Complexity Barriers as Independence

Antonina Kolokolova

February 18, 2016

After many years of effort, the main questions of complexity theory remain unresolved, even though the concepts involved are simple. Understanding the main idea behind the statement of the "P vs NP" problem does not require much background ("is it easier to check answers than to produce them?"). Yet, we are as far from resolving it as ever. Much work has been done to unravel the intricate structure in the complexity world, the "complexity zoo" contains hosts of inhabitants. But the main questions are still elusive.

So a natural question comes to mind: is there any intrinsic reason why this is still unknown? Is there any rationale why the proofs are out of our reach? Maybe we are not using the right techniques – or maybe we are not pushing our techniques far enough? After trying to prove a statement and failing we try to prove its negation; after failing at that, as well, we resort to looking for an explanation that might give us a hint at why our attempts are failing. Indeed, in the world of computational complexity there have been several results of this nature: results that state that current techniques are, in a precise mathematical sense, insufficient to resolve the main open problems. We call these results "barriers".

A pessimistic view of the barrier results would be that the questions are hard, intrinsically hard. But there is a more optimistic way of interpreting them. The fact that certain classes of proof techniques, ones that have specific properties, are eliminated gives us a direction to search for new techniques. It gives us a method for discovering ways of approaching questions in places where we might not have been looking, if not for the barrier results.

In this paper, we will focus on three major complexity barriers: Relativization [BGS75], Algebrization [AW09], and Natural Proofs [RR97]. Interestingly enough, all three of those can be recast in the framework of independence of a theory of logic. That is, theories can be constructed which formalize (almost) all known techniques, yet for which the main open questions of complexity theory are independent.

1 Introduction

Complexity theory evolved from computability theory by restricting the notions of computable to computable *efficiently*. The main objects of complexity the-

ory can be viewed as "scaled down" versions of basic objects of computability theory; here, "scaling down" amounts to limiting computational resources, such as bounding quantifiers to make quantified objects "small" and requiring all computation to be "efficient".

There are problems which are known to be decidable (that is, there is an algorithm producing a definite answer for every input), but for which the time to compute the answers is immense, exceeding the number of atoms in the universe even for relatively small inputs. Here we assume the customary way of measuring the computational complexity of a problem as a function of the input size. To make the notion of "computable" more realistic, it is natural to put a bound on the computational resources allotted to compute the answer: time for how long the computation is allowed to run (treated here as the length of the computation, that is, the number of steps of the algorithm), or amount of memory the algorithm is allowed to use. We bound the running time (memory) on all possible inputs of a given length, which is the worst-case complexity measure.

Intuitively, checking whether an input string is a palindrome is a simpler problem than deciding whether the input is a true statement of Presburger arithmetic, though the latter also has an algorithm producing an answer for every input. And in general, a scaled-down version of a problem of finding a solution out of infinitely many possibilities (i.e., finding an accepting computation of a Turing machine) can be formulated as finding a small solution (comparable to the size of the input) significantly faster than by brute-force search over all small answers.

1.1 The early history of efficient time-bounded computation

What would be a reasonable bound on the length of computation which would make it "efficient"? In his 1956 letter to von Neumann[1], Gödel asks whether there is an algorithm that, given an n-bit long encoding of a formula, can find a a short proof in $k \cdot n$ or $k \cdot n^2$ steps, where k is a number (providing the $k \cdot n$ as a lower bound on such computation); thus, here, "efficient" is kn or kn^2. There is a problem though with defining "efficient" as linear (or quadratic) in the length of the input: it is too model-dependent. For example, any single-tape Turing machine needs roughly n^2 time to decide if an n-bit string is a palindrome, whereas there is a two-tape Turing machine which accomplishes this task in about $2n$ time.

A few more papers in the early 1960s deal with the notions of efficient computation and computational difficulty of problems. Hartmanis and Stearns [HS64, HS65] introduced time and space complexity classes and proved, using computability-theoretic techniques, that giving a Turing machine more time or space allows it to solve more problems (when time and space bounds are

[1]This letter, though it outlines many of the central concepts of complexity theory, was only rediscovered in 1988, well after these concepts were defined independently

computable functions). Edmonds [Edm65] discusses the notion of efficient algorithms in the paper where he provides a better than brute-force search algorithm for finding a maximum matching in a general graph. And in his talk at the International Congress on Logic Methodology and Philosophy of Science in 1964, Cobham [Cob64] formulates the notion of efficient computation that became the standard in complexity theory: the number of steps of a computation should be bounded by some fixed polynomial function of the length of an input. There, Cobham also presents a recursion-theoretic characterization of polynomial time which we will discuss at greater length later in this paper.[2]

As the class of recursive (decidable) languages scales down to the class of polynomial-time computable languages (which we will denote P), the recursively enumerable (semi-decidable) languages become NP, non-deterministic polynomial-time computable languages. That is, this is a class of languages such that for every string in the language there is a certificate (of size at most polynomial in the length of the input), which can be verified by a (polynomial-time) computable predicate. Rather than asking for an existence of a finite computation that can be computably verified, we ask for an existence of a polynomial-length witness which can be checked in time polynomial in the length n of the input string. Note that there are only exponentially many strings of length polynomial in n, so finding such a witness is always computable in exponential time (EXP): just try all possible small witnesses and see if one of them works.

Is trying all possible witnesses necessary, or is it always possible to do significantly better? This question is sometimes referred to as the problem of avoiding exhaustive search, also known as a "perebor problem" (проблема перебора) in Russian-language literature. This question was formalized as the "P vs. NP" question in the seminal papers of Stephen Cook [Coo71] in which the notion of NP-completeness was first presented; independently, very similar notions have been formulated by Leonid Levin [Lev73]. Unlikely as it seems, this question is wide open: though we know from the Hartmanis-Stearns paper [HS65] that $\mathsf{P} \subsetneq \mathsf{EXP}$, that is there are languages computable in exponential time, but not in polynomial time, and that $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{EXP}$, we are still not able to rule out either of these two inclusions being equalities.

1.2 Bounding other resources

There are many more beasts of the complexity zoo roaming in this range between P and EXP. Probably the most notable one is PSPACE, the class of languages computable by algorithms using only the amount of memory polynomial in the size of the input. The canonical complete problem for this class is True Quantified Boolean Formulae (TQBF) with arbitrary nesting depth of quantifiers; a number of other PSPACE-complete problems ask if there is a winning strategy

[2] This paragraph is by no means intended to give a full history of development of these concepts. There are numerous books and surveys that address this question much better. My main goal is to give a quick introduction to how complexity classes came from the corresponding classes in computability theory, and motivate why we focus on polynomial-time computation.

in a two-player game: there, the alternating quantifiers code alternating moves by the two players.

Naturally, $\mathsf{P} \subseteq \mathsf{PSPACE}$, since even visiting a different memory location at every time step there are only polynomially many locations that can be touched. Similarly, $\mathsf{NP} \subseteq \mathsf{PSPACE}$: just notice that the algorithm that checks all possible witnesses can be implemented with only the polynomial amount of memory: each check takes polynomial time (and thus polynomial space), and the counter to keep track of the witnesses is bounded by the length of the largest witness. Also, $\mathsf{PSPACE} \subseteq \mathsf{EXP}$, since for a given language L and its input, there are exponentially many possible configurations (say, of the tape of a Turing machine deciding this language with polynomial amount of space). Then the question of whether this Turing machine accepts becomes a reachability question: is there a path from the start configuration to some accepting configuration in the configuration graph of this computation, where there is an edge from one configuration to another if the latter can be obtained from the former by a single transition. Since reachability for polynomial-size graphs can be tested in polynomial time using numerous algorithms starting from depth first search/breadth first search, reachability for exponential-size graphs such as this configuration graph can be decided in exponential time. Note that his also works for non-deterministic setting, when there are multiple possible transitions out of a given configuration, so even for non-deterministic polynomial-space computable languages can be computed in EXP. It turns out that for polynomial space non-determinism does not help: a classic result of Savich [Sav70] shows that non-deterministic polynomial-space computation can be simulated by a deterministic computation with only polynomial overhead. Thus, $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} = \mathsf{NPSPACE} \subseteq \mathsf{EXP}$, where only the first and last class are known to be distinct.

2 The first barrier: Relativization

Hartmanis and Stearns [HS64, HS65] were the first to define a notion of a complexity class (as a class of recursive sequences computable by a multi-tape Turing machine within a specified time bound given as a function of the input size). With this definition, they show that such complexity classes form a strict hierarchy: given more time, Turing machines can compute more complex languages. The main tools they use are the familiar methods from computability theory: simulation and diagonalization. Essentially, they show how, given a computable function $T(n)$, to construct a language computable in time $(T(n))^2$ which differs from any language computable in time $T(n)$. The quadratic time bound was later improved to $T(n) \log T(n)$ by Hennie and Stearns [HS66]; subsequently, hierarchy theorems were proven for non-deterministic computation [Coo72] and space complexity [SHLI65].

The proofs of these hierarchy theorems, as is usually the case with proofs based on diagonalization, have a property which makes the results stronger: they are insensitive to the presence of oracles. More precisely, an "oracle", introduced by Turing [Tur39], is just a language and an oracle Turing machine for

a given oracle can query whether a given string is in the language in constant (unit) time. This immediately allows oracle Turing machines to compute incomputable languages by taking the language in question as an oracle (e.g., oracle Turing machines can decide Halting problem by a single query to the language of all positive instances of the Halting problem). Thus, any language can be decided by a Turing machine with an oracle to this language, trivially. However, there is no language powerful enough so that with this language as an oracle Turing machines can decide everything. And the proof of this is the same as the standard diagonalization proof of existence of undecidable languages: the list of Turing machines all with an access to the same oracle is countable. Therefore, replacing the list of all Turing machines with the list of all Turing machines with an oracle A does not change the proof. We will call proofs which have this property, insensitivity to the presence of oracles, *relativizable*.

Hartmanis-Stearns proof of the hierarchy theorem uses just the diagonalization and so is relativizable. Given an oracle, there is a hierarchy of time and space complexity classes with respect to this oracle. However, when types of resources are mixed, the situation gets more complicated, leading to the first barrier that we will consider, the *relativization barrier*.

2.1 Relativization: The Baker, Gill, and Solovay theorem

If there is a purely diagonalization-based proof of $\mathsf{P} \neq \mathsf{NP}$, then this proof should be insensitive to the presence of oracles. So, P with an oracle A (denoted P^A) should be not equal to NP^A for any oracle A. However, Baker, Gill and Solovay in their 1975 paper presented oracles A and B such that with respect to A, $\mathsf{P} = \mathsf{NP}$, and with respect to B, $\mathsf{P} \neq \mathsf{NP}$.[3] Thus, in particular, neither proof of $\mathsf{P} = \mathsf{NP}$ or of $\mathsf{P} \neq \mathsf{NP}$ can be done using relativizing techniques such as diagonalization.

The first oracle, A, for which $\mathsf{P}^A = \mathsf{NP}^A$ is chosen to be a language powerful enough that it can be used to solve problems in both P and NP, and in a class with good closure properties. In particular, A can be any language that is complete for PSPACE. It is possible to decide languages in NP^A in $\mathsf{NPSPACE}$ by simulating all polynomially-many queries in PSPACE for each non-deterministic branch of computation. Note that, since $\mathsf{PSPACE} = \mathsf{NPSPACE}$ by Savitch's theorem, NP^A is already in PSPACE. Finally, $\mathsf{PSPACE} \subseteq \mathsf{P}^A$, as every language in PSPACE can be decided by a reduction to A (in polynomial time) by definition of its completeness. Therefore, $\mathsf{P}^A = \mathsf{NP}^A$.

The oracle B for which P and NP are separate is constructed in [BGS75] using diagonalization ideas. Consider, for a given set B, a language $L(B)$ which contains all strings of a given length if there is a string of that length in B, and no strings of that length otherwise (that is, $L(B) = \{x | \exists y \in B \ (|x| = |y|)\}$. This language is in NP^B, because it is possible, given x, to check if any y of

[3] According to Baker, Gill and Solovay, an oracle for which $\mathsf{P} = \mathsf{NP}$ was independently constructed by Meyer and Fischer, and also by H.B Hunt III, however they did not use the oracle described here. For $\mathsf{P} \neq \mathsf{NP}$, they attribute some constructions to Ladner, and some again to Meyer/Fischer.

the same length is in B, checking one such y on any computation path. The interesting part is to construct B such that $L(B) \notin \mathsf{P}^B$.

The set of strings which could potentially be in the oracle is divided into blocks by size, where each block is designed to fool the i^{th} polynomial-time Turing machine M_i. Consider the behaviour of M_i on the string 0^n, for n exponentially increasing at each step and such that 2^n is larger than the limit on M_i's running time. If M_i accepts 0^n, then i^{th} block will contain no strings (so for all x such that $|x| = n$, $x \notin L(B)$, yet M_i accepts 0^n) . Otherwise, it will contain one string, which is the smallest string of length n not queried by M_i on 0^n; it exists because 2^n is larger than M_i's running time. Since M_i does not query this string, its presence or absence in B will not affect M_i not accepting 0^n. But if M_i were deciding $L(B)$, it would have to accept all strings of length n, including 0^n. Finally, because each n is chosen to be exponentially larger than the n from the previous stage, but all M_j run in polynomial time, no previous M_j can query strings of length n, so adding strings of length n does not change their behavior.

In the same paper, Baker, Gill and Solovay construct a number of oracles corresponding to various scenarios of relationships between these classes. Is NP closed under complementation, when it is not P? For both there is an oracle with respect to which this scenario is true. If not, then is P the subset of NP closed under intersection? Again, for either of them an oracle world making it true can be constructed. A number of results followed showing that for a vast majority of complexity questions there are contrary relativizations: existence of one-way functions, power of probabilistic algorithms (BPP vs. P), interactive games (IP vs. PSPACE), etc.

This surprising result can be immediately recast to show, for example, that with respect to some oracle C, P^C vs NP^C is independent of the axioms of ZFC (or other axiomatizable consistent formal theories). More specifically, Hartmanis and Hopcroft [HH76] construct a recursive set C using the [BGS75] oracles A and B as follows. Let C be the language $\mathcal{L}(M)$ of a Turing machine M which accepts x if either there exists a proof of $\mathsf{P}^{\mathcal{L}(M)} = \mathsf{NP}^{\mathcal{L}(M)}$ among the first x proofs in the theory and $x \in B$, or a proof of $\mathsf{P}^{\mathcal{L}(M)} \neq \mathsf{NP}^{\mathcal{L}(M)}$ among the first x proofs in the theory and $x \in A$. The existence of such M comes from the recursion theorem. Let $C = \mathcal{L}(M)$. But now C is essentially A, except for the finite part, if there is a proof of $\mathsf{P}^C \neq \mathsf{NP}^C$, and essentially B if there is a proof of $\mathsf{P}^C = \mathsf{NP}^C$, contradicting the [BGS75] theorem that the opposite is true for oracles A and B.

2.2 Polynomial time as a black box: The recursion-theoretic definition

Hartmanis and Hopcroft use results from [BGS75] to define a complexity-theoretic problem independent of Zermelo-Fraenkel set theory. A different logic question that the relativization barrier inspires is whether it itself can be restated as an independence result. It does have a similar flavour: no technique with a certain

property can be used to resolve questions. Thus, intuitively, a theory formalizing the reasoning having this property would only prove relativizing results, and all non-relativizing results would be independent from it. This intuition has been made precise in the unpublished manuscript[4] by Arora, Impagliazzo and Vazirani [AIV92].

But which property would such a theory formalize? One way to summarize the techniques that give relativizing results is to say that they treat computation as a black box. Such techniques rely on the closure properties of the corresponding classes of functions, and on properties such as the existence of a universal function for a class (useful for simulation). However, they do not consider the inner workings of a model of computation such as a Turing machine: in most such results, a Turing machine can be readily replaced with a Random Access Machine or a recursion-theoretic characterization of the corresponding function class. Consider, for example, the difference between lambda calculus and the Turing machine models. In the former framework, the only information about the functions is their recursive definition. So they are, computationally, black boxes. There is no extra information. But in the Turing machine computation, there is a lot of additional information: for example, subsequent steps of computation only affect a small number of adjacent cells, etc.

So, a theory formalizing only relativizing techniques can be based on reasoning that only works with some generic recursion-theoretic definition of a class of functions in a complexity class. And in [AIV92] Arora, Impagliazzo and Vazirani explored that idea building upon Cobham's [Cob64] definition of polynomial time computable functions.

Definition 2.1. Let FP' be a class of functions satisfying the following properties:

1. FP' contains basic functions:[5] constants, addition, subtraction, length $|x|$, $BIT(x, j)$, projections, multiplication and the "smash" function $2^{|x|^2}$.
2. FP' is closed under function composition $f \circ g$ and pairing $\langle f(x), g(x)\rangle$ of functions.
3. FP' is closed under the limited recursion on notation: for functions $g, h_0, h_1 \in FP'$ and constants c, d

$$f(x, 0) = g(x) \tag{1}$$

$$f(x, 2k) = h_0(x, k, f(x, k)) \qquad f(x, 2k+1) = h_1(x, k, f(x, k)) \tag{2}$$

$$|f(x, k)| \leq c|x|^d \tag{3}$$

[4] This paper has never been published, although it is mentioned in a number of published works. The standard graduate complexity theory textbook by Arora and Barak [AB09] devotes a page to it, Fortnow discusses it in his 1994 [For94] paper. Most publications on the algebrization barrier, including [AW09], reference [AIV92] and the follow-up paper [IKK09] relies heavily on the results of [AIV92] to formalize algebrization; it is mentioned in a number of other publications by Aaronson. Sometimes, this manuscript is dated 1993, and/or cited with a title "On the role of the Cook-Levin theorem in complexity theory".

[5] In fact, successors s_0, s_1 and smash are sufficient.

Here, $c|x|^d$ provides a polynomial bound on the length of the output of $f(x,k)$, to avoid e.g. using repeated squaring to define exponentiation. In this case, we use a true polynomial bound rather than a function from FP'.

Cobham's celebrated result states that the class FP of polynomial-time computable functions is the minimal class FP' satisfying the definition 2.1. But what can be said about classes FP' satisfying these axioms which are not minimal? Such classes might contain, in additional to polynomial time functions, spurious functions of arbitrary complexity and their closure under polynomial-time operations. This already becomes reminiscent of the idea of an oracle: indeed, adding a characteristic function of an oracle to FP and closing under the operations (composition, pairing, limited recursion on notation) gives an FP' satisfying the definition.

Suppose now that FP' is a class satisfying the definition. Can it be viewed as FP^O for some oracle O? First of all, if the spurious functions produce huge (non-polynomial length) output, then they would not fit into the oracle framework: an oracle polynomial time Turing machine gets one bit of information from each query to the oracle, and has only a polynomial in input length amount of time to write its answer on the tape. But what if every function in FP' produces an output of polynomial length, in the same manner as the output of the function in the limited recursion on notation is polynomially bounded? Then, an oracle can be constructed with the use of one more device: a universal function for FP'. This universal function $U(i,t,x)$ is defined to compute $f_i(x)$ within the time bound t, for every $f_i \in FP'$. Note that without the time bound there is no universal function for FP that would be in FP, even for the actual class of polynomial time computable functions: there is no specific polynomial that can bound the running time of every polynomial-time function by the time hierarchy theorem. However, for every polynomial-time function there is a bound $2^{|x|^c}$ for some constant c depending on the function which would allow the universal function to be computed within the time polynomial in its parameters. Now, provided $U(i,t,x)$ is in FP' and is a universal function for FP', an oracle O that on a query (i,t,x,j) outputs the j^{th} bit of $U(i,t,x)$ is sufficient to compute all functions in FP' in FP^O.

2.3 Relativization as independence: The Arora, Impagliazzo and Vazirani [AIV92] framework

Let T be a theory (for example, the Peano Arithmetic). Rather than reasoning about the actual polynomial-time functions in T, we would like to reason about functions in FP' given by the recursive definition above: this will be essentially the only information about these functions given to the theory. The question now becomes: what kind of results can be proven in T given only this limited information about FP'? It does not matter too much what is the reasoning power of T itself: we can take T to be as powerful as needed to formalize combinatorial arguments a proof might require. The main restriction is the black-box view of FP' functions.

More precisely, define RCT (for "relativized complexity theory") to be T

(Peano Arithmetic) augmented with function symbols $f_1, f_2, \ldots$ satisfying the axioms from definition 2.1, together with two more axioms discussed above, one bounding the length of FP' functions and another giving the existence of a universal function for FP'.

$$\forall f \in FP' \exists c, d \forall x \quad |f(x)| \leq c|x|^d \qquad \text{(Axiom Length)}$$

$$\exists U \in FP' \forall f \in FP' \exists i, c \forall x \quad f(x) = U(i, 2^{|x|^c}, x) \qquad \text{(Axiom } \mathcal{U}\text{)}$$

The resulting theory can be viewed as a two-sorted theory, with a number sort and a function sort. Functions here are defined over integers; multiple arguments can be represented using pairing. The notation $f(x)$ is a shortcut for $Apply(f, x)$; the latter notation allows for the theory to remain first-order with the introduction of the function symbols.

Recall that a standard model of a theory of arithmetic interprets number variables as natural numbers, and function and relation symbols as functions and relations over natural numbers, with symbols such as $+$, $\times$, 0 getting their usual meaning. A standard model of RCT can be fully described by interpretations of all symbols in T together with interpretations of all $f_i \in FP'$. Thus, interpreting Peano Arithmetic symbols in the usual manner, a standard model of RCT can be viewed as a set of interpretations of functions in FP'. This, together with the discussion above, gives the following correspondence: any standard model with FP' a set of additional functions satisfies RCT if an only if there is a set $O \subset \mathbb{N}$ such that $FP' = FP^O$.

For the proof, it is enough to consider the same arguments as above for encoding "spurious functions" as an oracle by $O(i, t, x, j) = j^{th}$ bit of $U(i, t, x)$ for one direction, and adding a characteristic function of O as a "spurious function" to FP to obtain FP' as the closure for the other.

In this framework, the results from [BGS75] become statements about independence: there exist two models of RCT, corresponding to FP^A and FP^B, which give contrary relativizations of the P vs NP question. On the other hand, by assumption about the power of the underlying theory, proofs of theorems such as Hartmanis/Stearns hierarchy theorem [HS65] are formalizable in RCT.

3 Non-relativizing results and the next barrier

The result of Baker, Gill, and Solovay [BGS75] shows that we cannot rely on the simulation and diagonalization techniques alone to resolve major complexity questions. But what other options are there? What other techniques can be used that avoid this barrier? Intuitively, these have to be techniques that look at the computation more closely than in the black-box way. Techniques that analyse intrinsic properties of computation of specific computational models. Thus, the choice of the computational model will matter for approaching these problems.

Already Stockmeyer [Sto74] talked about the significance of a representation of a machine model for complexity theoretic results. The question of convenience

of various representations for complexity results, even as a difference between different ways to write a configuration of a Turing machine, has been studied on its own, in particular in [vEB12].

After [BGS75] appeared, a targeted search for non-relativizing results ensued. But even before, there were results in complexity theory that avoided the relativization barrier, the prime example being the Cook-Levin theorem. A number of important non-relativizing results came in the early 1990s, mainly in the setting of computation recast as "proof checking", including the $\mathsf{IP} = \mathsf{PSPACE}$ proof of [LFKN92, Sha92] (especially surprising since even with respect to a random oracle, $\mathsf{IP} \neq \mathsf{PSPACE}$ with probability 1 [CCG+94]) and the PCP theorem [AS98, ALM+98].

3.1 Interactive proof systems

Both the $\mathsf{IP} = \mathsf{PSPACE}$ and the PCP theorem view, though in a somewhat different manner, solving a decision problem as a dialogue between a resource-bounded verifier and a powerful prover. In that setting, NP can be viewed as a class of problems where it is possible to verify in polynomial time that a given proof (i.e., a solution) is correct; for an input not in the language, no alleged proof will pass the scrutiny of the verifier. Interactive proofs generalize this in two ways. First, they allow for multiple rounds of interaction, with verifier asking the prover for more and more answers to its questions. Alone, this generalization does not give extra power: the resulting class of problems is still NP, as the whole protocol of the interaction, being of polynomial length, can serve as a proof. More importantly, the verifier is allowed to use randomness, but then it may make mistakes. However, as long as the probability of the verifier making a mistake is less than $1/3$, we consider the interaction to be successful.

Definition 3.1. A language L is in the class IP if there exists a verifier such that, on every input x,

- if $x \in L$, then there is a prover (interaction strategy) that can convince the verifier with probability $> 2/3$ (over verifier's randomness)
- if $x \notin L$, no prover can convince the verifier that it is in the language with probability $> 1/3$.

Here, the constant $1/3$ is not essential: any $1/2 - \epsilon$ would do, for ϵ within an inverse polynomial of the length of the input; the reason being that repeating the protocol with a different choice of randomness decreases the error probability.

A classic example of a problem with a natural interactive proof protocol is graph (non)isomorphism: checking whether two graphs are the same up to permutation of vertices. In graph non-isomorphism problem, the input consists of encodings of two graphs G_1, G_2, and the algorithm should accept if G_1 and G_2 are not isomorphic. This problem is not known to be in P, but neither it is

believed to be NP-complete[6]. To see that its complement, graph isomorphism, is in NP, note that giving a permutation of vertices of G_1 that makes it G_2 is sufficient for the certificate, with the verifier checking that all the edges now match. Now, suppose the verifier randomly picks one of G_1, G_2, permutes its vertices to obtain a new graph G_3, and then sends G_3 to the prover, asking to identify whether G_3 is a permuted version of G_1 or of G_2. The protocol, due to [GMW91], is described below; there, r denotes verifier's random bits.

$Verifier(r, G_1, G_2)$		$Prover(G_1, G_2)$
Pick $i_1 \in \{1, 2\}$ randomly		
Permute G_{i_1} to get G_3		
	$\xrightarrow{G_3}$	
		find j_1 such that $G_{j_1} \cong G_3$
	$\xleftarrow{j_1}$	
if $j_1 \neq i_1$ reject		
else pick $i_2 \in \{1, 2\}$		
compute $G_4 \cong G_{i_2}$		
	$\xrightarrow{G_4}$	
		find j_2 such that $G_{j_2} \cong G_4$
	$\xleftarrow{j_2}$	
if $j_2 \neq i_2$ reject		
else accept		

If the two graphs are isomorphic, then G_3 could have come from either of them, so there is $1/2$ chance that the prover answers correctly. However if G_1 and G_2 were different, then, being computationally powerful, the prover can say with certainty which of G_1, G_2 the verifier picked. Thus, if the graphs are non-isomorphic, the prover answers correctly with probability 1, and if they are isomorphic, with probability $1/2$. Now, as $1/2$ is not good enough for our definition of IP, suppose the verifier repeats this procedure by picking a graph and permuting the vertices twice with independent randomness. This creates two graphs G_3 and G_4, and the verifier then asks the prover for the origin of each. If G_1, G_2 are isomorphic, then the probability that the prover is correct in both cases is only $1/4$. Note that here it is not necessary to use multiple rounds of interaction: even though the verifier can ask about G_3, receive the answer, and then ask about G_4, it is just as easy to send G_3 and G_4 to the prover simultaneously.

[6]The recent breakthrough result by Babai[Bab16] puts Graph Isomorphism in quasi-polynomial time (i.e., time $n^{\mathsf{poly}\log(n)}$), which is tantalizingly close to polynomial time.

Interactive proofs can be simulated in PSPACE: since each interaction protocol is of polynomial length, it is possible to go over all such protocols for a given input x and a given verifier, and find the answers given by the best prover. With those, it is possible to count in PSPACE the fraction of random strings for which the verifier accepts or rejects.

3.2 Power of IP in oracle worlds

The main non-relativizing result that we will discuss is $\mathsf{IP} = \mathsf{PSPACE}$. However, there are oracles (and in fact, nearly all oracles are such) for which the inclusion of IP in PSPACE is strict. The construction of the first such oracle goes back to Fortnow and Sipser [FS88]; they present an oracle C with respect to which there is a language in coNP (the class of languages with complements in NP). but not in IP. Guided by this result, they conjecture that IP does not contain coNP, and that proving such a result would require non-relativizing techniques as with respect to the oracle A for which $\mathsf{P}^A = \mathsf{NP}^A$, IP contains coNP since it contains P. The $\mathsf{IP} = \mathsf{PSPACE}$ result shows that this intuition was misleading, as PSPACE is believed to be so much more powerful than coNP, containing all of polynomial-time hierarchy and more.

The idea behind Fortnow/Sipser's oracle construction is similar to the construction of the oracle B from [BGS75], for which $\mathsf{P}^B \neq \mathsf{NP}^B$. As for B, the oracle C is constructed using diagonalization ideas. A language $L(C)$ will contain all strings 1^n for which all strings of length n are in C, that is, $L(C) = \{1^n \mid \forall x, |x| = n, x \in C\}$. It is easy to see that this language is in coNP^C, as it is enough to ask the oracle for all strings of length n, and accept if all answers are "yes". Now, as for the construction of B from [BGS75], the oracle C is constructed in blocks, where the i^{th} block is constructed to fool the i^{th} potential verifier V_i. Assume that the running time of V_i is within n^i for all i; then V_i cannot ask queries longer than n^i. Let N_i be the length on which V_i will be forced to accept or reject incorrectly with probability higher than $2/3$ (there, N_i should be large enough so that $N_{i-1}^{i-1} < N_i$, and $2^{N_i} > 3N_i^i$).

First, put into C all strings that V_i asks the oracle about. If there are no provers that make V_i accept 1^{N_i} with probability at least $2/3$, then put all the remaining strings of length N_i into C, in which case $1^{N_i} \in L(C)$ and V_i cannot be convinced of that. As well, put into C all strings V_i would ever ask about for any other provers, other than ones considered in previous blocks.

Now, suppose that there is a prover P that will make V_i accept 1^{N_i} with probability at least $2/3$. One cannot guarantee that there is a string of length N_i that has not been queried by V_i for any choice of V_i's random bits; however, one can argue that a string x that is queried the least appears in less than $1/3$ possible computation paths, as N_i was chosen to be such that $2^{N_i} > 3N_i^i$. Remove x from C; this will be the only string queried by V_i over all possible interactions with P that is not in C. Now, even if querying this string makes V_i reject, it will affect less than $1/3$ of possible computation paths of V_i on 1^{N_i}. So V_i will still accept 1^{N_i} with probability greater than $1/3$, completing the proof.

Even though there is an oracle with respect to which IP is quite weak, one

could say that this is an outlier, and maybe in general the power of IP with respect to an arbitrarily picked oracle will be more in sync with its power in the real world. This intuition, that behaviour with respect to a random oracle was indicative of the situation in the real world, became known as the random oracle hypothesis, following the result from Bennett and Gill [BG81] that $\mathsf{P}^A \neq \mathsf{NP}^A$ with probability 1 over oracles A. There, a random oracle is constructed by putting each string x in the oracle with probability 1/2. However, a 1994 paper by Chang, Chor, Goldreich, Hartmanis, Håstad, Ranjan and Rohatgi, titled "Random oracle hypothesis is false" [CCG+94], shows that with respect to a randomly chosen oracle A, $\mathsf{coNP}^A \subsetneq \mathsf{IP}^A$, and so the behavior of IP^C for the oracle C of [FS88] is typical rather than exceptional.

3.3 Arithmetization

Many of these new non-relativizing results rely on a technique called *arithmetization*. In arithmetization, a Boolean formula is interpreted as a polynomial, which for 0/1 values of variables (corresponding to false/true in the Boolean case) evaluates to 0 or 1. However, one can evaluate this polynomial on any integers or elements of a given finite field, and obtain a host of values giving more information about the structure of the formula. More precisely, to convert a Boolean formula such as $(\bar{x} \vee y) \wedge z$ to a polynomial, the conjunction $\wedge$ is interpreted as multiplication, and negation $\bar{x}$ as $(1 - x)$. Then, $(x \vee y)$ is converted, as $\overline{(\bar{x} \wedge \bar{y})}$, to $1 - (1 - x)(1 - y) = x + y - x \cdot y$. So the formula $(\bar{x} \vee y) \wedge z$ becomes a three-variate polynomial $(1 - x \cdot (1 - y)) \cdot z$. It can be checked that a formula resulting from this transformation indeed evaluates to 0/1 values when inputs are 0s and 1s, and it evaluates to 1 exactly when the original formula is true.

Arithmetization is a powerful algorithmic technique: for example, if the resulting polynomial is multilinear (that is, no variable occurs with a degree more than 1), then counting the fraction of assignments of this formula that are satisfying can be done by evaluating this polynomial with every variable set to 1/2 [JKRS09]. In converting a formula to a polynomial, though, both $\wedge$ and $\vee$ translations involve multiplication; thus, resulting polynomials, though of polynomial degree and described by polynomial-size circuits, are quite unlikely to be multilinear. One can always create a multilinear polynomial of a formula by taking a sum over terms corresponding to each satisfying assignment or taking a Fourier representation, but such a sum is likely to have more than a polynomial number of coefficients.

3.4 IP protocol for counting satisfying assignments

In general, counting the number of satisfying assignments is believed to be a harder problem than determining whether a formula is satisfiable. In fact, this problem (and, equivalently, computing the Permanent of a Boolean matrix) is at least as hard as any other problem in the polynomial time hierarchy. The latter is the celebrated theorem by Toda [Tod91]; see[For09] for a short proof. The

problem of counting the number of satisfying assignments, sometimes referred to as $\#SAT$, is in PSPACE and the technique for proving that it can be done using interactive proofs is very similar, except for a few details, to proving IP = PSPACE. Historically, this was one of the results leading the proof that IP = PSPACE; a quite entertaining account of the history of this result is presented in Babai's "E-mail and the unexpected power of interaction" [Bab90].

Theorem 3.2 ([LFKN92])**.** $\#SAT \in \mathsf{IP}$.

Proof. To simplify the notation, let us use a decision version $\#SAT_D$ of $\#SAT$, where, given a formula ϕ and a number K, the goal is to check whether K is the number of satisfying assignments of ϕ. To show that $\#SAT_D$ is in the class IP of problems that have interactive proofs we will explicitly exhibit an interactive proof protocol for it. That is, we will describe which protocol the verifier should follow to check if an alleged proof presented by the prover is correct; if presented with a correct proof in the format it expects, the verifier will accept with probability 1, and for any incorrect proof, it will reject with probability significantly higher than 2/3.

The number of satisfying assignments of a formula $\phi(x_1, \ldots, x_n)$ is equal to the sum, over all possible assignments, of the polynomial $p(x_1, \ldots, x_n)$ obtained by arithmetizing ϕ. Thus computing the number of satisfying assignments amounts to computing

$$\sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} p(x_1, \ldots, x_n).$$

For example, if the original formula is $\phi = (x_1 \vee \neg x_2)$, then the resulting polynomial is $1 - (1 - x_1)x_2 = 1 - x_2 + x_1 x_2$, and the number of satisfying assignments of ϕ is

$$\sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} 1 - x_2 + x_1 x_2 = 3.$$

The first key idea of the protocol is checking the value of the sum one variable at a time: setting

$$p_1(x_1) = \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} p(x_1, \ldots, x_n),$$

if p_1 is correct then it is sufficient to compute $p_1(0) + p_1(1)$ to obtain the answer. The second idea is recursing on evaluating the sum with a random number inserted in place of x_1. More specifically, the protocol proceeds as follows.

1. *Verifier* asks for the coefficients of the univariate polynomial

$$p_1(x_1) = \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} p(x_1, \ldots, x_n).$$

2. *Prover* sends the (alleged) coefficients of $c_{1,0}, \ldots c_{1,m}$ of p_1 to the Verifier.

3. *Verifier* checks that $K = p_1(0) + p_1(1)$. If not, it rejects as it knows that the prover must have been lying.

4. *Verifier* picks a random number $0 \leq a_1 \leq 2^n$ and sends it to the Prover.

5. Now, repeat the following from $i = 2$ to $i = n$, or until the Verifier rejects.

   (a) *Prover* sends the alleged coefficients of

   $$p_i(x_i) = \sum_{x_{i+1} \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} p(a_1, \ldots, a_{i-1}, x_i, \ldots, x_n).$$

   Note that p_i is a univariate polynomial in x_i.

   (b) *Verifier* checks that $p_{i-1}(a_{i-1}) = p_i(0) + p_i(1)$. If not, reject.

   (c) *Verifier* picks a random $0 \leq a_i \leq 2^n$ and sends it to the prover.

6. *Prover* sends the alleged number $c = p(a_1, \ldots, a_n)$

7. *Verifier* checks that $p(a_1, \ldots, a_n) = c$. If not, reject. If so, then the verifier concludes that K is indeed the correct number of assignments of ϕ.

As an example, suppose that $\phi = (x_1 \vee \neg x_2)$, with $K = 3$ assignments, and $p(x_1, x_2) = 1 - x_2 + x_1 x_2$. Here, $p_1(x_1) = 1 + x_1$, and so the correct prover sends back $c_0 = 1, c_1 = 1$. Now, the verifier checks that $p_1(0) + p_1(1) = 1 + 2 = 3$; this check passes. Then the verifier picks a number a_1; for example, $a_1 = 6$, and sends that to the prover. The prover now has to compute the coefficients of $p_2(x_2) = 1 + 5x_2$ and send them to the verifier. The verifier checks that $p_1(6) = p_2(0) + p_2(1) = 7$. It then picks another number, say $a_2 = 4$, and sends it to the prover. The prover then computes $p(a_1, a_2) = 21$, and sends 21 to the verifier. Now, the verifier checks that $p(a_1, a_2) = 21$, and concludes that 3 is indeed the number of the satisfying assignments to ϕ.

If the prover always produces the correct coefficients and K is the correct number of assignments, then it is clear that the verifier will accept. To show that the probability of accepting an incorrect value is small, note that two distinct univariate polynomials of degree $d + 1$ may agree on at most d values of the variable, and the probability of randomly picking one of these d values is at most $d/2^n$. This is the probability that any check $p_{i-1}(a_{i-1}) = p_i(0) + p_i(1)$ passes incorrectly. Thus, if K is wrong, then the probability that the verifier rejects is at least $(1 - d/2^n)^n$, which is much larger than $2/3$. □

3.5 IP = PSPACE

A very similar protocol can be used to check whether a quantified Boolean formula $\Phi = \forall x_1 \exists x_2 \forall x_3 \ldots Q x_n \phi(\bar{x})$ is true: here, Q is either $\forall$ or $\exists$ depending on n. There, existential quantifiers correspond to a sum, and universal quantifiers to a product of the values of the arithmetized formula under them. For example, suppose that $\Phi = \forall x_1 \exists x_2 \forall x_3 (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3)$. Then the polynomial p_ϕ for the formula $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3)$ is $(1 - x_2 + -x_1 x_2)(1 - x_1 x_3)$. Now, given

Boolean values for x_1 and x_2, the formula $\forall x_3(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3)$ is true when polynomial $p_\phi(x_1, x_2, 0) \cdot p_\phi(x_1, x_2, 1) = (1 - x_2 + x_1 x_2)(1 - x_2 + x_1 x_2) \cdot (1 - x_1)$ is 1; false corresponds to 0. Proceeding in this manner, we obtain an arithmetic expression which is equal to 1 iff Φ is true, and to 0 iff Φ is false.

Note that the universal quantifier created a product of polynomials, thus doubling the degree. If there are n universal quantifiers, then the resulting degree can be exponential in the number of variables, thus the protocol above would need exponentially many coefficients to describe such a polynomial. To get around this problem, intermediate polynomials can be converted into a multilinear form by noticing that

$$\begin{aligned} p(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n) =& x_i \cdot p(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) + \\ &+ (1 - x_i) \cdot p(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) \end{aligned}$$

over Boolean values of x_i.

With this modification, essentially the same protocol can be used to check if Φ is true, except the verifier would check if the sum $p_{i-1}(a_{i-1}) = p_i(0) + p_i(1)$ is right when x_i is existentially quantified, and if the product $p_{i-1}(a_{i-1}) = p_i(0)p_i(1)$ is correct for a universally quantified x_i.

3.6 Algebrization barrier

Although there were results on the complexity of oracles with respect to which $\mathsf{IP} = \mathsf{PSPACE}$ holds soon after the results themselves have been proven (for example, [For94]), the barrier itself was defined in 2008 by Aaronson and Wigderson [AW09]. There, the main idea was to allow access not just to an oracle A as a language, but also its algebraic extension $\tilde{A}$: here, $\tilde{A}$ is a low-degree polynomial which agrees with A on Boolean inputs. We view an oracle access for a formula or a polynomial as having operations of the form $A(b_1, \ldots, b_k)$, where $b_1 \ldots b_k$ form a query.

Aaronson and Wigderson define an inclusion of complexity classes $C_1 \subseteq C_2$ (for example, $\mathsf{PSPACE}^A \subseteq \mathsf{IP}^{\tilde{A}}$) to be algebrizing when $C_1^A \subseteq C_2^{\tilde{A}}$ no matter how $\tilde{A}$ is chosen, as long as it is a low-degree polynomial extension of A. As for separations, $C_1 \not\subset C_2$ is algebrizing whenever, again for any polynomial extension $\tilde{A}$, $C_1^{\tilde{A}} \not\subset C_2^A$. That is, an inclusion in their setting algebrizes if C_2 can simulate C_1^A with a little more powerful access to the oracle A; and a separation algebrizes if a more powerful access to A for C_1 makes it impossible to simulate C_1 in C_2 with the conventional access.

With these definitions, they show that algebrization indeed provides a pretty precise boundary of the current techniques: most known non-relativizing results algebrize, and most open questions do not. In particular, the proof of $\mathsf{PSPACE}^A \subseteq \mathsf{IP}^{\tilde{A}}$ uses the same protocol as the original proof, with the verifier relying on the oracle access to $\tilde{A}$ to evaluate $p_\phi(a_1, \ldots, a_n)$. However, questions such as P vs NP do not algebrize: an oracle A and its algebraic extension $\tilde{A}$ can be constructed for which $\mathsf{NP}^{\tilde{A}} \subseteq \mathsf{P}^A$, whereas for a different B with its extension

$\tilde{B}$, $\mathsf{NP}^B \not\subset \mathsf{P}^{\tilde{B}}$. A number of other open questions are similarly non-algebrizing. The main tool used for proving such independence results is communication complexity.

But these results need a different kind of oracle on two sides of an inclusion or a separation. For what kinds of oracles would it be possible to use the same language as an oracle on both sides? In 1994, Fortnow [For94] showed how to construct a language that encodes its algebraic extension, and proved that with respect to these kinds of languages, $\mathsf{IP} = \mathsf{PSPACE}$. He constructs such "self-algebrizing" language inductively. (A multilinear extension of $TQBF^L$ gives another example of a language that encodes its own algebraic structure). Let L be a language, and let $\tilde{L}$ be the unique multilinear extension of L. Start by setting $A(0, x_1, \ldots, x_n) = L(x_1, \ldots, x_n)$. Now, if $\tilde{L}(x_1, \ldots, x_n) > 0$, put $(1, x_1, \ldots, x_n)$ in A. Finally, $\forall i \geq 0$, set $A(i+2, x_1, \ldots, x_n)$ to be the i^th bit of $\tilde{L}(x_1, \ldots, x_n)$. Thus, a value of $\tilde{L}(x_1, \ldots, x_n)$ could be obtained by using A as an oracle.

Though $\mathsf{IP} = \mathsf{PSPACE}$ with respect to any oracle of this form, it is not clear whether there are such oracles giving differing outcomes for major open questions such as P vs NP.

3.7 Axiomatizing algebrization

To create a theory capturing algebrizing techniques, we start with the theory RCT for relativizing techniques, and use additional axioms to keep only the standard models with enough algebraic structure [IKK09]. To achieve this goal, we add an axiom stating that NP is a class of languages that have solutions verifiable by polynomial-time computable low-degree polynomials in polynomially many variables. There, the "polynomial-time computable" is in the setting of RCT, that is, functions definable by Cobham axioms without minimality. We call the resulting theory ACT, for "algebraic complexity theory".

Now, the models of ACT can still contain spurious functions interpreted as polynomial-time, as long as they are closed under polynomial-time operations. However, now such functions used as polynomial-time verifiers have to be representable by low-degree polynomials. Alternatively, we can add an axiom redefining polynomial time, by stating that every polynomial time function has a unique polynomial-time computable witness.

For example, consider the classic NP-complete problem Independent Set: given a graph and a number k, check if there are k vertices in the graph such that there are no edges between any of these k vertices; this set of vertices is an independent set in the graph. Let input variables $x_{i,j}$ be 1 if there is an edge between vertices i and j, and be 0 otherwise. Let witness variables y_i be 1 iff vertex i is in the independent set, and 0 otherwise. Now, the polynomial

$$f(x, y) = \Pi_{i,j}(x_{i,j} + y_i + y_j - 3) \cdot \Pi_{t<k}(\Sigma_{i=1}^{n} y_i - t)$$

will be non-zero if and only if there is an independent set of size at least k in the graph. There, the first product will be 0 if there is an edge between two

vertices i, j for which $y_i = y_j = 1$, and the second product is 0 if there are only $t < k$ variables y_i that are 1.

This view of computation is less black-box than the original Cobham axioms: we do require polynomial-time functions to have some structure. However, there are still abitrarily powerful functions that could show up in models of ACT. In particular, a characteristic function for any oracle built using Fortnow's construction described above, the "self-algebrizing oracles", would be a function with respect to which this axiom is satisfied, that is, NP has witnesses definable by low-degree polynomial-time computable polynomial families.

Now, this theory is clearly more powerful than RCT. It is possible to show that nearly all results that algebrize in the sense of [AW09] are provable in ACT, and open questions that require non-algebrizing techniques are independent of ACT. In particular, IP = PSPACE is provable in ACT, and P vs NP is independent of it.

One notable exception is MIP = NEXP by Babai, Fortnow and Lund [BFL91]: when a randomized polynomial-time verifier is allowed to interact with several provers that do not talk to each other, it becomes possible to solve not just PSPACE problems, but ones as hard as non-deterministic exponential time. This equaity is also independent of ACT, even though Aaronson and Wigderson show in [AW09] that a version of this statement algebrizes. However, they only allow a NEXP Turing machine to ask oracle queries of polynomial length, to make it "more fair" for MIP, where the verifier cannot possibly ask longer queries. But such a restriction makes NEXP^A much too weak [IKK]. Consider an oracle $A = \{\langle M, x, 1^t\rangle \mid M \text{ is a non-deterministic Turing machine and } M^A \text{ accepts } x \text{ on some path with all oracle queries shorter than } t\}$. With respect to this oracle, $\mathsf{NEXP}^A = \mathsf{P}^A$ when NEXP^A is restricted to ask only polynomially long queries. This goes against the time hierarchy theorem, which relativizes...

This axiomatic approach is one example where considering a complexity barrier from the logic viewpoint, as an independence of a logic theory, allows for a more convenient setting. Moreover, with the closure under logic operations it becomes possible to argue about composite statements such as "BPP = P and P ≠ NP": this statement is independent of ACT, as well.

4 Efficient reasoning, circuits and natural proofs

4.1 Bounded arithmetic

Before, we talked about creating specific theories of arithmetic to formalize polynomial-time computation and its relativizations. However, there is an area of mathematical logic that is specifically developed to study the reasoning corresponding to efficient computation, such as reasoning with polynomial-time definable concepts. Starting with Parikh's [Par71] fragment of Peano Arithmetic where induction is limited to bounded formulas, followed by Cook's equational theory PV [Coo75] and then Buss' theories [Bus86], bounded arithmetic (term coined by Buss) became one of the standard ways to work with complexity the-

ory concepts in the logic framework. By contrast with RCT and ACT described above, the polynomial time computable objects are now indeed polynomial time, and, moreover, the reasoning power of bounded arithmetic theories is severely restricted by allowing only reasoning with such "efficient" concepts.

This is accomplished by restricting, for example, the induction axiom of Peano Arithmetic to formulas where all quantifiers are bounded by terms in the language. The resulting theory $I\Delta_0$ captures the linear-time hierarchy, but a number of theories with richer language (including $x\#y = 2^{|x|\cdot|y|}$) capture exactly polynomial-time computable functions. Another way is to construct two-sorted theories operating on strings as well as numbers, and length of strings, as well as value of numbers, bounded by the term. In this case, there is no need to introduce # in the language. There is a direct correspondence between these theories and Buss' hierarchies via "RSUV isomorphism" [Raz93, Tak93]. One advantage of such characterization is that "reasoning with concepts from a class C" is easy to state using finite model theory characterizations of the corresponding class [Kol12, CN08]: for example, induction over Grädel's second-order Horn formulas [Grä91] gives a theory of polynomial-time reasoning [CK03].

We will skip the formal definitions; please see Krajicek [Kra95], Buss [Bus86], Pudlák and Hajek [HP98], and Cook and Nguyen [CN08] for more information.

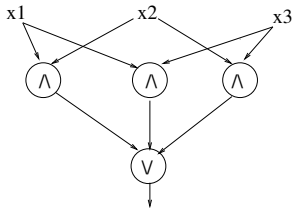
These theories are too weak to prove the totality of exponentiation, that is, that for every x there exists $y = 2^x$. And overall, independence results for them mean "not provable with computationally easy reasoning". For example, in Buss' theory S^1_2 which operates with NP-definable predicates and has induction on the length of a number, any bounded existential statement ϕ for which S^1_2 proves that $\phi \in \mathsf{NP} \cap \mathsf{coNP}$ can be witnessed in polynomial time. This result, which is known as Buss' witnessing theorem, immediately implies that S^1_2 cannot prove that $\mathsf{P} \neq \mathsf{NP} \cap \mathsf{coNP}$.

So what kinds of results, in particular complexity results, can be proven using that kind of limited reasoning? It is possible that these theories, or just slight extension of these, are sufficient to formalize the known complexity results: see, for example, [Raz95a], Krajicek [Kra95], Cook and Nguyen [CN08] or Pudlák [Pud13]. Some of the more recent results about provability and unprovability of complexity-theoretic statements in bounded arithmetic are due to Jerabek (formalizing probabilistic reasoning and pseudorandomness in an extension of S^1_2 with a dual weak pigeonhole principle) [Jeř04, Jeř07a, Jeř07b, Jeř09]; many other related results are in the work of Krajicek, Cook, Buss, Razborov, and others.

One may ask whether independence of P vs. NP from a very strong classical theory, such as ZFC or Peano arithmetic, is a possibility that should be considered instead of focusing on these very restrictive theories. A possible reason why this is quite unlikely is given by Ben-David and Halevi [BDH92]. They show that if P vs. NP is independent of Peano arithmetic augmented with all Π_1 sentences true in the standard model, then NP is "essentially polynomial-time", as there exists a deterministic algorithm for SAT that makes few mistakes, with complexity $n^{f(n)}$ where $f(n)$ is a very slow growing function. An expository paper by Aaronson [Aar03] contains an excellent discussion of this subject.

4.2 Circuit lower bounds

The 1990s have seen a flurry of beautiful results in another area of computational complexity theory: circuit complexity. There, the requirement that there is a single Turing machine solving a problem for all input lengths in bounded time is relaxed to consider a family of computational devices solving the problem, one for each input size, with the sizes of the devices growing only slowly (polynomially) with the number of input bits. The latter setting is often referred to as *non-uniform*. Non-uniformity does make for a much more powerful model: for example, the unary Halting Problem, asking if the n^{th} Turing machine halts on blank input, with n encoded in unary or as $|x|$, is trivially solvable in this model using only constant size devices.



In particular, devices that we will consider here are Boolean circuits: acyclic graphs with inputs as sources, a single output as a sink, and every node (gate) computing a Boolean function over values of gates with edges coming into it. In the most basic case, the Boolean functions at gates are AND, OR and NOT. For example, this circuit computes the majority of three bits.

As time and space in the uniform setting, in the circuit setting the main resources are size (the number of gates) and depth (the length of the longest path from inputs to the output). A class of languages computed by a family of polynomial-size circuits is denoted by $\mathsf{P/poly}$; this is a non-uniform analogue of P. Trivially, $\mathsf{P} \subset \mathsf{P/poly}$, but whether $\mathsf{NP} \subset \mathsf{P/poly}$ is a major open question: a negative answer would be a stronger result than $\mathsf{P} \neq \mathsf{NP}$.

However, restricting the depth of the circuits to be constant (here, we are assuming that gates can have arbitrary fan-in) gives a complexity class AC^0 for which the lower bounds are known. In particular, the function $PARITY(x)$, outputting 1 iff the number of 1s in binary string x is odd, is not AC^0-computable [FSS84, Ajt83, Hås89]. Even allowing parity or modulo a prime gates to appear in these circuits results in a class with strong lower bounds. But how complex is the reasoning needed to prove those statements? And could these techniques be extended to argue about NP vs. $\mathsf{P/poly}$?

4.3 Natural proofs

These questions fascinated researchers in proof complexity for many years, with a number of interesting results proven. But focusing on complexity barriers, let us turn to a series of papers by Razborov [Raz95b, RR97, Raz95a] that have tried to address these questions, using extensively the framework of bounded arithmetic. By far the most well-known of them is a pure complexity paper, though: "Natural proofs" by Razborov and Rudich [RR97]. Introducing the notion of "natural" proofs, they come to a somewhat discouraging conclusion that the circuit lower bound proofs known at the time are "natural", and such proofs cannot resolve NP vs. $\mathsf{P/poly}$, albeit under a believable cryptographic conjecture.

Intuitively, if a proof of a lower bound for a given complexity class is "natural enough" it would contain an algorithm to distinguish easy problems that are in this class from hard problems that are not, according to whether a problem possesses a given "natural" property. Note that, in particular, algorithms from the class for which there is a natural proof would not be able to generate distributions of strings computationally indistinguishable from random (uniform).

Definition 4.1. A property (set) F of Boolean functions is natural if it (or its subset) satisfies three conditions:

1. *Usefulness*: functions in F are infinitely often not in a complexity class C (that is, F can be used to prove lower bounds on C).
2. *Largeness*: A large fraction of all functions are in F.
3. *Constructivity*: Given a truth table of a function f, it is computationally easy to check whether $f \in F$. If it can be checked in a class C', we say that a proof is C'-natural.

For example, the proof that $PARITY(x)$ is not in AC^0 ([FSS84]) works by showing that any function in AC^0 becomes constant if enough of its input variables are set (with high probability over the choice of the subset of variables). The $PARITY(x)$ function, however, does not become constant even if one bit is left unset, and thus $PARITY(x) \notin \mathsf{AC}^0$. So the natural property F is that a function does not become constant under any restriction of large enough fraction of its input variables. This property is useful against AC^0, as functions in AC^0 do not have it. It has largeness by the counting argument (most functions on $n-k$ variables are not constant). And it has constructivity, even AC^0-constructivity: given a truth table of a function on n variables, which is of length 2^n, a depth-3 circuit of size $2^{O(n)}$ can check whether this truth table satisfies the property: just consider all possible restrictions of $n-k$ variables (roughly $2^{O(n)}$ of them), and check that not all input bits corresponding to this restriction are the same.

Razborov and Rudich proceed to show that a number of circuit lower bounds proofs are indeed natural. The main result of [RR97] states, though, that there is no $\mathsf{P/poly}$-natural proof useful against $\mathsf{P/poly}$, provided that there is an exponentially hard pseudo-random generator computable in $\mathsf{P/poly}$. Pseudo-random generators are functions $PRG\colon \{0,1\}^k \to \{0,1\}^{2k}$; such a PRG is s-hard if s is the minimal size of a circuit that can distinguish a random $2k$ bit string from the output of a k-bit generator with probability $\geq 1/s$. The main idea of the proof is to use the $\mathsf{P/poly}$-computable check from the constructivity property to distinguish between the PRG output and the random string for any given $\mathsf{P/poly}$-computable PRG.

It is believed that PRGs based on factoring or the discrete logarithm problem (solving $b^k = g$ over a finite group) are exponentially hard. With that, Razborov and Rudich also show unconditionally that there is no $\mathsf{P/poly}$-natural proof that discrete logarithm problem requires exponential size circuits.

4.4 Natural proof as independence in bounded arithmetic

The formalization of the notion of natural proofs as independence in bounded arithmetic was presented by Razborov in [Raz95a]. The theories he considers are extensions of Buss' S_2^1; there are technical details in allowing these theories to talk about functions. The connection between natural proofs and these theories is not as tight as for the RCT/ACT, though.

Recall that the language of Buss' S_2^1 is the language of arithmetic plus $x\#y = 2^{|x|\cdot|y|}$, and it is axiomatized by the basic axioms describing the operators together with induction on the length of the number. The formulas allowed in the induction are of the form $\exists \bar{x} \leq t(n)\phi(\bar{x}, n)$, where all quantifiers in ϕ are bounded by a term in length of input variables n ("strictly bounded" quantifiers). Generally, theories S_2^i allow up to i alternations of bounded (rather than strictly bounded) quantifiers in the induction.

In Razborov's setting, there is a free relational variable γ added to the theories. This variable is interpreted as encoding a Boolean circuit. There are bounded existential formulas defining various properties of the circuit encoded by γ: its type, size, function it computes, etc. With an extension of these definitions, he proves that $S_2^2(\gamma)$ cannot disprove that γ encodes a circuit of superpolynomial size, under the same assumption as for natural proofs of existence of strong PRGs. For weaker systems, he proves similar statements under weaker assumptions, for some even unconditionally. The proofs rely on a communication complexity characterization of the circuit size. However, this is not quite the same setting as proving lower bounds in S_2^1 itself, rather than arguing about a given circuit γ.

5 Conclusion: avoiding barriers

One can look at complexity barriers from either the pessimistic or the optimistic viewpoint. A pessimist would say that the results beyond the barriers, non-relativizing and non-algebrizing results, are intrinsically hard to prove. Surely it shows just how formidable a problem is when one can prove that "nearly all current techniques" are inadequate for resolving it. And even though there is some truth to this viewpoint, and barrier results historically came from trying to understand the failed attempts to resolve these open problems, there is a bright side to the barriers.

The optimistic view of the barrier results is finding precise properties of techniques that would make it possible to resolve the open problems. For example, relativization tells us that it is fruitless to treat computation as a black box, and representation matters. Algebrization tells us that representing polynomial-time functions by low-degree polynomials, powerful as it is, is not enough to lead us all the way to resolving P vs NP. However, they do tell us where to look: the less "black-box" is our view of computation, the more we can show about it. Besides, the barriers do not tell us to throw out relativizing and algebrizing techniques altogether, they just point that any meaningful approach to resolv-

ing open problems should use at least some non-relativizing and non-algebrizing component.

And indeed, there are already such components known to us. The $\mathsf{MIP} = \mathsf{NEXP}$ result mentioned above does rely on verifying computations by $3CNF$ formulas in a different way than just treating these formulas as polynomials. The PCP theorem, stating that any language in NP has proofs that can be verified using $O(\log n)$ randomness and examining just a constant number of bits of the proof, uses an especially fine-grained view of a computational process, and is not algebrizing in the sense of [AW09] nor is provable in ACT.

Besides, proving a relativizing result can be more useful, in a sense that it automatically generalizes for any oracle. In particular, sometimes it is convenient to consider circuits with SAT or $TQBF$ oracle gates, and any relativizing result about the respective family of circuits generalizes to the circuits with such gates.

In the case of natural proofs, we do know the techniques providing proofs that are not natural, most notably diagonalization and counting arguments. Moreover, there are results proved using techniques that are avoiding both barriers such as Santhanam's proof that $\mathsf{PromiseMA}$ does not have circuits of size n^k for a fixed k [San07]. And a recent result by Williams [Wil14] that there are problems in NEXP not solvable by bounded-depth circuits with arbitrary mod d gates uses both a non-relativizing element and a non-naturalizing element (diagonalization) to avoid both barriers simultaneously.

Viewing complexity barriers as independence of logic theories allows us to make precise what exactly the barriers capture. It is a very natural setting, and a convenient way to specify what exactly is meant by classes of techniques such as relativizing techniques. Besides, we can always hope that the formidable machinery of logic will come to our service if only we could phrase the right questions in the right framework.

6 Acknowledgements

I am very grateful to a number of people for suggestions and comments for this survey. I am especially thankful to Valentine Kabanets and Russell Impagliazzo, who introduced me to the whole topic of axiomatic approach to barriers, answered many of my questions and gave lots of insightful suggestions. I am also grateful to Avi Wigderson, Rahul Santhanam, Toni Pitassi, Sam Buss and Peter van Emde Boas for discussions that greatly influenced this paper.

References

[Aar03] Scott Aaronson. Is P versus NP formally independent? *Bulletin of the EATCS*, 81:109–136, 2003.

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[AIV92] Sanjeev Arora, Russell Impagliazzo, and Umesh Vazirani. Relativizing versus nonrelativizing techniques: The role of local checkability. Manuscript, 1992.

[Ajt83] Miklós Ajtai. σ_1^1-formulae on finite structures. *Annals of pure and applied logic*, 24(1):1–48, 1983.

[ALM+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the Association for Computing Machinery*, 45(3):501–555, 1998. (preliminary version in FOCS'92).

[AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the Association for Computing Machinery*, 45(1):70–122, 1998. (preliminary version in FOCS'92).

[AW09] Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory. *TOCT*, 1(1), 2009. (preliminary version in STOC'08).

[Bab90] László Babai. E-mail and the unexpected power of interaction. In *Structure in Complexity Theory Conference, 1990, Proceedings., Fifth Annual*, pages 30–44. IEEE, 1990.

[Bab16] László Babai. Graph Isomorphism in Quasipolynomial Time. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, 2016.

[BDH92] Shai Ben-David and Shai Halevi. On the Independence of P versus NP. Technical Report 714, Technion, 1992.

[BFL91] László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational complexity*, 1(1):3–40, 1991.

[BG81] Charles H. Bennett and John Gill. Relative to a Random Oracle A, $P^A \neq NP^A \neq coNP^A$ with probability 1. *SIAM Journal on Computing*, 10(1):96 – 113, 1981.

[BGS75] Theodore P. Baker, John Gill, and Robert Solovay. Relativizations of the P=?NP question. *SIAM Journal on Computing*, 4(4):431–442, 1975.

[Bus86] Samuel R. Buss. *Bounded Arithmetic*. Bibliopolis, Naples, 1986.

[CCG+94] Richard Chang, Benny Chor, Oded Goldreich, Juris Hartmanis, Johan Håstad, Desh Ranjan, and Pankaj Rohatgi. The random oracle hypothesis is false. *Journal of Computer and System Sciences*, 49(1):24–39, 1994.

[CK03] Stephen A. Cook and Antonina Kolokolova. A second-order system for polytime reasoning based on Grädel's theorem. *Annals of Pure and Applied Logic*, 124:193–231, 2003.

[CN08] Stephen A. Cook and Phuong Nguyen. *Logical Foundations of Proof Complexity*. Perspectives in Logic of the Association for Symbolic Logic. Cambridge University Press, 2008.

[Cob64] Alan Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Proceedings of the 1964 International Congress for Logic, Methodology, and Philosophy of Science*, pages 24–30. North-Holland, Amsterdam, 1964.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[Coo72] Stephen A. Cook. A hierarchy for nondeterministic time complexity. In *Proceedings of the Fourth Annual ACM Symposium on Theory of Computing*, STOC '72, pages 187–192, 1972.

[Coo75] Stephen A. Cook. Feasibly constructive proofs and the propositional calculus. In *Proceedings of the Seventh Annual ACM Symposium on Theory of Computing*, pages 83 – 97, 1975.

[Edm65] Jack Edmonds. Paths, trees, and flowers. *Canadian J. Math.*, 17:449–467, 1965.

[For94] Lance Fortnow. The role of relativization in complexity theory. *Bulletin of the European Association for Theoretical Computer Science*, 52:229–244, February 1994. Columns: Structural Complexity.

[For09] Lance Fortnow. A Simple Proof of Toda's Theorem. *Theory of Computing*, 5(7):135–140, 2009.

[FS88] Lance Fortnow and Michael Sipser. Are there interactive protocols for co-NP Languages? *Information Processing Letters*, 28:249 –251, 1988.

[FSS84] Merrick Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, April 1984.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the Association for Computing Machinery*, 38:691–729, 1991.

[Grä91] Erich Grädel. The Expressive Power of Second Order Horn Logic. volume 480 of *LNCS*, pages 466–477. Springer-Verlag, 1991.

[Hås89] Johan Håstad. Almost optimal lower bounds for small depth circuits. In S. Micali, editor, *Randomness and Computation*, pages 143–170, Greenwich, Connecticut, 1989. Advances in Computing Research, vol. 5, JAI Press.

[HH76] Juris Hartmanis and John E. Hopcroft. Independence results in computer science. *ACM SIGACT News*, 8(4):13–24, October 1976.

[HP98] Petr Hájek and Pavel Pudlák. *Metamathematics of First-Order Arithmetic*. Perspectives in Mathematical Logic. Springer Berlin Heidelberg, 1998.

[HS64] Juris Hartmanis and Richard E. Stearns. Computational complexity of recursive sequences. In *Proceedings of the Fifth Annual IEEE Symposium on Foundations of Computer Science*, pages 82–90, 1964.

[HS65] Juris Hartmanis and Richard E. Stearns. On the computational complexity of algorithms. *Transactions of the AMS*, 117:285–306, 1965.

[HS66] Fred C Hennie and Richard E Stearns. Two-tape simulation of multitape Turing machines. *Journal of the Association for Computing Machinery*, 13(4):533–546, 1966.

[IKK] Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. An Axiomatic Approach to Algebrization. (in preparation).

[IKK09] Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. An Axiomatic Approach to Algebrization. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, pages 695–704, 2009.

[Jeř04] Emil Jeřábek. Dual weak pigeonhole principle, Boolean complexity, and derandomization. *Annals of Pure and Applied Logic*, 129:1–37, 2004.

[Jeř07a] Emil Jeřábek. Approximate counting in bounded arithmetic. *Journal of Symbolic Logic*, 72(3):959–993, 2007.

[Jeř07b] Emil Jeřábek. On independence of variants of the weak pigeonhole principle. *Journal of Logic and Computation*, 17(3):587–604, 2007.

[Jeř09] Emil Jeřábek. Approximate counting by hashing in bounded arithmetic. *Journal of Symbolic Logic*, 74(3):829–860, 2009.

[JKRS09] Ali Juma, Valentine Kabanets, Charles Rackoff, and Amir Shpilka. The black-box query complexity of polynomial summation. *computational complexity*, 18(1):59–79, 2009.

[Kol12] Antonina Kolokolova. Expressing versus proving: Relating forms of complexity in logic. *Journal of Logic and Computation*, 22(2):267–280, 2012.

[Kra95] Jan Krajíček. *Bounded Arithmetic, Propositional Logic and Complexity Theory*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1995.

[Lev73] Leonid Levin. Universal sorting problems. *Problems of Information Transmission*, 9:265–266, 1973.

[LFKN92] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the Association for Computing Machinery*, 39(4):859–868, 1992.

[Par71] Rohit Parikh. Existence and feasibility of arithmetic. *Journal of Symbolic Logic*, 36:494 –508, 1971.

[Pud13] Pavel Pudlák. *Logical Foundations of Mathematics and Computational Complexity: A Gentle Introduction*. Springer Monographs in Mathematics. Springer, 2013.

[Raz93] Alexander A. Razborov. An equivalence between second-order bounded domain bounded arithmetic and first-order bounded arithmetic. In Peter Clote and Jan Krajíček, editors, *Arithmetic, proof theory and computational complexity*, pages 247–277. Clarendon Press, Oxford, 1993.

[Raz95a] Alexander A. Razborov. Bounded arithmetic and lower bounds in boolean complexity. In *Feasible Mathematics II*, pages 344–386. Springer, 1995.

[Raz95b] Alexander A. Razborov. Unprovability of lower bounds on circuit size in certain fragments of bounded arithmetic. *Izvestiya: Mathematics*, 59(1):205–227, 1995.

[RR97] Alexander A. Razborov and Steven Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55:24–35, 1997.

[San07] Rahul Santhanam. Circuit lower bounds for Merlin-Arthur classes. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, pages 275–283, 2007.

[Sav70] Walter J. Savitch. Relationships between nondeterministic and deterministic space complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[Sha92] Adi Shamir. IP=PSPACE. *Journal of the Association for Computing Machinery*, 39(4):869–877, 1992.

[SHLI65] Richard E Stearns, Juris Hartmanis, and Philip M Lewis II. Hierarchies of memory limited computations. In *SWCT (FOCS)*, pages 179–190, 1965.

[Sto74] Larry Joseph Stockmeyer. *The complexity of decision problems in automata theory and logic.* PhD thesis, Massachusetts Institute of Technology, 1974.

[Tak93] Gaisi Takeuti. RSUV isomorphism. In Peter Clote and Jan Krajíček, editors, *Arithmetic, proof theory and computational complexity*, pages 364–386. Clarendon Press, Oxford, 1993.

[Tod91] Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.

[Tur39] Alan M. Turing. Systems of logic based on ordinals. *Proceedings of the London Mathematical Society. Second Series*, 45:161–228, 1939.

[vEB12] Peter van Emde Boas. Turing machines for dummies - why representations do matter. In *SOFSEM*, pages 14–30, 2012.

[Wil14] Ryan Williams. Nonuniform ACC circuit lower bounds. *Journal of the ACM (JACM)*, 61(1):2, 2014.