynomial time.

But now consider the case where Γ subsumes the mathematical axioms which we accept in practice (or even can *foresee* accepting) and F is the formalization of an open problem such as the Riemann Hypothesis.[44] If there were an algorithm which allowed us to efficiently decide $\langle \bigwedge \Gamma \rightarrow F, c\rangle \in n$-PROVABLE, then by choosing a sufficiently large c – e.g. 10^{12} – we could rule out the possibility that there is a proof of F from Γ which we could ever hope to understand or survey in practice.[45] As Gödel observes, in such cases "it would not make sense to think about F". And on this basis he reaches the conclusion that the existence of an $O(n)$ or $O(n^2)$ (or $O(n^k)$ by extrapolation) algorithm for n-PROVABLE would entail that in the case of decision algorithms the "thinking of a mathematician ... could be completely replaced by machines". But as he finally remarks, he has found no means of ruling out this eventuality, and considers it to be "in the realm of possibility" that such an algorithm might exist.

With respect to the first point, it is not hard to see that the existence of a polynomial time algorithm for n-PROVABLE does indeed run strongly counter to expectation. For the reason mathematicians continue to work on open problems in established frameworks presumably derives from their conviction that problems like the Riemann Hypothesis are not only decidable from conven-

[43] One such reduction – originally due to Cook and described by (Clote and Krajícek 1993, p. iv) – maps a propositional formula A containing atomic letters $X_1, \ldots, X_k$ into a first-order formula $r(A) = \exists x \exists y(Q(x) \wedge \neg Q(y)) \rightarrow \exists x_1 \ldots \exists x_k A'$ where $Q(x)$ is a new unary predicate and A' results form A by replacing X_i with $Q(x_i)$. It is easy to see that $r(A)$ can be constructed in time polynomial in the length of A and also that by understanding x, y as the truth values 0 and 1, $A \in$ SAT if and only if $r(A) \in$ VALID. Finally note that if A is satisfiable, then it is possible to demonstrate $r(A)$ via a "short" first-order derivation which begins by assuming $\neg r(A)$ and then instantiates $x_1, \ldots, x_k$ with a satisfying assignment for A to reach a contradiction in $O(n)$ steps. This demonstrates that $A \in$ SAT just in case $\langle r(A), p(|A|)\rangle \in n$-PROVABLE for a suitable polynomial $p(n)$.

[44] Although a natural choice for Γ would be ZFC (or perhaps an extension ZFC + L with some finite class of large cardinal axioms L), it is more convenient for what follows to assume that Γ is finitely axiomatizable (and thus the conjunction $\bigwedge \Gamma$ is a single sentence). In light of this, references to ZFC below can be replaced as needed with a conservative two-sorted extension such as GB (or a suitable finite extension thereof).

[45] For instance choosing $c = 10^{12}$ would be sufficient to show that there was no proof of F less than 200,000,000 standardly formatted pages in length from (say) the ZFC axioms. Even taking into account various "formalization overheads" (which could be eliminated via appropriate extensions by definition and lemmas), this is presumably a safe upper bound on the length of a proof which a human mathematician could comprehend in a lifetime.

tionally accepted axioms, but also that they may be proven or refuted on the basis of a proof which we can discover and survey in practice. Given the extensive effort which has been invested in such questions, it would thus be exceedingly surprising if there were an algorithm which allows us to efficiently test whether arbitrary mathematical statements are feasibly decidable from conventionally accepted axioms.

Of course such an algorithm would not itself allow us to mechanically *find* such a proof or refutation of the (e.g.) Riemann Hypothesis from the axioms of (e.g.) ZFC, even if it pronounced that one containing fewer than (e.g.) 10^{12} symbols existed. But it is easy to see how with the aide of such an algorithm, a knowledgable human mathematician could engage in an interactive proof search – e.g. by iteratively hypothesizing lemmas which could the be checked by the algorithm with varying length bounds – which would allow a feasibly sized proof or refutation to be efficiently discovered (presuming that one exists). And this would indeed come close to replacing the creative efforts of mathematicians (as we now understand them) with a quasi-mechanistic activity similar to that which Gödel seems to envision even in the case of non-"yes-or-no" questions.[46] In light of this, it is indeed surprising that Gödel explicitly remarks that it seems "quite within the realm of possibility" that there could exist a polynomial time algorithm for n-PROVABLE. For since n-PROVABLE is **NP**-complete, this would entail that $\mathbf{P} = \mathbf{NP}$.[47]

On the other hand, it has recently come to light that six years prior to Gödel's letter, John Nash (1950) independently formulated a similar sequence of observations while appearing to make the opposite conjecture. Nash's remarks also appear in a letter, this time written to the American National Security Agency addressing the challenges of designing secure cryptographic systems based on (what we would now call) privately shared keys:

> The most direct computation procedure [for breaking a cryptographic system] would be for the enemy to try all 2^n possible keys, one by one. Obviously this is easily made impractical for the enemy by simply choosing n large enough .... Now my general conjecture is as follows: for almost all sufficiently complex types of enciphering ... the mean key computation length increases exponentially with the length of the key ... The nature of this conjecture is such that I cannot *prove* it, even for a special type of ciphers. Nor do I expect it to be proven. But this does not destroy its significance. The probability of the truth of the conjecture can be guessed at on the basis of experience with enciphering and deciphering ... ¶ ... If qualified opinions incline to believe in the exponential conjecture, I think we (the U.S.) cannot afford not to make use of it. Also we should try to keep track of the progress of foreign nations towards "inbreakable" types of ciphers.

Since Nash is not explicit about the particular encryption methods he has in mind, it is not as clear whether we should regard his "exponential conjecture" as a precise anticipation of the $\mathbf{P} \neq \mathbf{NP}$

[46] See (Buss 1995, pp. 60-62) for further discussion of this possibility.

[47] Since Gödel was writing before **NP**-completeness had been defined, one explanation of his remark is that he failed to realize that the existence of a polynomial time algorithm for n-PROVABLE would entail the existence of such an algorithm for *all* of the "finitist combinatorial problems" he had mind. A more intriguing possibility is that he was using a version of the $\mathbf{P} \neq \mathbf{NP}$ problem to make a connection to his "disjunctive" thesis about mechanism – i.e. "either ... the human mind (even within the realm of pure mathematics) infinitely surpasses the powers of any finite machine, or else there exist absolutely unsolvable diophantine problems" (Gödel 1951, p. 310). A complexity theoretic version of this might read "with respect to decidable problems, either the human mind surpasses the powers of any reasonable model of computation, or else there exist intrinsically difficult search problems". Gödel is sometimes interpreted as favoring the former alternative in the original disjunction. But by holding open the possibility that $\mathbf{P} = \mathbf{NP}$, he could be understood as suggesting that the advantage of human's over mechanical computers is only operative for undecidable (i.e. non-recursive) problems rather than decidable ones which may initially appear infeasible.

problem. What is clear, however, is that he understood the potential *benefits* of the existence of intrinsically difficult search problems similar to ones which are now known to be **NP**-hard relative to the practical goal of designing secure cryptographic protocols. And in this context it becomes all the more significant that although he expressed a high degree of confidence that his conjecture was true, he also appears to have recognized that the content of the problem itself might provide some insight into why it is difficult to resolve. At least in incipient form, Nash thus appears to be the first person to have suggested that $\mathbf{P} \neq \mathbf{NP}$ might be hard in the sense of *problem difficulty*.

We are now in an improved position to understand the basis for such a prediction. For as we have already observed in §2.1, $\mathbf{P} \neq \mathbf{NP}$ is itself a mathematical statement. By taking advantage of the **NP**-completeness of a language such as SAT and familiar techniques from the arithmetization of syntax, it is in fact easy to see that $\mathbf{P} \neq \mathbf{NP}$ is equivalent to the following statement which can be expressed in the language of first-order arithmetic:

N: For all Turing machines T and $k \in \mathbb{N}$, there exists $\varphi \in$ PROP such that either $t_T(|\varphi|) \geq |\varphi|^k$ (i.e. T does not have polynomial running time) or $T(\ulcorner\varphi\urcorner) = 1$ and $\varphi \notin$ SAT or $T(\ulcorner\varphi\urcorner) = 0$ and $\varphi \in$ SAT (i.e. T does not decide SAT).

Note that if N were *false* (i.e. if $\mathbf{P} = \mathbf{NP}$) then there would be a polynomial time algorithm for n-PROVE which would allow us to decide whether N itself is derivable in (say) ZFC in fewer than (say) 10^{12} steps. And this would in turn most likely allow us to find a surveyable proof of $\neg N$ in the manner described above. On the other hand, if N is *true* there would exist no such algorithm. And in this case, we should expect it should be difficult to prove N – i.e. no easier than brute force search – absent particular insights we might have about its content which point us in the direction of a proof. A point on which Gödel and Nash appear to agree is that this is indeed the situation in which we find ourselves – i.e. there nothing about the *formulation* of the statement $\mathbf{P} \neq \mathbf{NP}$ which itself points towards a strategy for proving or refuting it.

Further to P12, these observations also highlight a sense in which $\mathbf{P} \neq \mathbf{NP}$ also has an apparent affinity to a self-referential statement about provability – i.e. if it is true, then it itself is most likely difficult to prove and if it is false then it itself is most likely easy to prove. This in turn suggests a potential analogy between $\mathbf{P} \neq \mathbf{NP}$ and well-known metamathematical statements which "assert their own unprovability" – e.g. the conventional Gödel sentence G_{PA} for a theory such as first-order Peano arithmetic (PA). There is indeed some value in exploring such analogies since it is easy to see that N is equivalent to a Π^0_2-statement in the language of PA – i.e. one of the form $\forall x \exists y A(x, y)$ where $A(x, y)$ contains only bounded quantifiers. And thus since its logical complexity (as measured by quantifier alternations) is greater than that of G_{PA} (which is Π^0_1), it cannot be ruled out on metamathematical grounds alone that N is undecidable in PA or even in far stronger systems such as ZFC (whose resources vastly outstrip those which are required to formulate it).

There are, nonetheless, several reasons to suspect that $\mathbf{P} \neq \mathbf{NP}$ is not independent of PA. For since N is a statement in the language of first-order arithmetic there is no chance that its truth value is indeterminate in the manner which is occasionally claimed for set theoretic statements such as the Continuum Hypothesis which are independent of ZFC.[48] And as N is not directly generated

[48] If we adopt the received view about the determinacy of the standard model of arithmetic, this follows immediately from the fact that $\mathbf{P} \neq \mathbf{NP}$ can be expressed as an arithmetical statement. But even if we do not adopt this assumption, it may also be noted that the Lévy-Shoenfield Absoluteness Lemma (Jech 2003, §25.20) show that the truth value of Σ^1_2-statements – and perforce all arithmetical ones – are absolute between L and V. This rules out the possibility that the truth value of N can be changed by forcing constructions of the sort which are occassionally used to argue for the indeterminacy of statements like CH on the basis of their undecidability in ZFC.

by a self-referential construction akin to Gödel's Diagonal Lemma, there is no apparent way in which the prior analogy can be made precise so as to show that N is *provably* equivalent to a statement asserting its own unprovability. In particular, the truth of N (or more precisely of the presumably weaker statement $\mathbf{NP} \neq \mathbf{coNP}$) only rules out the existence of so-called *polynomial bounded propositional proof systems* in which *all* tautologies have proofs which are polynomially proportion to their length (see, e.g. Segerlind 2007). But this is still compatible with the existence of a short proof of the *particular* statement formalizing $\mathbf{P} \neq \mathbf{NP}$, even from a fixed set of axioms. This underscores in yet another the way the difficulties involved in drawing conclusions about propositional difficulty from results or conjectures about problem difficulty as described in §2.1.

These considerations suggest that $\mathbf{P} \neq \mathbf{NP}$ cannot be shown to be independent in the manner of metamathematical Π^0_1-statements like G_{PA}. On the other hand, since N is a Π^0_2-statement this might raise the hope that it could be shown to be independent in a manner similar to arithmetical statements like Goodstein's Theorem or similar principles about functions which grow faster than those which can be proven to be computable in PA. But in this case metamathematical considerations intervene to show that if N could shown to be independent of PA in the manner of such prior independence proofs, then there would exist algorithms for $\mathbf{NP}$-complete search problems with running time $O(n^{f(n)})$ for all PA-provably computable functions $f(x)$. But as this would include ones with exceedingly slow rates of growth (e.g. the inverse Ackermann function), such a result would then entail that $\mathbf{P} = \mathbf{NP}$ would itself be "almost true". Unlike Π^0_1-statements – for which undecidability entails *truth* – the undecidability of N in PA would thus come close to entailing its *falsity*.[49]

5 Barriers

The foregoing observations suggest that if open separation questions such as $\mathbf{P} \neq \mathbf{NP}$ were undecidable from conventionally accepted axioms, then a proof of this fact would have to rely on methods for demonstrating formal independence which go beyond those currently known for arithmetic or set theory. Given the attention which these questions have attracted both inside and outside computer science, this observation should itself be of interest to philosophers of mathematics. But within complexity theory itself, it has given rise to systematic reflection on the limitations of methods which were originally introduced with the hope of yielding separation theorems. Specific results about these methods have come to be known as *barriers*.

It is currently standard to identify three such classes of limitative results which are respectively known as the *relativization*, *natural proofs*, and *algebrization barriers*. The latter two were inspired by work in circuit complexity and interactive proof systems which cannot be fully be surveyed here. On the other hand, the relativization barrier came first historically and is related to the use of the familiar technique of *diagonalization* by which Turing (1936) originally demonstrated the

[49] The sequence of observations (originally due to Ben-David and Halevi 1992) required to reach this conclusion can be summarized as follows: 1) if $\mathsf{PA} \nvdash N$, then it can be shown that $\mathsf{PA}+\mathbf{\Pi}^0_1 \nvdash N$ where $\mathbf{\Pi}^0_1$ is the set of all Π^0_1-sentences true in the standard model of arithmetic (this follows because it can be shown that adding true Π^0_1-statements to PA does not increase its provably computable functions); 2) in this case it can be shown that the so-called *approximation rate* of $\textsc{Sat}$ by $\mathbf{P}$ – i.e. $r(i) = \max_{j \leq i}\{\min\{|x| : \textsc{Sat}(x) \neq M_j(x)\}\}$ where $M_0, M_1, \ldots$ is some fixed polynomial time computable enumeration of machines deciding languages in $\mathbf{P}$ – is not bounded by any provably computable function of PA; 3) using this fact – which intuitively means that in any polynomial time computable enumeration of $\varphi_0, \varphi_1, \ldots$ of PROP there will be "long" intervals on which $\textsc{Sat}$ is correctly computed by a Turing machine which runs in polynomial time – it is possible to specify an $O(n^{g(n)})$-time algorithm which solves the search problem "on input $x \in \textsc{Sat}$, find a satisfying valuation v" for any polynomial time computable $g(n)$ bounding $r^{-1}(n)$.

undecidability of the Halting Problem. This method was later adapted by Kleene (1943; 1955) to prove separation results in computability theory – i.e. the properness of the arithmetical and analytical hierarchies – and by Hartmanis and Stearns (1965) to demonstrate the Time Hierarchy Theorem described in §2.4 which yields results such as $\mathbf{P} \neq \mathbf{EXP}$.

For purposes of describing relativization as a barrier, it is useful to characterize a *proof by diagonalization* for a model of computation $\mathfrak{M}$ as one which satisfies three properties. First, such a proof is able to make use of the fact that $\mathfrak{M}$ may be uniformly enumerated as $M_0, M_1, M_2, \ldots$ – e.g. in the conventional manner of an indexation of Turing machines. Second, a proof by diagonalization is also able to make use of the fact that there is a *universal machine* $M_u \in \mathfrak{M}$ by which it is possible to simulate the computation of a given machine M_i on a given input x – i.e. so that $M_u(i, x) = M_i(x)$ – within a given time or space bound. Third, these are the *only* properties about $\mathfrak{M}$ which the proof may assume to hold – i.e. it must otherwise treat the machines in $\mathfrak{M}$ as "black boxes" in the sense that the details of the proof cannot rely on their specific mode of operation.[50]

It is conventionally taken to be a consequence of the third property that any result which can be shown via diagonalization to hold for a complexity class $\mathbf{C}$ defined in terms of $\mathfrak{M}$ can also be shown to hold for the class $\mathbf{C}^A$ which is derived from $\mathbf{C}$ by allowing the machines in $\mathfrak{M}$ access to an arbitrary set $A \subseteq \{0,1\}^*$ as an *oracle* – i.e. a set for which we assume that there is an *oracle machine* $M \in \mathfrak{M}^A$ which can decide membership queries about A in a single step. This is assumed to be so because adding the ability to compute $\chi_A(x)$ – i.e. the characteristic function of A – as a primitive operation to the model $\mathfrak{M}$ will not in general affect whether a proof involving $\mathfrak{M}$ satisfies the first and second properties described above.[51]

It can be shown on this basis that separation results which follow from the Hierarchy Theorems relativize. For instance, for all oracles $A \subseteq \{0,1\}^*$, $\mathbf{P}^A \neq \mathbf{EXP}^A$ – i.e. the classe of languages decidable in polynomial time relative to an oracle is always distinct from those decidable in exponential time relative to the same oracle. It thus follows that if $\mathbf{P} \neq \mathbf{NP}$ were provable by a similar form of diagonalization proof, then we also expect that $\mathbf{P}^A \neq \mathbf{NP}^A$ for all oracles. However a classical result of Baker, Gill, and Solovay (1975) gives the existence of sets $B_1, B_2 \subseteq \{0,1\}^*$ such that $\mathbf{P}^{B_1} = \mathbf{NP}^{B_1}$ and $\mathbf{P}^{B_2} \neq \mathbf{NP}^{B_2}$.[52] Baker et al. also constructed oracles which give similarly conflicting results for

[50] For instance, consider the traditional proof of the undecidability of the Halting Problem for the Turing machine model, which also shows that the recursive (i.e. Δ_1^0-definable) sets are properly included in the recursively enumerable (i.e. Σ_1^0-definable) sets. Here we define the set $K = \{M_u(x,x) \downarrow : x\}$ and then show that the assumption that K is recursive leads to a contradiction. Note that while the definition of K uses the first and second properties of diagonalization proofs for the Turing machine model $\mathfrak{T}$, it also satisfies the third as it is does not otherwise depend on how Turing machines operate in a step-by-step manner.

[51] For instance in the case of the oracle variant $\mathfrak{T}^A$ of the Turing machine model, the characteristic function of A is assumed to be written on a read-only "oracle tape" which an oracle machine $T^A \in \mathfrak{T}^A$ can consult during its computation and then read, write (to other tapes), or move its head accordingly. Adding such a tape effects neither our ability to enumerate the machines in $\mathfrak{T}^A$ nor the availability of a universal machine (which is now also assumed to have access to A on its oracles tape). A consequence of these observations is that versions of the Enumeration and Normal Form Theorems hold for oracle Turing machines as well (Rogers 1987, §9.3). With these results in place, the proof of the fact that the sets recursive in A are properly included in the sets recursively enumerable in A (Rogers 1987, Theorem 9.X) precisely mirrors the proof that the recursive sets are properly included in the recursively enumerable sets (Rogers 1987, Theorem 5.VI).

[52] For B_1 it is sufficient to take a language complete for **PSPACE**. For in this case we will have $\mathbf{P}^{B_1} = \mathbf{PSPACE} = \mathbf{NPSPACE} = \mathbf{NP}^{B_1}$ where the middle equality is provided by a well-known theorem of Savitch (1970). B_2 can constructed by a diagonal procedure which forces any deterministic Turing machine deciding a particular language X (which is defined relative to B_2) to make $O(2^n)$ oracle queries but which at the same time can be decided by a single non-deterministic query to B_2.

the closure of the relativized version of **NP** under complementation (even if the relativized version of **P** and **NP** are distinct) and of coincidence of the relativized versions of **NP** and **coNP**.

Results in this spirit suggest that techniques resembling the traditional method of diagonalization are indeed incapable of deciding the major outstanding separation questions in complexity theory. This conclusion can be sharpened in virtue of Arora, Impagliazzo, and Vazirani's (1992) axiomatic characterization of such methods. The resulting theory – which Arora et al. refer to as RCT for *Relativizing Complexity Theory* – abstracts away from the particular machine models used to define complexity classes in favor of axioms which implicitly characterize such classes in terms of their closure properties in a manner similar to Cobham's (1965) original characterization of **P**. RCT additionally includes an axiom formalizing the existence of a time bounded universal function of the sort we have seen is involved in diagonalization proofs.

It can be shown that the class $\mathcal{FP}$ of functions computable in polynomial time is the minimal model of RCT and also that this theory is capable of proving relativizing separation results such $\mathbf{P} \neq \mathbf{EXP}$ – i.e. statements those whose truth value is not affected by access to an oracle and will thus be true "in the real world" when no oracle is available. More generally, however, a class of functions $\mathcal{X}$ is a model of RCT if and only if $\mathcal{X} = \mathcal{FP}^A$ – i.e. if it is the result of closing $\mathcal{FP}$ together with the characteristic function for some oracle $A \subseteq \{0,1\}^*$. It thus follows that there exist models $\mathcal{X}_1$ and $\mathcal{X}_2$ of RCT which are derived from Baker et al.'s oracles B_1 and B_2 such that the statement formalizing $\mathbf{P} \neq \mathbf{NP}$ is false in $\mathcal{X}_1$ but true in $\mathcal{X}_2$. And from this it follows that non-relativizing statements such as $\mathbf{P} \neq \mathbf{NP}$ are formally independent of RCT.

One moral which is commonly derived from these observations is that a successful proof of a statement like $\mathbf{P} \neq \mathbf{NP}$ must make use of *specific* features of $\mathfrak{T}$ (or some similar model in the first machine class) beyond the effective enumerability of its members and the existence of efficient simulations by a universal machine. A potentially relevant observation in this regard is that the operation of Turing machines is "local" in the sense that it can be determined whether one global state follows via a single transition from another by examining a fixed-width "window" of information about the machine's internal state, tape contents, and head position. This fact is crucially exploited in the proof of the Cook-Levin Theorem – i.e. the **NP**-completeness of 3-SAT – to show how the step-by-step operation of a Turing machine can be described by a propositional formula (see, e.g. Arora and Barak 2009, §2.3). As a consequence, this result is also a paradigmatic example of a relativizing theorem.

At present we do not know how to harness such observations to prove the desired separations. But they appear to be related to a widespread phenomenon concerning sensitivities in the formulation of **NP**-complete problems themselves. For instance, while 3-SAT is **NP**-complete, 2-SAT and HORNSAT (where each clause is a disjunction of non-negated propositional variables together with a single negated one) are both in **P**. And while it might seem as if any algorithm for deciding 3-SAT must require $O(2^n)$ steps (since, naively, every row in the truth table for a formula must be checked in order to determine whether it is satisfiable), there is in fact a decision algorithm which runs in time $O(1.476^n)$ (Rodošek 1996).[53] Another sort of obstacle to proving $\mathbf{P} \neq \mathbf{NP}$ is that a

[53]Such contrasts are illustrative of a wide class of cases in which modifying an apparently arbitrary parameter in the formulation of a problem changes it from being "easy" to being "hard" (at least as far as we currently known). For instance, although deciding whether a graph is 3-colorable in **NP**-complete, deciding within it is 2-colorable is decidable in linear time and although finding the maximum clique in a graph is an **NP**-complete search problem, finding a maximum matching can be accomplished in polynomial time. Aaronson (2016) has describing cases like this as illustrating an "invisible fence" surrounding **NP**-complete problems for which we currently lack a uniform explanation.

correct proof must be responsive to such results in the sense of not "proving too much" – e.g. that 2-SAT $\notin$ **P** or that there is an $\Omega(2^n)$ lower bound on algorithms for 3-SAT.

The isolation of both the natural proofs and algebrization barriers can be understood as deriving from attempts to develop proof techniques which do not run afoul of these difficulties and which also avoid the relativization barrier. One way in which the former goal can potentially be achieved is by recharacterizing a decision algorithm for a language X in terms of a family of *boolean circuits* – i.e. families $\mathcal{C} = \{C_n : n \in \mathbb{N}\}$ of finite networks of boolean logic gates computing function $f : \{0,1\}^* \to \{0,1\}$ such that C_n decides membership in X for all strings of length X. This leads to the complexity class known as **P/poly** consisting of those languages X which have polynomial sized circuits – i.e. there exists such a family $\mathcal{C}$ deciding X for which $|C_n| < n^k$ for some $k \in \mathbb{N}$. This class is *non-uniform* in the sense that each C_n can be thought of as implementing a distinct algorithm for deciding length n inputs. On the other hand, not only is locality enforced directly by the definition of a boolean circuit, the circuit model is sufficiently simple to provide some hope that lower bounds might be provable by essentially combinatorial arguments.

In the 1980s and 1990s some progress was in fact made in this manner leading, e.g., to the result of Ajtai (1983) and Furst et al. (1984) that the language PARITY – consisting of the strings containing an odd number of 1s – is not in the class $\mathbf{AC}^0$ consisting of languages decidable by circuits of constant depth, polynomial size, and whose gates have unbounded fan-in. On the other hand, the methods used in this and related proofs were subsequently analyzed in detail in a highly influential paper of Razborov and Rudich (1997) entitled "Natural proofs". Therein it was shown that the proof of PARITY $\notin \mathbf{AC}^0$ and several similarly promising results in circuit complexity all possessed what Razborov and Rudich dubbed "natural properties" of a sort which render them incapable of proving separation results such as $\mathbf{NP} \subsetneq \mathbf{P/poly}$ or $\mathbf{P} \neq \mathbf{NP}$ relative to a widely believed conjecture about the existence of pseudorandom generators.[54]

The route to the algrebrization barrier began by reflecting on how oracles interact with newer non-relativizing results. A notable example is the theorem of Shamir (1990) that the class **IP** corresponding to the languages decidable by so-called *interactive proof systems* (see §6) coincides with **PSPACE**. However a more recent result of Aaronson and Wigderson (2009) demonstrates that while the techniques used to show **IP** = **PSPACE** avoid the original form of the relativization barrier, they still relativize with respect to a more sophisticated class of so-called *algebraic oracles* in a manner which again renders them incapable of proving $\mathbf{P} \neq \mathbf{NP}$.

The work just surveyed spans a period of more than 40 years during which complexity theorists have engaged in a process of methodological reflection which can be schematized as follows: 1) a technique is proposed for proving separations between classes (e.g. diagonalization or circuit lower bounds); 2) the technique is then applied to yield some initial positive results (e.g. $\mathbf{P} \neq \mathbf{EXP}$ or PARTITY $\notin \mathbf{AC}^0$); 3) experience with the methods in question leads to the discovery of limitative results which suggest that they are incapable of proving the separations we currently believe to be

[54] Very roughly, Razborov and Rudich's definition of a natural property P is one which holds of a boolean function $f : \{0,1\}^* \to \{0,1\}$ just in case it is *constructive* in the sense sense that $P(f)$ can be decided in time polynomial in $f \upharpoonright n$ and that P holds for a sufficiently *large* proportion of functions of this type. Such a property is said be *useful* against a complexity class **C** just in case just in case P fails to hold of the characteristic functions of infinitely many languages in **C**. Razborov and Rudich showed that the proof of PARITY $\notin \mathbf{AC}^0$ satisfies these properties. They then showed that the existence of *pseudorandom generators* – i.e. efficient procedures which enumerate functions $f : \{0,1\}^n \to \{0,1\}$ which are indistinguishable from "truly" random functions even by algorithms which can examine their entire graph in time $O(2^n)$ – implies that no natural proof can be useful against $\mathbf{NP} \notin \mathbf{P/poly}$. (Although it is at present also unknown whether psuedorandom generators exist, this assumption is crucial for the security of currently deployed cryptographic systems and is widely believe to be true – see, e.g. Arora and Barak 2009, §9.)

"hard" (e.g. $\mathbf{P} \neq \mathbf{NP}$ or $\mathbf{NP} \subsetneq \mathbf{P/poly}$); 4) a first round of reflection leads to the isolation of what are claimed to be the essential features of the technique (e.g. enumerability and efficient simulability in the case of relativization or the *constructivitiy* and *largeness* properties of natural proofs described by Razborov and Rudich (1997); 5) a second (optional) round of reflection leads to an axiomatic characterization of the technique which allows both for a formal derivation of the positive results and also a demonstration that the "hard" separations are formally independent of the axioms in question (e.g. the theory RCT of (Arora et al. 1992) or the theory ACT proposed by (Impagliazzo et al. 2009) as a formalization of the algebrization barrier); 6) a final round of reflection points towards novel methods of proof which have the promise of avoiding the barrier characterized by the prior steps (e.g. the program of Mulmuley described below).

This process highlights a number of conceptual questions:

P13: In what ways is the study of barriers similar or dissimilar to prior episodes in the history of mathematics during which there has been prolonged reflection on the inability of various proof techniques to answer an open question – e.g. squaring the circle or proving that π is trancedental? Is it possible to provide a general account of what it means to *axiomatize* a class of proof techniques? In cases where such axiomatizations are available, in what ways are barrier results like and unlike traditional instances of formal independence? Is it possible to provide a general account of how the isolation of barriers has sharpened our understanding of why open problems such as $\mathbf{P} \neq \mathbf{NP}$ are hard in the sense of propositional difficulty or of the concept of feasible computation more generally?[55]

In parallel to these issues, reflecting on barriers also raises additional technical questions of which the following is perhaps most immediate:

T5: Is it possible to find a proof technique which avoids the relativization, natural proofs, and algrebrization barriers simultaneously? Would finding such a method be sufficient to resolve $\mathbf{P} \neq \mathbf{NP}$ or other currently open separation questions?

At the time of writing, the consensus is that the most promising approach to T5 is the program of *geometric complexity theory* which was initiated by Ketan Mulmuley in the 1990s and which eventually led to a series of papers beginning with (Mulmuley and Sohoni 2001). This approach seeks to combine ideas from algebraic geometry and geometric invariant theory to prove new circuit lower bounds – e.g. an initial target has been a conjecture of Valiant (1979) about the size of arithmetical circuits required to compute the permanent of matrices over fields of characteristic greater than two. As even this brief description indicates, geometric complexity theory seeks to bring together ideas and results from several core areas of mathematics together with the work which led to the isolation of the natural proofs barrier. The reader is directed to the surveys (Regan 2002), (Mulmuley 2012), and (Aaronson 2016, §6.6) for further references and an overview of the rich array of techniques and conjectures which geometric complexity theory has inspired together with an account of why it is hoped that it can both avoid the barriers described above while also failing to give rise to novel ones.

[55]The natural proofs barrier in particular is often described as doing precisely this. For by proving that certain problems are "hard" relative to natural proofs, Razborov and Rudich's result can be understood as illustrating that there is another sense in which they become "easier" in virtue of admitting to a certain kind of description. Aaronson summarizes this phenomenon as follows: "Here, perhaps, we are finally face-to-face with a central conceptual difficulty of $\mathbf{P} \neq \mathbf{NP}$ ... [W]e're trying to prove that certain functions are hard, but the problem of deciding whether a function is hard is *itself* hard, according to the very sorts of conjectures that we're trying to prove" (2016, p. 58).

6 Non-classical models of computation and proof

The previous sections have presented an overview of computational complexity theory which is highly "classical" in two different respects. First, with the partial exception of §5, the work surveyed here was carried out before the 1990s. Second, in attempting to present a narrative which illustrates the significance of problems like $\mathbf{P} \neq \mathbf{NP}$ to traditional questions about the foundations of mathematics, little attention has been paid to the large amount of work on what are sometimes described as "non-classical" modes of computation. The goal of this final section will be to provide a small glimpse of this wider context while also highlighting how such work bears on traditional characterizations of mathematical knowledge and proof.

The notion of "classicality" at issue can be illustrated by the use of a deterministic Turing machines $T \in \mathfrak{T}$ to solve a decision problem $X \subseteq \{0,1\}^*$ or compute a function of type $f : \{0,1\}^* \to \{0,1\}^*$. The use of coding techniques described in §2 makes it possible to interpret T as deciding a set $A = \{\ulcorner x \urcorner^{-1} : x \in X\}$ or computing a function $g(\ulcorner x \urcorner^{-1}) = \ulcorner f(x) \urcorner^{-1}$ defined on the sorts of objects standardly considered in discrete mathematics – e.g. natural numbers, graphs, formulas, etc. Since $\mathfrak{T}$ is in the first machine class, it is also reasonable to assume that we can carry out the computation of T on a given input $x \in \{0,1\}^*$ of feasible length in practice, either by hand or using a computer implementation. By performing such computations, we thus are able to learn specific facts about membership in A or the values of f via a process which can be assimilated to a traditional mathematical proof. In particular, it presents no difficulty in principle to transform a computation carried out by T into a formal derivation of $x \in A$ or $f(x) = y$ in an axiomatic system such as Peano arithmetic – e.g. by using the Kleene T-predicate from computability theory to express facts about the result of applying A to x in the language of first-order arithmetic and then reasoning axiomatically about its operation.[56]

Some of the other models of computation which are studied in complexity theory do not fit in comfortably with this picture. For instance, we have seen that the parallel RAM model $\mathfrak{P}$ has members which run in polynomial time but whose computation we cannot carry out in practice for even more moderately sized inputs (at least if $\mathbf{P} \neq \mathbf{PSPACE}$). Similarly, the operation of non-deterministic models such as $\mathfrak{N}$ are intended to be interpreted in a manner which prevents us from employing them as practical means of deciding certain languages (at least if $\mathbf{P} \neq \mathbf{NP}$). Nonetheless, these formalisms play an important role as *notional* models of computation – e.g. in helping us to to delimit the notion of a *reasonable* model (as discussed in §2.2) or in defining complexity classes which characterize problems which can be solved by brute force search (as discussed in §2.3 and §2.4). But these models are unlike $\mathfrak{T}$ in the sense that they are not regarded as formalizing modes of computation which we can carry out in practice, at least in the general case.

Many models which are currently studied in complexity theory can be understood as borderline cases sitting between formalisms which can be employed in practice to yield unconditional knowledge of mathematical propositions and those which cannot. A paradigmatic example of such a model is

[56] See, e.g., (Kleene 1952, §57). This account also presumes that we are able to prove T together with $\ulcorner \cdot \urcorner$ is *correct* with respect to our prior specification of $f(x)$ relative to such an arithmetization – i.e. the input-output mapping determined by operation of A coincides with $f(x)$ or the characteristic function of X. Since in complexity theory we are typically interested in *decidable* languages, *recursive* functions, and *feasibly computable* (and invertible) encodings, it is not a significant idealization to assume that the relevant demonstration will be available. Of course other issues in the vicinity of those popularized by Tymoczko (1979) may arise about in the cases where the length of the computation sequence of T on x (and thus also the corresponding formal proof) is "unsurveyably long". But again, these will not characteristically arise in the cases where T implements an algorithm we are able to carry out in practice.

the so-called *probabilistic Turing machine* model $\mathfrak{C}$. Such a device C differs from a conventional Turing machine by having access to a random number generator which produces a new bit at each step in its computation. The action taken by C at the next step is then determined by the value of this bit, in addition to its internal state and the currently scanned symbol.

The class **BPP** (or *bounded-error probabilistic polynomial time*) is defined to include the problems X for which there exists a probabilistic Turing machine $C \in \mathfrak{C}$ and a constant $\frac{1}{2} < p \leq 1$ with the following properties:

i) C runs in polynomial time for all inputs;

ii) for all inputs $x \in X$, at least fraction p of the possible computations of C on x accept;

iii) for all inputs $x \notin X$, at least fraction p of the possible computations of C on x reject.

Presuming that we have access to a source of genuine randomness – e.g. a fair coin or quantum random number generator – then it is possible in practice to carry out an individual computation described by a probabilistic Turing machine. The preceding definition thus can be understood as seeking to characterize the class of problems which may be feasibly decided by a probabilistic algorithm as those for which there exists such a procedure which decides the problem correctly in a "clear majority" of cases – i.e. with probability p bounded away from $\frac{1}{2}$. It is not difficult to see that if we possessed an algorithm for deciding X which is implementable by a machine C satisfying i)-iii), then we would correctly decide if $x \in X$ with probability $1 - \varepsilon$ (for arbitrarily small ε) by applying the algorithm repeatedly and checking whether the majority of its computations are accepting.[57]

For a long time it was believed that the existence of a well-known probabilistic algorithm for PRIMES due to Miller and Rabin (1980) provided evidence that $\mathbf{P} \subsetneq \mathbf{BPP}$. But the hope that this would lead to a separation proof was dashed by the discovery of the deterministic AKS primality algorithm (Agrawal et al. 2004) which showed that PRIMES $\in \mathbf{P}$. There are, however, still problems for which the most efficient known classical (i.e. non-probabilistic) algorithms have exponential time complexity but which have polynomial time randomized algorithms. A prominent current example is that of determining whether a polynomial with integer coefficients is equal to the zero polynomial over a finite field $\mathbb{F}$ (ZEROP). This problem is decidable in bounded-error probabilistic polynomial time by a procedure known as the Schwartz-Zippel algorithm. On the other hand, no classical algorithm is known for this problem which improves upon the naive exponential time brute force search over $\mathbb{F}$.[58]

[57] In order to use such a procedure to decide membership correctly with probability $1 - \varepsilon$, it suffices to apply the machine C at least $1/(p - \frac{1}{2})^2 \ln(1/\sqrt{\varepsilon})$ times and then take the majority output (see Chernoff 1981). This bound shows that the probability of making an incorrect decision using this method decreases quickly relative to the number of applications – e.g. if $p = \frac{3}{4}$ and $\varepsilon = .00001$, then 92 applications will be sufficient and if $\varepsilon = 1.0 \times 10^{-10}$, then 185 applications will be sufficient.

[58] ZEROP is often described as the problem of *polynomial identity testing* as the equality of $p(x_1, \ldots, x_n) = q(x_1, \ldots, x_n)$ over $\mathbb{F}$ is equivalent to $p(x_1, \ldots, x_n) - q(x_1, \ldots, x_n) \in$ ZEROP. (It specifying the problem, it must also be assumed that instances of ZEROP are represented in the form of *algebraic circuit* – i.e as a directed acyclic graph with each input node labeled by one of the variables $x_1, ..., x_n$, and each internal node has in-degree two and is labeled with one of $+, -$ or $\times$. Such an input encoding is needed to allow high degree polynomials to have short representations). The brute force algorithm for deciding $p(x_1, \ldots, x_n) \notin$ ZEROP requires checking if $p(a_1, \ldots, a_n) = 0$ for all sequences $a_1, \ldots, a_n \in \mathbb{F}^n$. On the other hand, it follows from the Fundamental Theorem of Algebra that a degree-k polynomial can have at most k roots. The basis of the Schwartz-Zippel algorithm is thus that if $\deg(p)$ is small relative to the order of $\mathbb{F}$, the probability that $p(a_1, \ldots, a_n) = 0$ for a random sequence of elements is low if $p(x_1, \ldots, x_n) \notin$ ZEROP. See, e.g. (Shpilka et al. 2010).

As a consequence, there currently exist specific polynomials $p(x_1, \ldots, x_n)$ to which we can apply the Schwartz-Zippel algorithm to show (say) $p(x_1, \ldots, x_n) \in$ ZEROP with probability $1 - \varepsilon$ (for arbitrarily small ε) while at the same time being unable to apply the corresponding brute force search to demonstrate this proposition holds with probability 1. It is also possible to show classically that the Schwartz-Zippel algorithm is correct in the appropriate probabilistic sense. Nonetheless such instances illustrate cases in which we cannot convert the computations which we have carried out into a conventional mathematical proof that $p(x_1, \ldots, x_n) \in$ ZEROP in the manner described above. This situation raises questions such as the following:

P14: What can be said about our epistemic or doxastic situation with respect to the proposition expressed by $p(x_1, \ldots, x_n) \in$ ZEROP in such a case? For instance, how should the credence we place in such a statement be compared with that which we place in other mathematical propositions based on our other standing practices of proof and computation? How is this related to the value of ε and our ability to reduce its value via additional application of the Schwartz-Zippel algorithm?

Unlike many of the questions posed in prior sections, those in the vicinity of P14 *have* been systematically discussed by philosophers. For instance Fallis (1997) argued that there are no epistemic grounds on which the probabilistic computation by which we might come to believe that $p(x_1, \ldots, x_n) \in$ ZEROP with high probability can be rejected as a means of yielding genuine mathematical *knowledge* which does not also rule out other methods conventionally deemed adequate for this purpose (e.g. accepting propositions on the basis of incomplete proofs or ones which are so long or complicated that we are forced to carry them out using a computer).[59] However the original discussion of these issues concerned the specific case of the Miller-Rabin primality algorithm. The discovery of the AKS algorithm together with ongoing work on *derandomization* – i.e. the process of replacing the truly random choices employed by probabilistic algorithms with ones determined by a suitable pseudorandom generator – is now also presented as supporting the conjecture that **P** = **BPP**. This in turn diminishes somewhat the significance of the issues surrounding P14 to traditional discussions of mathematical knowledge.

Nonetheless, further reflection on $\mathfrak{C}$ and similar models points towards a variety of related questions about the role which computation plays in our practices which have not been systematically explored. For instance, in remarking on the sort of case just described Harel (2006, p. 314) makes the following pragmatic suggestion: "As long as we use probabilistic algorithms only for petty, down-to-earth matters such as wealth, health, and survival, we can easily make do with very-likely-to-be-correct answers to our questions." Such observations underscore various ways in which computation may figure in our practices outside of what would traditionally be described as pure mathematics – e.g. in employing a probabilistic algorithm to confirm or disconfirm a statement whose truth value is relevant to our practical interests but for which we believe it to be infeasible (or are otherwise disinclined) to seek a traditional proof or refutation.

Many of the models which have been studied by complexity theorists in the past decades similarly seek to extend the way in which we might hope to employ other modes of computation to achieve practical ends. Some salient examples are as follows:

Feasible parallel computation (e.g. Arora and Barak 2009, §6): We considered in §2.2 the parallel RAM machine model $\mathfrak{P}$ and also observed that if we were to define time and space complexity in the contextual manner relative to this machine (rather than $\mathfrak{T}$), then we would have that **P** = **PSPACE**

[59] Discussion of this question is continued in, e.g. (Corfield 2004), (Easwaran 2009), and (Baker 2017),

(collapsing what we currently believe to be a vast gulf in feasibility). On the other hand, a number of problems of practical import admit to efficient parallel algorithms in the sense that they can be solved in time $O(\log^c(n))$ using $O(n^k)$ processors (for fixed $c, k \in \mathbb{N}$). In this case, the amount of *work* – i.e. the sum of the number of steps required by each processor – performed by a machine satisfying this definition will be polynomial in the size of its input. This definition leads to a complexity class known as **NC** which contains such problems as matrix multiplication, graph reachability, and maximal matching.

Quantum computation (e.g. Arora and Barak 2009, §10): It is also possible to define a model of probabilistic computation by taking as a machine model not $\mathfrak{C}$, but rather a model known as the *quantum Turing machine* $\mathfrak{Q}$. Whereas machines in $\mathfrak{C}$ have access to a primitive source of randomness, machines in $\mathfrak{Q}$ are given by an effectively presented sequence of unitary matrices (representing so-called *quantum gates*) which are successively applied to states consisting of superpositions of so-called *qubits* (i.e. physical systems which can be in two entangled states at once with given probabilities). The output of a quantum Turing machine is determined by a taking a measurement of its final state, thereby collapsing the superposition so as to probabilistically yield an output of 0 or 1. We define *Q decides language X* in parallel to the definition given above for $\mathfrak{C}$ and also the class **BQP** to be the class of languages accepted by some $Q \in \mathfrak{Q}$ which operates in polynomial time. As there is a well-known quantum polynomial time procedure for deciding FACTORS (i.e. *Shor's algorithm*), this in turn is often presented as evidence in favor of the conjecture that $\mathbf{P} \subsetneq \mathbf{BQP}$. However it is also conjectured that $\mathbf{NP} \subsetneq \mathbf{BQP}$.

Interactive proofs (e.g. Arora and Barak 2009, §8): An *interactive proof system* is a form of two-way communication protocol by which a computationally-unbounded but untrustworthy agent known as the *prover* attempts to convince a skeptical polynomial-bounded agent known as the *verifier* that some mathematical statement is true. More formally, the class **IP** is defined to be the set of languages $X \subseteq \{0,1\}^*$ for which there is a probabilistic polynomial time algorithm which allows the verifier to decide if $x \in \{0,1\}^n$ is in X by asking $O(n^k)$ queries to the prover and which satisfies the following properties: i) if $x \in X$, then there is a strategy for the prover to convince the verifier to accept with probability $p > 1/2$; ii) if $x \notin P$, then regardless of the prover's strategy, the verifier rejects with probability $p > 1/2$. It is easy to see that $\mathbf{NP} \subseteq \mathbf{IP}$ as in this case the prover can use his unbounded power to return a polynomial-size string which is a potential certificate for $x \in X$ which the verifier can then check in polynomial time, accepting if x is a genuine certificate and rejecting otherwise. By designing increasingly ingenious protocols, it can also be shown that all languages in **PSPACE** can be decided in this manner – i.e. $\mathbf{IP} = \mathbf{PSPACE}$ (Shamir 1990).

Probabilistic proofs (e.g. Arora and Barak 2009, §11): A *probabilistically checkable proof* is a type of randomized algorithm for efficiently deciding membership of a string x in a language X based on a bounded amount of information extracted from a conventional proof that $x \in X$. Such an algorithm is required to accept correct proofs and reject incorrect proofs both with probability $p > 1/2$. A certificate y for x's membership in a language $X \in \mathbf{NP}$ may serve as such a conventional proof, since deterministically checking the entire certificate allows us to classically decide if $x \in X$. But we can also consider the case in which the algorithm is able to only consult a few randomly selected bits of y. This leads to the definition of the class of languages $\mathbf{PCP}[r(n), q(n)]$ consisting of those languages which can be correctly decided in the described sense by making at most $r(n)$ random decisions and by reading at most $q(n)$ bits of y. The well-known *PCP Theorem* of Arora et al. (1998) states that $\mathbf{NP} = \mathbf{PCP}[O(\log(n)), O(1)]$ – i.e. for any language $X \in \mathbf{NP}$, X can be correctly decided with

probability $p > 1/2$ by a probabilistic algorithm which makes $O(\log(|x|)$ random decisions about how to sample a fixed number of bits of a potential certificate y demonstrating $x \in X$.[60]

Like $\mathfrak{C}$, the formalisms just described were introduced in an effort to sharpen our understanding of the limits of classical feasible computation. But unlike, e.g., $\mathfrak{P}$ these models are of interest in part because they formalize various modes of computation which we might hope to employ in practice, at least in particular cases. For instance, it is a topic of current research which algorithms can be usefully parallelized so as to achieve efficiency gains within **P** (although the currently favored conjecture that $\mathbf{P} \neq \mathbf{NC}$ would imply that certain problems in **P** are *inherently sequential* in the sense that they cannot be solved by algorithms which can be non-trivially sped up by parallelization). Attempts are also currently underway to construct quantum devices capable of carrying out Shor's quantum factorization algorithm for inputs sizes of practical concern (although it is currently beyond the capabilities of experimental physics to implement the algorithm to perform factorizations much beyond $15 = 3 \times 5$). Interactive or probabilistic protocols similar to those involved in the definitions of **IP** and **PCP** are also currently employed in cryptographic systems.

The study of such models has not to date yielded a substantial expansion of our mathematical knowledge in the traditional sense – e.g. the discovery of new integer factorizations or proofs of mathematical theorem. But like much recent work in complexity theory, the described models are of interest not only because they deepen our understanding of the limits of feasible computability, but also because they illustrate other ways in which computation might come to play new roles in our practices more generally. Providing a systematic characterization of this ongoing process is likely to be a complex undertaking in its own right. But the diversity of models currently under investigation underscores the importance of additional questions at the boundaries of computer science and philosophy of which the following are characteristic:

P15: Do the definitions of $\mathfrak{C}$/ **NC**/$\mathfrak{Q}$/**IP**/**PCP** provide faithful models of probabilistic/parallel/quantum/interactive/probabilistically verifiable modes of computation which we might hope to carry out in practice? How might reflection on these models lead to further refinements or revisions to how we currently characterize our epistemic or doxastic situation in mathematics? What other modes of computation might come to play a useful role in our practices in the future, both inside and outside mathematics?

References

S. Aaronson. Why philosophers should care about computational complexity. In J. Copeland, C. Posy, and O. Shagrir, editors, *Computability: Turing, Gödel, Church, and Beyond*. MIT Press, Cambridge, Mass., 2013a.

S. Aaronson. *Quantum computing since Democritus*. Cambridge University Press, 2013b.

S. Aaronson. $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$. In J. F. Nash and M. Rassias, editors, *Open problems in mathematics*, pages 1–122. Springer, Berlin, 2016.

[60] As we have seen in §5, the problem n-Provable which asks whether a given mathematical formula has a proof from axioms Γ of length less than n is in **NP**. Recall also that a correct derivation $\mathcal{D}$ such that $|\mathcal{D}| \leq n$ serves as a certificate for $\langle \varphi, n \rangle \in n$-Provable. A striking consequence of the PCP Theorem is thus that it is always possible to determine whether a claimed proof of φ is correct with high probability by examining only a small constant number of randomly chosen bits from $\mathcal{D}$.

S. Aaronson and A. Wigderson. Algebrization: A new barrier in complexity theory. *ACM Transactions on Computation Theory (TOCT)*, 1(1):1–54, 2009.

M. Agrawal, N. Kayal, and N. Saxena. PRIMES in **P**. *Annals of Mathematics. Second Series*, 160 (2):781–793, 2004.

M. Ajtai. Σ^1_1-formulae on finite structures. *Annals of pure and applied logic*, 24(1):1–48, 1983.

S. Arora and B. Barak. *Computational complexity: a modern approach*. Cambridge University Press, Cambridge, 2009.

S. Arora, R. Impagliazzo, and U. Vazirani. Relativizing versus nonrelativizing techniques: the role of local checkability. *Manuscript, University of California, Berkeley*, 1992.

S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM (JACM)*, 45(3):501–555, 1998.

J. Avigad and R. Morris. Character and object. *The Review of Symbolic Logic*, 9(3):480–510, 2016.

A. Baker. Non-deductive methods in mathematics. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 2017 edition, 2017. URL https://plato.stanford.edu/entries/mathematics-nondeductive/.

T. Baker, J. Gill, and S. Solovay. Relativizations of the **P** = **NP**? question. *SIAM Journal on computing*, 4(4):431–442, 1975.

J. J. Bartholdi, C. A. Tovey, and M. A. Trick. The computational difficulty of manipulating an election. *Social Choice and Welfare*, 6(3):227–241, 1989.

S. Bellantoni and S. Cook. A new recursion-theoretic characterization of the polytime functions. *Computational complexity*, 2(2):97–110, 1992.

R. Bellman. Dynamic Programming Treatment of the Travelling Salesman Problem. *Journal of the ACM*, 9:61–63, 1962.

S. Ben-David and S. Halevi. On the independence of P versus NP. Technical Report 714, Technion, 1992.

P. Benacerraf and H. Putnam, editors. *Philosophy of Mathematics: Selected readings*. Prentice-Hall, Englewood Cliffs, NJ, second edition, 1983.

P. Bernays. Sur le platonisme dans les mathématiques. *L'enseignement mathematique*, 34:52– 69, 1935. Reprinted as "On plantonism in mathematics" (translated by C. Parsons) in Benacerraf and Putnam (1983), pp. 58-71.

F. Brandt, V. Conitzer, U. Endriss, J. Lang, and A. D. Procaccia. *Handbook of computational social choice*. Cambridge University Press, 2016.

S. Buss. *Bounded Arithmetic*. Bibliopolis, Naples, 1986.

S. R. Buss. On Gödel's theorems on lengths of proofs II: Lower bounds for recognizing k symbol provability. In P. Clote and J. Remmel, editors, *Feasible mathematics II*, pages 57–90. Springer, 1995.

J. Carlson, A. Jaffe, and A. Wiles, editors. *The millennium prize problems*. American Mathematical Society, Providence, R. I., 2006.

P. Cegielski. Le théoreme de dirichlet est finitiste. *Preprint LITP*, 92, 1992.

B. Chazelle and L. Monier. Unbounded hardware is equivalent to deterministic Turing machines. *Theoretical Computer Science*, 24(2):123–130, 1983. ISSN 0304-3975.

C. Cherniak. *Minimal Rationality*. MIT Press, 1986.

H. Chernoff. A note on an inequality involving the normal distribution. *The Annals of Probability*, 9(3):533–535, 1981.

A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):345–363, 1936.

P. Clote and J. Krajícek. *Arithmetic, proof theory, and computational complexity*. Oxford Univeristy Press, Oxford, 1993.

A. Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Proceedings of the Third International Congress for Logic, Methodology and Philosophy of Science, Amsterdam*, pages 24–30. North-Holland, 1965.

L. Colson. About primitive recursive algorithms. *Theoretical Computer Science*, 83(1):57–69, 1991. ISSN 0304-3975.

S. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on theory of computing*, pages 151–158. ACM, 1971.

S. Cook. A hierarchy for nondeterministic time complexity. *Journal of Computer and System Sciences*, 7(4):343 – 353, 1973.

S. Cook. Feasibly constructive proofs and the propositional calculus (preliminary version). In *Proceedings of seventh annual ACM symposium on Theory of computing*, pages 83–97. ACM, 1975.

S. Cook. The **P** versus **NP** problem. In J. Carlson, A. Jaffe, and A. Wiles, editors, *The millennium prize problems*, pages 87–106. American Mathematical Society, Providence, R.I., 2006.

D. Corfield. *Towards a Philosophy of Real Mathematics*. Cambridge University Press, Cambridge, 2004.

M. Cuffaro. Universality, invariance, and the foundations of computational complexity in the light of the quantum computer. In S. Hansson, editor, *Technology and Mathematics: Philosophical and Historical Investigations*, pages 253–282. Springer, 2018.

C. Daskalakis, P. Goldberg, and C. Papadimitriou. The complexity of computing a nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.

W. Dean. Computational complexity theory. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 2016 edition, 2016a. URL https://plato.stanford.edu/archives/win2016/entries/computational-complexity/.

W. Dean. Algorithms and the mathematical foundations of computer science. In P. Welch and L. Horsten, editors, *Gödel's Disjunction: The Scope and Limits of Mathematical Knowledge*, pages 19–66. Oxford University Press, 2016b.

W. Dean. Squeezing feasibility. In A. Beckmann, L. Bienvenu, and N. Jonoska, editors, *Pursuit of the Universal: 12th Conference on Computability in Europe, CiE 2016, Paris, France, June 27 - July 1, 2016, Proceedings*, pages 78–88, 2016c.

W. Dean. Strict finitism, feasibility, and the sorites. *The Review of Symbolic Logic*, 11(2):295–346, 2018. doi: 10.1017/S1755020318000163.

M. Detlefsen. On an alleged refutation of Hilbert's program using Gödel's first incompleteness theorem. *Journal of Philosophical Logic*, 19(4):343–377, 1990.

M. Detlefsen. Philosophy of Mathematics in the Twentieth Century. In S. G. Shanker, editor, *Philosophy of Science, Logic and Mathematics in the Twentieth Century*, volume 9 of *Routledge History of Philosophy*, pages 50–123. Routledge, London, 1996.

M. Detlefsen and A. Arana. Purity of methods. *Philosophers' imprint*, 11(2):1–20, 2011.

M. Dummett. Wang's Paradox. *Synthese*, 30(3/4):301–324, 1975.

M. Dummett. *Elements of Intuitionism*, volume 39. Oxford University Press, Oxford, 2000.

K. Easwaran. Probabilistic proofs and transferability. *Philosophia Mathematica*, 17:341–362, 2009.

J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17(3):449–467, 1965a.

J. Edmonds. Minimum partition of a matroid into independent subsets. *Journal of Research of the National Bureau of Standards-B. Mathematics and Mathematical Physics*, 69:67–72, 1965b.

W. Ewald. *From Kant to Hilbert: A Source Book in the Foundations of Mathematics*. Oxford University Press, New York, 1996.

W. Ewald and W. Sieg, editors. *David Hilbert's Lectures on the Foundations of Logic and Arithmetic 1917 – 1933*. Springer, Berlin, 2013.

D. Fallis. The Epistemic Status of Probabilistic Proof. *The Journal of Philosophy*, 94(4):165–186, 1997.

S. Feferman, H. Friedman, P. Maddy, and J. Steel. Does Mathematics need New Axioms? *The Bulletin of Symbolic Logic*, 6(4):401–446, 2000.

S. Feferman et al., editors. *Kurt Gödel Collected Works. Vol. I. Publications 1929-1936*. Oxford Univeristy Press, Oxford, 1986.

S. Feferman et al., editors. *Kurt Gödel Collected works. Vol. II. Publications 1938–1974*. Oxford Univeristy Press, 1990.

S. Feferman et al., editors. *Kurt Gödel Collected Works. Vol. V. Publications Correspondence H-Z.* Oxford Univeristy Press, 2003.

F. Ferreira. A feasible theory for analysis. *The Journal of Symbolic Logic*, 59(3):1001–1011, 1994.

L. Fortnow. The status of the **P** versus **NP** problem. *Communications of the ACM*, 52(9):78–86, 2009.

L. Fortnow. *Golden Ticket: **P**, **NP**, and the Search for the Impossible.* Princeton University Press, 2013.

M. Furst, J. B. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical systems theory*, 17(1):13–27, 1984.

H. Gaifman. Non-standard models in a broader perspective. In A. Enayat and R. Kossak, editors, *Non-standard models of arithmetic and set Theory*, pages 1–22. American Mathematical Society, Providence, 2004.

R. Gandy. Limitations to mathematical knowledge. In D. van Dalen, D. Lascar, and J. Smiley, editors, *Logic Colloquium 80*, volume 108, pages 129–146. North Holland, Amsterdam, 1982.

M. Garey and D. Johnson. *Computers and intractability. A guide to the theory of **NP**-completeness.* W.H. Freeman and Company, 1979.

K. Gödel. On formally undecidable propositions of *Principia Mathematica* and related systems I. Reprinted in Feferman et al. (1986), pp. 144-195, 1931.

K. Gödel. On undecidable propositions of formal mathematical systems. Princeton lectures, reprinted in Feferman et al. (1986), pp. 338-371, 1934.

K. Gödel. Some basic theorems on the foundations of mathematics and their implications. 1951. Reprinted in Feferman et al. (1990), pp. 290-323.

K. Gödel. Letter to von Neumann (1956). Reprinted in Feferman et al. (2003), pp. 373-377., 1956.

O. Goldreich. *P, NP, and NP-completeness: The basics of computational complexity.* Cambridge Univ Press, 2010.

A. Grzegorczyk. Some classes of recursive functions. *Rozprawy Matematyczne*, 4:3–45, 1953.

G. Gutin and A. Punnen. *The traveling salesman problem and its variations*, volume 12. Springer, 2002.

D. Harel. *Algorithmics: the spirit of computing.* Addison-Wesley, Reading, Massachusetts, 2006.

J. Hartmanis. Gödel, von Neumann and the **P** = **NP**? Problem. *Current Trends in Theoretical Computer Science*, pages 445–450, 1993.

J. Hartmanis and R. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117(5):285–306, 1965.

D. Hilbert. Neubegründung der Mathematik: Erste Mitteilung. *Abhandlungen aus dem Seminar der Hamburgischen Universität*, 1:157–77, 1922. English translation as "The new grounding of mathematics: First report" in Ewald (1996), pp. 1115–1134.

D. Hilbert and W. Ackermann. *Grundzüge der theoretischen Logik*. Springer, first edition, 1928. Reprinted in Ewald and Sieg (2013).

D. Hilbert and P. Bernays. *Grundlagen der Mathematik*, volume I. Springer, Berlin, 1934. Second edition 1968.

J. Hopcroft and J. Ulman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

R. Impagliazzo, V. Kabanets, and A. Kolokolova. An axiomatic approach to algebrization. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 695–704. ACM, 2009.

T. Jech. *Set Theory*. Springer Monographs in Mathematics. Springer-Verlag, Berlin, 2003.

R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of computer computations*, pages 85–103. Plenum Press, 1972.

S. Kleene. General recursive functions of natural numbers. *Mathematische Annalen*, 112(1):727–742, 1936.

S. Kleene. Recursive predicates and quantifiers. *Transactions of the American Mathematical Society*, 53(1):41–73, 1943.

S. Kleene. *Introduction to Metamathematics*. North-Holland, Amsterdam, 1952.

S. Kleene. Arithmetical predicates and function quantifiers. *Transactions of the American Mathematical Society*, 79(2):312–340, 1955.

G. Kreisel. Informal Rigour and Completeness Proofs. In *Problems in the Philosophy of Mathematics*, volume Lakatos, I., pages 138–186. North-Holland, Amsterdam, 1967a.

G. Kreisel. Review of "Le programme ultra-intuitionniste des fondements des mathématiques." (Volpin). *Zentrablatt Math*, 134:9–10, 1967b.

G. Kreisel. Church's Thesis: A kind of reducibility axiom for constructive mathematics. In A. Kino, J. Myhill, and R. Vesley, editors, *Intuitionism and Proof Theory*, pages 121–150. North-Holland, Amsterdam, 1970.

S. Kripke. *Philosophical troubles: Collected papers*, volume 1. Oxford Univeristy Press, Oxford, 2011.

R. Ladner. On the structure of polynomial time reducibility. *Journal of the ACM*, 22(1):155–171, 1975.

S. Lavine. *Understanding the Infinite*. Harvard University Press, Cambridge, 1994.

D. Leivant. A foundational delineation of Poly-Time. *Information and Computation*, 110(2):391–420, 1994.

J. Lenstra, W. Hendrik, and C. Pomerance. Primality testing with Gaussian periods. *Journal of the European Mathematical Society*, 2019.

L. Levin. Universal sorting problems (corrected translation published in Trakhtenbrot (1984)). *Problems of Information Transmission*, 9(3):265–266, 1973.

R. J. Matthews and E. Dresner. Measurement and computational skepticism. *Noûs*, 51(4):832–854, 2017.

C. Moore and S. Mertens. *The Nature of Computation*. Oxford University Press, 2011.

K. Mulmuley. The GCT program toward the **P** vs. **NP** problem. *Commun. ACM*, 55(6):98–107, 2012.

K. Mulmuley and M. Sohoni. Geometric complexity theory I: An approach to **P** vs. **NP** and related problems. *SIAM Journal on Computing*, 31(2):496–526, 2001.

J. Nash. Letter to the united states national security agency, 1950. URL https://www.nsa.gov/Portals/70/documents/news-features/declassified-documents/nash-letters/nash_letters1.pdf.

E. Nelson. *Predicative Arithmetic*, volume 32 of *Mathematical Notes*. Princeton University Press, Princeton, 1986.

M. Pantsar. Cognitive and computational complexity: Considerations from mathematical problem solving. *Erkenntnis*, pages 1–37, 2019.

C. Papadimitriou. *Computational complexity*. Addison-Wesley, New York, 1994.

R. Parikh. Existence and feasibility in arithmetic. *Journal of Symbolic Logic*, 36(3):494–508, 1971.

C. Parsons. Mathematical intuition. In *Proceedings of the Aristotelian Society*, volume 80, pages 145–168, 1979.

P. Pudlak. A bottom-up approach to foundations of mathematics. In P. Hájek, editor, *Gödel'96: Logical foundations of mathematics, computer science and physics—Kurt Gödel's legacy*, pages 81–97. Association for Symbolic Logic, 1996.

H. Putnam. *Representation and Reality*. MIT Press, Cambridge, 1988.

Z. Pylyshyn. *Computation and cognition: toward a foundation for cognitive science*. The MIT Press, Cambridge, Massachusetts, 1984. ISBN 0262160986.

M. Rabin. Probabilistic algorithm for testing primality. *Journal of number theory*, 12(1):128–138, 1980.

A. A. Razborov and S. Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55(1): 24–35, 1997.

K. W. Regan. Understanding the Mulmuley-Sohoni Approach to **P** vs. **NP**. *Bulletin of the EATCS*, 78:86–99, 2002.

M. Rescorla. The representational foundations of computation. *Philosophia Mathematica*, 23(3): 338–366, 2015.

R. W. Ritchie. Classes of predictably computable functions. *Transactions of the American Mathematical Society*, pages 139–173, 1963.

R. Rodošek. A new approach on solving 3-satisfiability. In *International Conference on Artificial Intelligence and Symbolic Mathematical Computing*, pages 197–212. Springer, 1996.

H. Rogers. *Theory of recursive functions and effective computability*. MIT Press, Cambridge, MA, 1987. First edition 1967.

H. Rose. *Subrecursion: functions and hierarchies*. Clarendon Press Oxford, 1984.

W. Savitch. Relationship between deterministic and non-determinsitic tape classes. *Jounral Computer System Science*, 4:177–192, 1970.

A. Schorr. Physical parallel devices are not much faster than sequential ones. *Information Processing Letters*, 17(2):103 – 106, 1983.

A. Schrijver. On the history of combinatorial optimization (till 1960). *Discrete optimization*, 12: 1–68, 2005.

J. R. Searle. Is the brain a digital computer? *Proceedings and Addresses of the American Philosophical Association*, 64(3):21–37, 1990.

N. Segerlind. The complexity of propositional proofs. *Bulletin of Symbolic Logic*, 13(04):417–481, 2007.

A. Shamir. **IP** = **PSPACE**. In *Proceedings 31st Annual Symposium on Foundations of Computer Science*, pages 11–15. IEEE, 1990.

S. Shapiro. Acceptable notation. *Notre Dame Journal of Formal Logic Notre-Dame, Ind.*, 23(1): 14–20, 1982.

S. Shapiro. Computing with numbers and other non-syntactic things: *De re* knowledge of abstract objects. *Philosophia Mathematica*, 25(2):268–281, 2017.

A. Shpilka, A. Yehudayoff, et al. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3–4):207–388, 2010.

W. Sieg. On computability. In A. Irvine, editor, *Philosophy of mathematics*, volume 4 of *Handbook of the philosophy of science*, pages 535–630. North Holland, Amsterdam, 2009.

M. Sipser. The history and status of the **P** versus **NP** question. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 603–618. ACM, 1992.

M. Steiner. Kripke on logicism, Wittgenstein, and *de re* beliefs about numbers. In A. Berger, editor, *Saul Kripke*, pages 160–176. Cambridge University Press, Cambridge, 2011.

W. Tait. Finitism. *Journal of Philosophy*, 78(9):524–546, 1981.

B. Trakhtenbrot. A survey of Russian approaches to *perebor* (brute-force searches) algorithms. *Annals of the History of Computing*, 6(4):384–400, 1984.

A. S. Troelstra. History of constructivism in the 20th century. In *Set Theory, Arithmetic, and Foundations of Mathematics: Theorems, Philosophies*, pages 150–179. Cambridge University Press, 2011.

A. Turing. On computable numbers, with an application to the *Entscheidungsproblem*. *Proceedings of the London mathematical society*, 42(2):230–265, 1936.

T. Tymoczko. The four-color problem and its philosophical significance. *Journal of Philosophy*, 76 (57-83), 1979.

A. Urquhart. Complexity, computational. In *The Routledge Encyclopedia of Philosophy*, volume 2, pages 471–476. Routledge, 1998.

A. Urquhart. Von Neumann, Gödel and complexity theory. *Bulletin of Symbolic Logic*, 16(4): 516–530, 2010.

L. G. Valiant. The complexity of computing the permanent. *Theoretical computer science*, 8(2): 189–201, 1979.

P. van Emde Boas. Machine models and simulations. In J. Van Leeuwen, editor, *Handbook of theoretical computer science (vol. A): algorithms and complexity*. MIT Press, Cambridge, MA, 1990.

P. Vitányi. Locality, communication, and interconnect length in multicomputers. *SIAM Journal on Computing*, 17:659, 1988.

P. Vopěnka. *Mathematics in the Alternative Set Theory*. Teubner, Leipzig, 1979.

H. Wang. Eighty years of foundational studies. *Dialectica*, 12(3-4):466–497, 1958.

H. Wang. *Computation, Logic, and Philosophy*. Kluwer, Dordrecht, 1990.

A. Yessenin-Volpin. Le programme ultra-intuitionniste des fondements des mathématiques. In *Infinitistic Methods, Proceedings of the Symposium on the Foundations of Mathematics*, pages 201–223. Pergamon Press, Oxford, 1961.

A. Yessenin-Volpin. The ultra-intuitionistic criticism and the antitraditional program for the foundations of mathematics. In A. Kino, J. Myhill, and R. Vesley, editors, *Intuitionism and Proof Theory*, pages 3–45. North Holland, Amsterdam, 1970.

A. Yessenin-Volpin. About infinity, finiteness and finitization (in connection with the foundations of mathematics). In *Constructive mathematics*, pages 274–313. Springer, Berlin, 1981.