

What One Has to Know When Attacking P vs. NP (Extended Abstract)

Juraj Hromkovič¹(✉) and Peter Rossmanith²(✉)

¹ Department of Computer Science, ETH Zürich,
Universitätstrasse 6, 8092 Zürich, Switzerland
juraj.hromkovic@inf.ethz.ch

² Department of Computer Science, RWTH Aachen University, 52056 Aachen,
Germany
rossmani@cs.rwth-aachen.de

Abstract. Mathematics was developed as a strong research instrument with fully verifiable argumentations. We call any consistent and sufficiently powerful formal theory that enables to algorithmically verify for any given text whether it is a proof or not algorithmically verifiable mathematics (AV-mathematics for short). We say that a decision problem $L \subseteq \Sigma^*$ is almost everywhere solvable if for all but finitely many inputs $x \in \Sigma^*$ one can prove either “ $x \in L$ ” or “ $x \notin L$ ” in AV-mathematics.

First, we formalize Rice’s theorem on unprovability, claiming that each nontrivial semantic problem about programs is not almost everywhere solvable in AV-mathematics. Using this, we show that there are infinitely many algorithms (programs that are provably algorithms) for which there do not exist proofs that they work in polynomial time or that they do not work in polynomial time. We can prove the same also for linear time or any time-constructible function.

Note that, if $P \neq NP$ is provable in AV-mathematics, then for each algorithm A it is provable that “ A does not solve SATISFIABILITY or A does not work in polynomial time”. Interestingly, there exist algorithms for which it is neither provable that they do not work in polynomial time, nor that they do not solve SATISFIABILITY. Moreover, there is an algorithm solving SATISFIABILITY for which one cannot prove in AV-mathematics that it does not work in polynomial time.

Furthermore, we show that $P = NP$ implies the existence of algorithms X for which the true claim “ X solves SATISFIABILITY in polynomial time” is not provable in AV-mathematics. Analogously, if the multiplication of two decimal numbers is solvable in linear time, one cannot decide in AV-mathematics for infinitely many algorithms X whether “ X solves multiplication in linear time”.

Finally, we prove that if P vs. NP is not solvable in AV-mathematics, then P is a proper subset of NP in the world of complexity classes based on algorithms whose behavior and complexity can be analyzed in AV-mathematics. On the other hand, if $P = NP$ is provable, we can construct an algorithm that provably solves SATISFIABILITY almost everywhere in polynomial time.

1 Introduction

Mathematics was developed as a special language in which each word and thus each sentence has a clear, unambiguous meaning, at least for anybody who mastered this language. The goal was not only to communicate with unambiguous interpretations, but to create a powerful research instrument that enables everybody to verify any claim formulated in this language.

This way, experiments and mathematics became the main tools for discovering the world and for creating our technical world. The dream of Leibniz was to develop such a formal language, in which almost every problem can be formulated and successfully analyzed by a powerful calculus (thus his famous words “Let us calculate, without further ado, to see who is right.”). After introducing logic as a calculus for verifying the validity of claims and proofs, there was hope to create mathematics as a perfect research instrument (see, for instance, Hilbert [5]). In 1930, Gödel [4] showed that mathematics will never be perfect, that is, that the process of increasing the power of mathematics as a research instrument is infinite. An important fact is that, in each nontrivial mathematics based on finitely many axioms, one can formulate claims in the language of mathematics whose validity cannot be verified inside the same mathematics.

Since the introduction of the concept of computational complexity, computer scientists have not been able to prove nontrivial lower bounds on the complexity of concrete problems. For instance, we are unable to prove that the multiplication of two decimal numbers cannot be computed in linear time, that matrix multiplication cannot be computed in $O(n^2)$ time, or that reachability cannot be solved in logarithmic space. In this paper, we strive to give an explanation for this trouble by showing that the concept of computational complexity may be too complex for being successfully mastered by mathematics. In particular, this means that open problems like **P** vs. **NP** or **DLOG** vs. **NLOG** could be too hard to be investigated inside of current mathematics. In fact, we strive to prove results about the unprovability of some mathematical claims like Gödel [4] did. However, the difference is that we do not focus on meta-statements about mathematics itself, but on concrete fundamental problems of complexity theory that are open for more than 40 years. Interestingly, fundamental contributions in this direction were made by Baker et al. [2], who showed that proof techniques that are sensible to relativization cannot help to solve the **P** vs. **NP** problem, and by Razborov and Rudich [8], who showed that natural proofs covering all proof techniques used in complexity theory cannot help to prove $\mathbf{P} \neq \mathbf{NP}$. Aaronson gives an excellent survey on this topic [1]. Here, we first prove that there are infinitely many algorithms whose asymptotic time complexity or space complexity cannot be analyzed in mathematics. Our results pose the right questions. How hard could it be to prove a superlinear lower bound on multiplication of two decimal numbers (that is, the nonexistence of linear time algorithms for multiplication) if there exist algorithms for which mathematics cannot find out whether they work in linear time or not, or even recognize what they really do? Similarly, we can discuss **SATISFIABILITY** and polynomial time, or **REACHABILITY** and logarithmic space.

In what follows, we focus on the unprovability of theorems in AV-mathematics. First, we reformulate Rice’s theorem [9] about the undecidability of nontrivial semantic problems about programs (Turing machines) to unprovability. More precisely, we say that a decision problem $L \subseteq \Sigma^*$ is *almost everywhere solvable* in AV-mathematics if for all but finitely many inputs $x \in \Sigma^*$, one can prove either “ $x \in L$ ” or “ $x \notin L$ ” in AV-mathematics. Here, we prove that each nontrivial semantic problem about programs is not almost everywhere solvable. This means, for instance, that there exist infinitely many programs for which one cannot prove whether they compute a constant function or not. Note that this has a deep consequence for our judgement about undecidability. Originally one was allowed to view the existence of an algorithm as a reduction of the infinity of the problem (given by its infinite set of problem instances) to finiteness (given by the finite description of the algorithm). In this sense, one can view the undecidability of a problem as the impossibility to reduce the infinite variety of a problem to a finite size. Here, we see another true reason. If, for particular problem instances, one cannot discover in AV-mathematics what the correct output is, then, for sure, there does not exist any provably correct algorithm for the problem.

We use Rice’s theorem on unprovability as the first step for establishing the hardness of the analysis of computational complexity in AV-mathematics. We will succeed to switch from programs to algorithms as inputs,¹ and ask which questions about algorithms are not solvable almost everywhere. We prove results such as

- (i) for each time-constructible function $f(n) \geq n$, the problem whether a given algorithm works in time $O(f(n))$ is not almost everywhere solvable in AV-mathematics, and
- (ii) the problem whether a given algorithm solves SATISFIABILITY (REACHABILITY, multiplication of decimal numbers, etc.) is not almost everywhere solvable in AV-mathematics.

Particularly, this also means that there are infinitely many algorithms for which one cannot distinguish in AV-mathematics whether they work in polynomial time or not. Note that this is essential because, if $P \neq NP$ is provable in AV-mathematics, then, for each algorithm A , the statement

$$\begin{aligned} &\text{“}A \text{ does not work in polynomial time} \\ &\text{or } A \text{ does not solve SATISFIABILITY”} \end{aligned} \tag{*}$$

would be provable in AV-mathematics.

In this paper, we show that there exist algorithms, for which it is neither provable that they do not work in polynomial time nor that they do not recognize SATISFIABILITY. Moreover, we show that if $P = NP$, then there would exist algorithms for which (*) is not provable.

¹ This means that one has a guarantee that a given program is an algorithm, or even a proof that the given program is an algorithm may be part of the input.

Finally, we show that if $P = NP$ is not provable in AV-mathematics, then $P_{\text{ver}} \neq NP_{\text{ver}}$ where P_{ver} and NP_{ver} are counterparts of P and NP in the world of algorithms that can be analyzed in AV-mathematics. On the other hand, if $P = NP$ is provable, then there exists a constructive proof of this fact: There is a concrete algorithm that provably solves an NP -complete problem in polynomial time.

We do not present the shortest way of proving our results. We present here the genesis of our ideas to reach this goal. This is not only better for a deeper understanding of the results and proofs, but also for deriving several interesting byproducts of interest. Note that combining some ideas from the following chapters with some fundamental theorems from ZFC, one can get shorter proofs for some of our results in ZFC if ZFC is consistent.

2 Rice's Theorem on Unprovability

The starting point for our unprovability results is the famous theorem of Chaitin [3], stating that one can discover the Kolmogorov complexity² for at most finitely many binary strings. Let, for each binary string $w \in \{0,1\}^*$, $K(w)$ denote the Kolmogorov complexity of w . As the technique is repeatedly used in this paper, we prefer to present our version of this theorem as well as a specific proof. In what follows, let Σ_{math} be an alphabet in which any mathematical proof can be written. Furthermore, let λ denote the empty word.

Theorem 1 (Chaitin [3]). *There exists $d \in \mathbb{N}$ such that, for all $n \geq d$ and all $x \in \{0,1\}^*$, there does not exist any proof in AV-mathematics of the fact “ $K(x) \geq n$ ”.*

Proof. Let us prove Theorem 1 by contradiction. Suppose there exists an infinite sequence of natural numbers $\{n_i\}_{i=1}^\infty$ with $n_i < n_{i+1}$ for $i = 1, 2, \dots$ such that for each n_i , there exists a proof in AV-mathematics of the claim

$$“K(w_i) \geq n_i”$$

for some $w_i \in \{0,1\}^*$. If, for some i , there exist several such proofs for different w_i 's, let w_i be the word with the property that the proof of “ $K(w_i) \geq n_i$ ” is the first one with respect to the canonical order of the proofs. Then we design an infinite sequence $\{A_i\}_{i=1}^\infty$ of algorithms as follows:

² Recall that the Kolmogorov complexity of a binary string w is the length of the binary code of the shortest program (in some fixed programming language with a compiler) generating w [6, 7].

```

 $A_i$ :   Input:  $n_i$ 
        Output:  $w_i$ 
        begin
             $x := \lambda$ ;
            repeat
                verify algorithmically whether  $x$  is a proof of “ $K(w) \geq n_i$ ”
                for some  $w \in \{0, 1\}^*$ ;
            if  $x$  is a proof of “ $K(w) \geq n_i$ ” then
                output( $w$ ); exit;
            else
                 $x :=$  successor of  $x$  in  $\Sigma_{\text{math}}^*$  in the canonical order;
            end
        end
    end

```

Obviously, A_i generates w_i . All the algorithms A_i are identical except for the number n_i . Hence, there exists a constant c such that each A_i can be described by

$$c + \lceil \log_2(n_i + 1) \rceil$$

bits. This way we get, for all $i \in \mathbb{N}$, that

$$n_i \leq K(w_i) \leq c + \lceil \log_2(n_i + 1) \rceil. \quad (1)$$

However, (1) can clearly hold for at most finitely many different $i \in \mathbb{N}$, and so we got a contradiction. \square

In what follows, we use the terms “program” and “Turing machine” (TM) as synonyms. Likewise, we use the terms “algorithm” and “Turing machine that always halts” as synonyms. The language of a TM M is denoted by $L(M)$. Let $c(M)$ denote the string representation of a TM M for a fixed coding of TMs. Obviously,

$$\text{code-TM} = \{c(M) \mid M \text{ is a TM}\}$$

is a recursive set, and this remains true if we exchange TMs by programs in any programming language possessing a compiler.

Now consider

$$\text{HALT}_\lambda = \{c(M) \mid M \text{ is a TM (a program) and } M \text{ halts on } \lambda\}.$$

If a program (a TM) M halts on λ , there is always a proof of this fact. To generate a proof one can simply let run M on λ , and the finite computation of M on λ is a proof that M halts on λ .

Theorem 2. *There exists a program P that does not halt on λ , and there is no proof in AV-mathematics of this fact.*

Proof. Assume the opposite, that is, for each program there exists a proof that the program halts or does not halt on λ . Then one can compute $K(w)$ for each $w \in \{0, 1\}^*$ as follows.

```

A:   Input:  $w$ 
      Output:  $K(w)$ 
      begin
        generate in the canonical order all programs  $P_1, P_2, \dots$ ;
        for each  $P_i$  do
          search for a proof of “ $P_i$  halts on  $\lambda$ ” or of
            “ $P_i$  does not halt on  $\lambda$ ” in the canonical order of proofs
          if  $P_i$  halts on  $\lambda$  then
            simulate  $P_i$  on  $\lambda$ ;
            if  $P_i$  generates  $w$  then output  $|P_i|$ ; exit;
            else continue with  $P_{i+1}$ ; end
          else
            continue with  $P_{i+1}$ ;
          end
        end
      end

```

Following Theorem 1, one can estimate $K(w)$ for at most finitely many w 's, and so we have a contradiction. \square

Theorem 3. *There exist infinitely many TMs (programs) A that do not halt on λ , and such that there is no proof in AV-mathematics for any of them that A does not halt on λ .*

Proof. Following Theorem 2, there exists a program P such that “ P does not halt on λ ” and there is no proof of this fact in AV-mathematics. There are several ways how to construct infinitely many programs P' such that there is a proof that “ P does not halt on λ ” iff there is a proof that “ P' does not halt on λ ”.

We present the following two ways:

- (i) Take an arbitrary program P_0 that halts on λ . Modify P to P' by taking the simulation of P at the beginning and if the simulation finishes, P' continues with the proper computation of P_0 .
- (ii) For each line of P containing **end**, insert some finite sequence of dummy operations before **end**. \square

Following Rice [9], a set $\mathcal{A} \subseteq \text{code-TM}$ is a semantically nontrivial decision problem on TMs if

- (i) $\mathcal{A} \neq \emptyset$,
- (ii) $\mathcal{A} \neq \text{code-TM}$, and
- (iii) if $c(M) \in \mathcal{A}$ for some TM M , then $c(M') \in \mathcal{A}$ for each TM M' with $L(M') = L(M)$.

Let $\overline{\mathcal{A}} = \text{code-TM} - \mathcal{A}$ for any $\mathcal{A} \subseteq \text{code-TM}$.

Observation 1. *The following is true for any $\mathcal{A} \subseteq \text{code-TM}$. If, for each TM M , there exists a proof in AV-mathematics of either “ $c(M) \in \mathcal{A}$ ” or “ $c(M) \notin \mathcal{A}$ ”, then, for each TM M' , there exists a proof in AV-mathematics of either “ $c(M') \in \overline{\mathcal{A}}$ ” or “ $c(M') \in \mathcal{A}$ ”.*

Proof. A proof of “ $c(M) \in \mathcal{A}$ ” is simultaneously a proof of “ $c(M) \notin \overline{\mathcal{A}}$ ”. A proof of “ $c(M) \notin \mathcal{A}$ ” is simultaneously a proof of “ $c(M) \in \overline{\mathcal{A}}$ ”. \square

Theorem 4 (Rice’s Theorem on Unprovability). *For each semantically nontrivial decision problem \mathcal{A} , there exist infinitely many TMs M' such that there is no proof of “ $c(M') \in \mathcal{A}$ ” and no proof of “ $c(M') \notin \mathcal{A}$ ”, that is, one cannot investigate in AV-mathematics whether $c(M')$ is in \mathcal{A} or not.*

Proof. Let \mathcal{A} be a semantically nontrivial decision problem. The scheme of the proof is depicted in Fig. 2 in the appendix. According to property (iii), either for all D with $L(D) = \emptyset$ we have $c(D) \in \mathcal{A}$, or for all such D we have $c(D) \notin \mathcal{A}$. Following Observation 1, we assume without loss of generality that $c(D) \in \mathcal{A}$ for all D with $L(D) = \emptyset$. Let M_\emptyset be a fixed, simple TM with the property $L(M_\emptyset) = \emptyset$, and thus $c(M_\emptyset) \in \mathcal{A}$. Let \overline{M} be a TM such that $c(\overline{M}) \notin \mathcal{A}$. In particular, $L(\overline{M}) \neq \emptyset$.

We prove Theorem 4 by contradiction. For all but finitely many TMs M' let there exist a proof of either “ $c(M') \in \mathcal{A}$ ” or “ $c(M') \notin \mathcal{A}$ ”. Then we prove that, for all but finitely many TMs M there exists a proof of either “ M halts on λ ” or “ M does not halt on λ ”, which contradicts Theorem 3.

Let M be an arbitrary TM. We describe an algorithm that produces either the proof of “ M does not halt on λ ” if M does not halt on λ or the proof of “ M halts on λ ” if M halts on λ .

First we apply the procedure A (Fig. 2) that transforms M into a TM M'_A with the following properties:

- (1.1) $L(M'_A) = \emptyset$ (and thus $c(M'_A) \in \mathcal{A}$) $\iff M$ does not halt on λ ,
- (1.2) $L(M'_A) = L(\overline{M})$ (and thus $c(M'_A) \notin \mathcal{A}$) $\iff M$ halts on λ .

This is achieved by constructing M'_A in such a way that M'_A starts to simulate the work of M on λ without reading its proper input. If the simulation finishes, M'_A continues to simulate the work of \overline{M} on its proper input. This way, if M does not halt on λ , M'_A simulates the work of M on λ infinitely long and does not accept any input. If M halts on λ , then $L(M'_A) = L(\overline{M})$, because M'_A simulates the work of \overline{M} on each of its inputs.

After that, one algorithmically searches for a proof of “ $c(M'_A) \in \mathcal{A}$ ” or a proof of “ $c(M'_A) \notin \mathcal{A}$ ” by constructing all words over Σ_{math} in the canonical order and

algorithmically checking for each word whether it is a proof of “ $c(M'_A) \in \mathcal{A}$ ” or a proof of “ $c(M'_A) \notin \mathcal{A}$ ”. If such a proof exists, one will find it in finite time. Due to (1.1) and (1.2), this proof can be viewed as (or modified to) a proof of “ M does not halt on λ ” or a proof of “ M halts on λ ”.

The construction of M'_A from M done by A is an injective mapping. As a consequence, if there exists a proof of “ $c(B) \in \mathcal{A}$ ” or a proof of “ $c(B) \notin \mathcal{A}$ ” for all but finitely many TMs B , then there exist proofs of “ M halts on λ ” or “ M does not halt on λ ” for all but finitely many TMs M . This contradicts Theorem 3. \square

Using concrete choices for \mathcal{A} , one can obtain a number of corollaries, such as the following ones.

Corollary 1. *For infinitely many TMs M , one cannot prove in AV-mathematics whether $L(M)$ is in P or not.*

Proof. Choose

$$\mathcal{A} = \{ c(M) \mid M \text{ is a TM and } L(M) \in P \}$$

in Theorem 4. \square

Corollary 2. *For infinitely many TMs, one cannot prove in AV-mathematics whether they accept SATISFIABILITY or not.*

Proof. Choose

$$\mathcal{A} = \{ c(M) \mid M \text{ is a TM and } L(M) = \text{SATISFIABILITY} \}$$

in Theorem 4. \square

Corollary 3. *For infinitely many TMs M , one cannot prove in AV-mathematics whether M is an algorithm working in polynomial time or not.*

Proof. Choose

$$\mathcal{A} = \{ c(M) \mid M \text{ is an algorithm working in polynomial time} \}$$

in Theorem 4. \square

Still, we are not satisfied with the results formulated above. One can argue that the specification of languages (decision problems) by TMs can be so crazy that, as a consequence, one cannot recognize what they really do. Therefore we strive to prove the unprovability of claims about algorithms, preferably for algorithms for which we even have a proof that they indeed are algorithms. This is much closer to the common specifications of NP-hard problems that can be usually expressed by algorithms solving them.

3 Hardness of Complexity Analysis of Concrete Algorithms

Among others, we prove here that, for each time-constructible function f , there exist infinitely many algorithms working in time $f(n) + \mathcal{O}(1)$ for which there is no proof in AV-mathematics that they do. To this end, we construct an algorithm $X_{A,B,f}(M)$ for given

- (i) algorithm A working in $\text{Time}_A(n)$ and $\text{Space}_A(n)$,
- (ii) algorithm B working in $\text{Time}_B(n)$ and $\text{Space}_B(n)$,
- (iii) time-constructible function f with $f(n) \geq n$ (or some other “nice” unbounded, nondecreasing function f), and
- (iv) TM M .

Here, A , B , and f are considered to be fixed by an appropriate choice, and $X_{A,B,f}(M)$ is examined for all possible TMs M . The algorithm $X_{A,B,f}(M)$ works as follows.

```

 $X_{A,B,f}(M)$ : Input:  $w$ 
  begin
    simulate at most  $f(|w|)$  steps of  $M$  on  $\lambda$ ;
    if  $M$  halts on  $\lambda$  during this simulation then
      simulate  $A$  on  $w$ ;
    else
      simulate  $B$  on  $w$ ;
    end
  end

```

We say that two languages L_1 and L_2 are *almost everywhere* equal, $L_1 =_\infty L_2$ for short, if the symmetric difference of L_1 and L_2 is finite. We say that M *almost everywhere* accepts L if $L(M) =_\infty L$.

Claim. If M halts on λ , then $L(X_{A,B,f}(M)) =_\infty L(A)$ and $X_{A,B,f}(M)$ works in time $\text{Time}_A(n) + \mathcal{O}(1)$ and space $\text{Space}_A(n) + \mathcal{O}(1)$.

Claim. If M does not halt on λ , then $L(X_{A,B,f}(M)) = L(B)$ and $X_{A,B,f}(M)$ works in time $\text{Time}_B(n) + f(n)$ and space $\text{Space}_B(n) + f(n)$.

If $L(A)$ and $L(B)$ are not almost everywhere equal, then one can replace the implications in the above two claims by equivalences. Moreover, $X_{A,B,f}(M)$ is an algorithm for each TM M , and if it is provable that A and B are algorithms, it is also provable that $X_{A,B,f}(M)$ is an algorithm.

Let us now present a few applications of the construction of $X_{A,B,f}(M)$. Choose A and B in such a way that $L(A) = L(B)$ and that $\text{Time}_B(n)$ grows asymptotically slower or faster than $\text{Time}_A(n)$. Let $f(n) = n$. Then $L(X_{A,B,f}(M)) = L(A)$ and

- M halts on $\lambda \iff X_{A,B,f}(M)$ works in $\text{Time}_A(n) + \mathcal{O}(1)$,
- M does not halt on $\lambda \iff X_{A,B,f}(M)$ works in $\text{Time}_B(n) + n$.

Corollary 4. *Suppose $\text{Time}_A(n) \in o(\text{Time}_B(n))$ and $\text{Time}_B(n) \in \Omega(n)$. Then there are infinitely many algorithms for which one cannot distinguish in AV-mathematics whether they run in $\Theta(\text{Time}_A(n))$ or in $\Theta(\text{Time}_B(n))$.*

Proof. If one can prove “ $X_{A,B,f}(M)$ works in $\Theta(\text{Time}_A(n))$ ”, then one can also prove that “ M halts on λ ”.

If one can prove “ $X_{A,B,f}(M)$ works in $\Theta(\text{Time}_B(n))$ ”, then there exists a proof that “ M does not halt on λ ”. \square

Choosing $\text{Time}_A(n)$ as a polynomial function and $\text{Time}_B(n)$ as an exponential function, and vice versa, implies the following statement.

Theorem 5. *There exist infinitely many algorithms which do not work in polynomial time, but for which this fact is not provable in AV-mathematics. Similarly, there exist infinitely many algorithms which work in polynomial time, but for which this fact is not provable in AV-mathematics.*

Proof. Let $f(n) = n$. Note that, for a TM M that halts on λ , the claim “ M halts on λ ” is always provable, but for a TM M that does not halt on λ , the claim “ M does not halt on λ ” is not provable for infinitely many TMs M (as shown in Theorem 3). Let M_1 be a TM that does not halt on λ , but for which this fact is not provable in AV-mathematics. Taking A as a polynomial time algorithm and B as an algorithm running in superpolynomial time, the algorithm $X_{A,B,f}(M_1)$ does not run in polynomial time, but the claim “ $X_{A,B,f}(M_1)$ does not run in polynomial time” is not provable in AV-mathematics.

Now, if one takes A as a superpolynomial time algorithm and B as a polynomial time algorithm, then “ M_1 does not halt on λ ” iff “ $X_{A,B,f}(M_1)$ runs in polynomial time”, but this fact is not provable in AV-mathematics, because otherwise “ M_1 does not halt on λ ” would be provable as well. \square

Theorem 5 shows how complex it may be to prove that some problem is not solvable in polynomial time since there are algorithms for which it is not provable whether they work in polynomial time or not. But if one takes $\text{Time}_A(n) \in \mathcal{O}(n)$ and $\text{Time}_B(n) \in \Omega(n^2)$, then we even realize that there are algorithms for which it is not provable whether they work in linear time or not. This could indicate why proving superlinear lower bounds on any problem in NP is hard. We are not able to analyze the complexity of some concrete algorithms for any problem, and the complexity of a problem should be something like the complexity of the “best” algorithm for that problem.

Similarly, one can look at the semantics of algorithms. Assume B solves SATISFIABILITY, A solves something else, and both A and B work in time smaller than $f(n) = 1000 \cdot n^n$ (or any sufficiently large time-constructible function f). In that case, $X_{A,B,f}(M)$ works in time $\mathcal{O}(n^n)$, and it solves SATISFIABILITY iff M does not halt on λ . One can also exchange the role of A and B in order to get that $X_{A,B,f}(M)$ solves SATISFIABILITY almost everywhere iff M halts on λ .

Theorem 6. *There are infinitely many algorithms for which it is not provable in AV-mathematics that they do not solve SATISFIABILITY.*

What is clear from Theorems 5 and 6 is that one cannot start proving $P \neq NP$ with the set of all polynomial time algorithms and try to show that none of them solves SATISFIABILITY, because one cannot decide in AV-mathematics for all algorithms whether they are in the set of polynomial time algorithms or not. Analogously, one cannot start with the set of all algorithms solving SATISFIABILITY and then to try to show that their complexity is superpolynomial, because the set of all algorithms solving SATISFIABILITY is also not exactly determinable in AV-mathematics.

In our considerations, one can exchange SATISFIABILITY for any other NP-hard problem or for FACTORIZATION in order to see that proving that these problems are not in P may be very hard.

Let us look at the problem from another point of view. If $P \neq NP$ is provable in AV-mathematics, then, as already stated, for each algorithm A , the following statement is provable in AV-mathematics:

$$\begin{aligned} & \text{“}A \text{ does not work in polynomial time} \\ & \text{or } A \text{ does not solve SATISFIABILITY”}. \end{aligned} \quad (*)$$

On the other hand, if $P = NP$, then one can take $f(n) = n$, and B as a polynomial time algorithm solving SATISFIABILITY and A as a superpolynomial time algorithm computing something else, and consequently get the following theorem.

Theorem 7. *If $P = NP$, then there exist infinitely many algorithms X for which one cannot prove or disprove in AV-mathematics the statement³ “ X solves SATISFIABILITY in polynomial time”.*

One can play the same game for investigating the computational complexity of the multiplication of two decimal numbers.

Theorem 8. *If multiplication of two decimal numbers is feasible in linear time, then there exist infinitely many algorithms X , for which one cannot decide in AV-mathematics whether “ X solves multiplication in linear time”, or “ X does not solve multiplication or does not work in linear time”.*

Proof. Take A as an algorithm for multiplication with $\text{Time}_A(n) = \Theta(n^2)$, and B as a linear time algorithm for multiplication. Let $f(n) = n$. Then $X_{A,B,f}(M)$ solves multiplication in linear time iff “ M does not halt on λ ”. Hence, if “ $X_{A,B,f}(M)$ solves multiplication in linear time” is provable in AV-mathematics, then “ M does not halt on λ ” is provable as well. \square

Similarly, one can consider space complexity, look at DLOG vs. NLOG with respect to REACHABILITY, and prove similar versions of Theorems 5 to 8.

³ That is, statement (*).

One only needs to modify our scheme by taking a reasonable, unbounded, non-decreasing function g that bounds the space complexity of the simulation of M on λ .

The previous results look promising, but we are still far from proving the unprovability of “ $P \neq NP$ ” in AV-mathematics. This is because we only proved for some algorithms that it is not provable that they “do not work in polynomial time”, and maybe for some other ones that it is not provable that “they do not solve SATISFIABILITY”. We now prove that the intersection of these two sets of algorithms is not empty, i.e., that there exists an algorithm X for which it is neither provable that “ X does not solve SATISFIABILITY”, nor provable that “ X does not work in polynomial time”. To do that, we use the following construction.

Construction of the Algorithm X_1

Let M_1 be a TM that does not halt on λ , and for which this fact is not provable in AV-mathematics (such TMs exist due to Theorem 3). Let C be an algorithm that provably solves SATISFIABILITY in exponential time, and works in exponential time on every input. We define, for each TM M , an algorithm $X_C(M)$ as follows.

```

 $X_C(M)$ : Input:  $w$ 
  begin
    simulate at most  $|w|$  steps of  $M$  on  $\lambda$ ;
    if  $M$  halts on  $\lambda$  within  $|w|$  steps then
      reject  $w$ ;
    else
      simulate the work of  $C$  on  $w$ ;
    end
  end

```

The following statements are true:

- M halts on $\lambda \iff X_C(M)$ accepts almost everywhere the empty set $\iff X_C(M)$ works in polynomial time (even in linear time).
- M does not halt on $\lambda \iff X_C(M)$ solves SATISFIABILITY $\iff X_C(M)$ works in exponential time (and does not work in polynomial time).

If, for any TM M , there exists a proof of either “ $X_C(M)$ works in polynomial time”, or “ $X_C(M)$ does not work in polynomial time”, then correspondingly “ M halts on λ ” or “ M does not halt on λ ” would be provable in AV-mathematics. Analogously, if, for any TM M , there exists a proof of “ $X_C(M)$ recognizes SATISFIABILITY” or “ $X_C(M)$ does not recognize SATISFIABILITY”, it is also provable whether M halts on λ or not.

Since M_1 does not halt on λ , and this fact is not provable in AV-mathematics, we have

$$X_1 := X_C(M_1) \text{ solves SATISFIABILITY,}$$

but it is neither provable that

“ X_1 does not work in polynomial time”

nor that

“ X_1 solves SATISFIABILITY”

Hence, we have the following theorem.

Theorem 9. *There exists an algorithm X_1 , for which it is neither provable whether X_1 recognizes SATISFIABILITY nor provable whether X_1 works in polynomial time.*

Unfortunately, this is not a proof of the fact that (*) is not provable for X_1 , i. e., that

“ X_1 does not work in polynomial time or X_1 does not solve SATISFIABILITY”

is not provable in AV-mathematics (i.e., we did not prove this way that “ $P \neq NP$ ” is not provable in AV-mathematics). Even the opposite is true. From the construction of $X_C(M)$, we see that, for each TM M , the statement (*) is provable for $X_C(M)$. Hence, we have something like an uncertainty principle about properties of algorithms. There is a proof of the statement “ $\alpha(X_1) \vee \beta(X_1)$ ” for the algorithm X_1 , but there does neither exist a proof of “ $\alpha(X_1)$ ” nor a proof of “ $\beta(X_1)$ ”.

Again, note that we can do the same as in Theorem 9, due to the construction of $X_C(M)$, for

1. any NP-hard problem or for FACTORIZATION by exchanging SATISFIABILITY by one of these in the construction of X_1 ,
2. the multiplication of two decimal numbers by taking C as an algorithm that computes multiplication in superlinear time.

4 P vs. NP in AV-Mathematics and the Existence of Constructive Proofs

In this chapter, we outline and discuss some important consequences of our work. As we showed, the computational complexity of some algorithms cannot be analyzed in AV-mathematics. Let us consider classes based only on algorithms that can be analyzed in AV-mathematics. Let Φ be a logic (formal system) that is powerful enough to specify any language in NP. Let $L(\alpha)$ denote a language determined by a specification α from Φ . We define the following classes

$P_{\text{ver}} = \{L(M) \mid \alpha \text{ is a specification from } \Phi \text{ and } M \text{ is an algorithm (a TM that always halts) and there exists a proof in AV-mathematics that } M \text{ works in polynomial time and recognizes } L(\alpha)\},$

$NP_{\text{ver}} = \{L(M) \mid \alpha \text{ is a specification from } \Phi \text{ and } M \text{ is a nondeterministic TM that provably in AV-mathematics works in polynomial time and accepts } L(\alpha)\},$

Analogously, one can define the class A_{ver} for each complexity class A .

Now we show that the unprovability of $P = NP$ in AV-mathematics immediately implies $P_{\text{ver}} \neq NP_{\text{ver}}$.

Theorem 10. *If the claim “ $P = NP$ ” (“ $NLOG = DLOG$ ”) is not provable in AV-mathematics, then*

$$\begin{aligned} P_{\text{ver}} &\neq NP_{\text{ver}} \\ (DLOG_{\text{ver}} &\neq NLOG_{\text{ver}}) \end{aligned}$$

Proof. There is no doubt about the fact that each known NP-complete problem L is in NP_{ver} , because for each such L , we have a polynomial time nondeterministic TM M with a proof of $L = L(M)$.

If $P_{\text{ver}} = NP_{\text{ver}}$ would hold, then SATISFIABILITY $\in P_{\text{ver}}$. Since $P_{\text{ver}} \subseteq P$, one obtains “SATISFIABILITY $\in P$ ” is provable and consequently “ $P = NP$ ” is provable. \square

We can get similar results for the comparisons of other classes for which the upper classes contain complete problems, with respect to the lower ones, e.g., DLOG vs. NLOG, P vs. PSPACE, etc.

5 Making Nonconstructive Proofs Constructive

It has been noted on several occasions that there is the theoretical possibility that we could be able to prove $P = NP$ in a nonconstructive way and still had no concrete algorithm for any NP-complete problem. Scott Aaronson writes in a recent survey [10]:

Objection: Even if $P = NP$, the proof could be nonconstructive—in which case it wouldn’t have any of the amazing implications discussed in Sect. 1.1, because we wouldn’t know the algorithm.

Response: A nonconstructive proof that an algorithm exists is indeed a theoretical possibility, though one that’s reared its head only a few times in the history of computer science. [...] Even then, however, once we knew that an algorithm existed, we’d have a massive inducement to try to find it. [...]

In a poll about what theoretists think about the future of the P vs. NP-question, including whether they believe them to be equal and when the question will be settled, two persons (out of a total of 100) commented they fear that $P = NP$ will be proved in a non-constructive way and call it a “worst case scenario” [11].

On the other hand, it is well known that a *completely nonconstructive* proof cannot exist – an exercise in the textbook *Computational Complexity* asks the student to show that (under the assumption that $P = NP$) there is a fixed algorithm that “solves” SATISFIABILITY in the following way: it provides satisfying assignments for all yes-instances in polynomial time, but is allowed arbitrary behavior on no-instances [12, p. 350].

While we cannot completely answer the question whether a nonconstructive proof can be converted into a constructive one, we can improve upon the answer to this exercise: if $P = NP$, we can present a concrete algorithm that solves SATISFIABILITY on all but finitely many instances in polynomial time (and not only on yes-instances).

While the possibility of having only a nonconstructive proof might appear to be “disturbing” [12], having a concrete algorithm that solves SATISFIABILITY in polynomial time iff $P = NP$ is also strange: in principle, we could just use such an algorithm to solve NP-complete problems in polynomial time even if we cannot prove that $P = NP$. It is sufficient that $P = NP$ holds. While we can arrive at this strange situation almost only for the $P = NP$ -question, there are other similar questions, where such concrete algorithms indeed exist: we will show that there is a randomized algorithm that solves QSAT in expected polynomial time iff PSPACE = BPP and a deterministic one that solves graph reachability with logarithmic space iff LOGSPACE = NLOGSPACE. We could implement these algorithm today and let them run. Owing to large constants, we probably could not observe the asymptotic behavior, but it is still a strange situation.

In the following, we will use an arbitrary, but fixed enumeration M_1, M_2, \dots , of all Turing machines with the input alphabet $\{0, 1\}$ such that a description of M_i can be computed in time polynomial in i and that M_i can be simulated with only polynomial overhead. In the following, *SlowSAT* denotes an exact algorithm that solves SATISFIABILITY in $O^*(2^n)$ steps, where n is the number of variables. It returns a satisfying assignment on yes-instances and “no” on no-instances. *Simulate*(M_i, w, t) denotes the result of running TM M_i on input $w \in \{0, 1\}^*$ for up to t steps. Figure 1 contains an algorithm that attempts to solve SATISFIABILITY in polynomial time. It succeeds to do so for all but finitely many inputs iff $P = NP$.

Theorem 11. *If $P = NP$, then Algorithm S solves SATISFIABILITY, runs in polynomial time, returns “no” on all no-instances, and returns a satisfying assignment on all but finitely many yes-instances.*

Proof. Assume that $P = NP$. Then there is a number d and infinitely many TMs M_i that solve SATISFIABILITY in time n^d . Among those, we choose one with $i \geq d$ and conclude that there exists a Turing machine M_k that solves SATISFIABILITY in time n^k for all input instances of size n .

If $i < k$, then there is at least one instance $w_i \in \{0, 1\}^{\leq m}$ such that M_i does not solve w_i correctly or runs for more than $|w_i|^i$ steps on w_i . Let $N = \min\{2^{\lceil w_1 \rceil}, \dots, 2^{\lceil w_{k-1} \rceil}\}$. If Algorithm S is run on an input instance I with $|I| > N$, then $m \geq w_i$ for all $i = 1, \dots, k-1$. Hence, M_i is simulated on w_i for at least $|w_i|^i$ steps. This shows that M_i fails to solve w_i as a SATISFIABILITY instance or exceeds its running time bound. Hence, i is added to F . On the other hand, k is *not* added to F , because M_k solves every instance w correctly within $|w|^k$ steps. In the end, Algorithm S returns the result of this simulation.

The running time is polynomial: for inputs of length n , there is only a polynomial number of TMs that are simulated for at most $(\log \log n)^{O(\log \log n)}$ steps

Input: A SATISFIABILITY instance $I \in \{0, 1\}^*$
Output: A satisfying assignment if I is a well-formed yes-instance,
no otherwise

```

 $m := \lceil \log \log |I| \rceil$ 
 $F := \emptyset$ 
for  $i = 1$  to  $m$  do
  for  $w \in \{0, 1\}^{\leq m}$  do
     $r_1 := \text{SlowSAT}(w)$ 
     $r_2 := \text{Simulate}(M_i, w, |w|^i)$ 
    if  $r_1 \not\equiv_w r_2$  then  $F := F \cup \{i\}$  fi
  od
od
 $k := \min\{i \in \mathbb{N} \mid i \notin F\}$ 
if  $k \leq m$  then
   $r := \text{Simulate}(M_k, I, |I|^k)$ 
  if  $r$  is a satisfying assignment to  $I$  then return  $r$ 
  else return “no” fi
fi
return  $\text{SlowSAT}(I)$ 

```

Fig. 1. Algorithm S

(which is polynomial in n). In the end, one more simulation is carried out for at most n^m steps if $n \geq N$. For inputs of length smaller than N , we cannot be sure what the running time is, but it is bounded by a (large) constant. \square

Using similar ideas and concepts from [11–14], we can establish the following results.

Theorem 12. *There is a concrete deterministic logspace-bounded algorithm that solves graph reachability iff $\text{LOGSPACE} = \text{NLOGSPACE}$.*

Theorem 13. *If graph isomorphism is in P , then for every $\epsilon > 0$, there exists an algorithm that solves graph isomorphism and has the following properties:*

1. *It is a randomized algorithm.*
2. *It runs in expected polynomial time.*
3. *For all but finitely many yes-instances, it always answers correctly.*
4. *For the remaining yes-instances, the answer is correct with probability at least $1 - 2^{-n^2}$.*
5. *For all no-instances, it always answers correctly.*

Theorem 14. *There exists a concrete randomized algorithm that solves QBF in expected polynomial time with error probability at most $1/3$ iff $\text{BPP} = \text{PSPACE}$.*

Acknowledgment. We would like to thank Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Kráľovič, Richard Kráľovič, and Georg Schnitger for interesting discussions related to the first verification of the proofs presented here. Essential progress was made during the 40th Mountain Workshop on Algorithms organized by Xavier Muñoz from UPC Barcelona that offered optimal conditions for research work.

A Concept of the Proof of Theorem 4

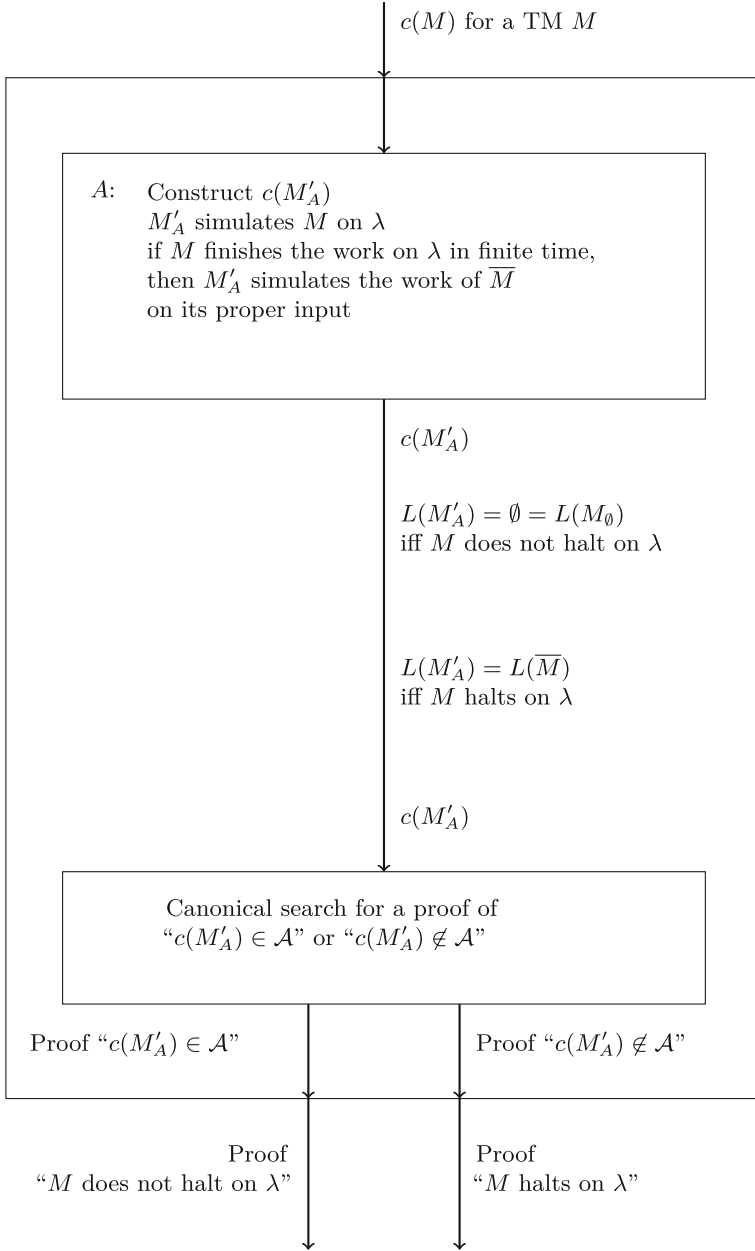


Fig. 2. The schema of the reduction for the existence of proofs in Theorem 4.

References

1. Aaronson, S.: Is P versus NP formally independent? Bull. EATCS **81**, 109–136 (2003)
2. Baker, T.P., Gill, J., Solovay, R.: Relativizations of the $P = ? NP$ question. SIAM J. Comput. **4**(4), 431–442 (1975)
3. Chaitin, G.: Information-theoretic limitations of formal systems. J. ACM **21**(3), 403–424 (1974)
4. Gödel, K.: Über formal unentscheidbare Sätze der Principia Mathematica und verwandte Systeme. Monatshefte für Mathematik und Physik **28**, 173–198 (1931)
5. Hilbert, D.: Die logischen Grundlagen der Mathematik. Math. Ann. **88**, 151–165 (1923)
6. Kolmogorov, A.: Three approaches for defining the concept of information quantity. Probl. Inf. Transm. **1**, 1–7 (1965)
7. Kolmogorov, A.: Logical basis for information theory and probability theory. IEEE Transit. Inf. Theory **14**, 662–664 (1968)
8. Razborov, A.A., Rudich, S.: Natural proofs. J. Comput. Syst. Sci. **55**(1), 24–35 (1997)
9. Rice, H.: Classes of recursively enumerable sets and their decision problems. Transact. ASM **89**, 25–59 (1953)
10. Aaronson, S.: $P \stackrel{?}{=} NP$. In: Electronic Colloquium on Computational Complexity (ECCC) (2017)
11. William, I.: Gasarch: guest column: the second $P=?NP$ poll. SIGACT News **43**(2), 53–77 (2012)
12. Papadimitriou, C.H.: Computational Complexity. Academic Internet Publishers, Ventura (2007)
13. Immerman, N.: Nondeterministic space is closed under complementation. SIAM J. Comput. **17**(5), 935–938 (1988)
14. Szelepcsényi, R.: The method of forced enumeration for nondeterministic automata. Acta Inf. **26**(3), 279–284 (1988)

Fundamentals of Computation Theory

21st International Symposium, FCT 2017, Bordeaux,
France, September 11–13, 2017, Proceedings

Klasing, R.; Zeitoun, M. (Eds.)

2017, XXI, 432 p. 33 illus., Softcover

ISBN: 978-3-662-55750-1