Why the Concept of Computational Complexity is Hard for Verifiable Mathematics (Extended Abstract)

(Extended Abstract)

Why the Concept of Computational Complexity is Hard for Verifiable Mathematics

(Extended Abstract)

Juraj Hromkovič

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

September 16, 2015

Abstract

Mathematics was developed as a strong research instrument with fully verifiable argumentations. We call any formal theory based on syntactic rules 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 prove Rice's theorem for 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. This explains why proving superlinear lower bounds on the time complexity of concrete problems may be really hard. Moreover, we prove that there are infinitely many algorithms for which one cannot decide in AV-mathematics whether they solve SATISFIABILITY or not.

Note that, if $\mathsf{P} \neq \mathsf{NP}$ 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, we finally show that 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 $\mathsf{P} = \mathsf{NP}$ implies the existence of algorithms X for which the 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."

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 [7]). In 1930, Gödel [6] 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 in informatics, computer scientists were not 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 can 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 [6] 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. Interesting, 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 [10], who showed that natural proofs covering all proof techniques used in complexity theory cannot help to prove $\mathsf{P} \neq \mathsf{NP}$. Aaronson gives an excellent survey on this topic [1]. Here, we prove first 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 can one 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.

At this point, one has to be careful with formulating what we really prove here. First, we formulate two hypotheses as possible expectations on mathematics.

H(1) There exists an algorithm for checking whether a given text is a mathematical proof of a given claim.

H(2) The mathematical community can always unambiguously agree in finite time whether a given text is a valid proof of a given claim or not.

For the part of mathematics (given a finite set of axioms) for which H(1) holds, we speak about "*automatically verifiable mathematics*" or "*algorithmically verifiable mathematics*" (AV-mathematics).

We assume that H(2) is a must for mathematics and simply speak about mathematics instead of introducing a new name.

In what follows, we focus on the unprovability of theorems in AV-mathematics. First, we extends Rice's theorem [11] 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 see 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 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 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 $\mathsf{P} \neq \mathsf{NP}$ is provable in AV-mathematics, then, for each algorithm A, the statement

$$\text{“}A \text{ does not work in polynomial time or } A \text{ does not solve SATISFIABILITY”} \qquad (*)$$

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 $\mathsf{P} = \mathsf{NP}$, then there would exist algorithms for which $(*)$ is not provable.

We do not present the shortest way of proving that the complexity of some algorithms cannot be analyzed in AV-mathematics. 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.

[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.

2 Rice's Theorem on Unprovability

The starting point for our unprovability results is the famous theorem of Chaitin [5], stating that one can discover the Kolmogorov complexity[2] 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. Let, in what follows, Σ_{math} be an alphabet in which any mathematical proof can be written. Furthermore, let λ denote the empty word.

Theorem 1 (Chaitin [5]). *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. Let there exist an infinite sequence of natural numbers $\{n_i\}_{i=1}^{\infty}$ with $n_i < n_{i+1}$ for $i = 1, \ldots, \infty$ such that 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:

```
A_i:  Input: $n_i$
      Output: $w_i$
      begin
        $x := \lambda$;
        $T := 0$;
        while $T = 0$ begin
          $x :=$ successor of $x$ in $\Sigma^*_{\text{math}}$ in the canonical order;
          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 $T := 1$;
        end
        output($w$);
      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

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

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

[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 [8, 9].

Alternatively, we can use H(2) as a basis for our arguments. Let Q be any finite object, typically a string over some alphabet. Let $\mathrm{MC}(Q)$ be the shortest binary description of Q that is unambiguous for every member of the mathematical community. This means, each mathematician is able to unambiguously generate Q from $\mathrm{MC}(Q)$. Let $\mathrm{lmc}(Q)$ be the binary length of $\mathrm{MC}(Q)$.

Theorem 2. *There exists a constant $d_{mc} \in \mathbb{N}$ such that, for all $n \geq d_{mc}$ and all words $w \in \{0,1\}^*$, there does not exist any valid proof of the fact*

$$lmc(w) \geq n.$$

Proof. Again, we prove the theorem by contradiction. Let there exist an infinite growing sequence $\{n_i\}_{i=1}^{\infty}$ of positive integers such that, for each $i \in \mathbb{N}$, there exists a mathematical proof of "$\mathrm{lmc}(w_i) \geq n_i$" for some $w_i \in \{0,1\}^*$. Let w_i be such a word that the proof of "$\mathrm{lmc}(w_i) \geq n_i$" is the first proof in the canonical order among the proofs of "$\mathrm{lmc}(y) \geq n_i$" for any $y \in \{0,1\}^*$.

Now, we design an infinite sequence $\{P_i\}_{i=1}^{\infty}$ of mathematical procedures such that P_i unambiguously produces w_i, and hence P_i is an unambiguous description of w_i.

```
$P_i$:  Input: $n_i$
        Output: $w_i$
        begin
            $t := \lambda$;
            $T := 0$;
            while $T = 0$ begin
                $t :=$ successor of $t$ in $\Sigma^*_{\mathrm{math}}$ with respect to the canonical order;
                the mathematical community agrees in finite time whether $t$ is a
                proof of "$\mathrm{lmc}(y) \geq n_i$" or not for some $y \in \{0,1\}^*$;
                if $t$ is a proof of "$\mathrm{lmc}(y_i) \geq n_i$" then $T := 1$;
            end
            output($y$);
        end
```

Since all the procedures P_i differ only in their inputs n_i, and P_i unambiguously generates w_i, we obtain

$$n_i \leq \mathrm{lmc}(w_i) \leq d_{\mathrm{mc}} + \lceil \log_2(n_i + 1) \rceil$$

for all $i \in \mathbb{N}$, where d_{mc} is the description size of the common part of all P_i's. Since this cannot be true for infinitely many different n_i's, we have a contradiction. □

Corollary 1. *One cannot consider to determine $K(w)$ or $lmc(w)$ in mathematics for more than finitely many $w \in \{0,1\}^*$.*

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

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

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 3. *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.

```
begin
    Generate in the canonical order all programs $P_1, P_2, \ldots$ until you find the
    first program that generates the given word $w$ in the following way.
    The binary length of this program is $K(w)$.
    For each $P_i$ one searches in all mathematical proofs (words over $\Sigma_{\mathrm{math}}$) in the
    canonical order in order to find a proof of either “$P_i$ halts on $\lambda$”
    or “$P_i$ does not halt on $\lambda$.”
    Following our assumption such a proof must exist and thus one finds it in finite time.
    If $P_i$ does not halt on $\lambda$, continue with $P_{i+1}$.
    If $P_i$ halts on $\lambda$, simulate $P_i$ on $\lambda$.
    If $P_i$ generates $w$, the binary length of $P_i$ is $K(w)$.
    If $P_i$ does not generate $w$, continue with $P_{i+1}$.
end
```

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

Theorem 4. *There exist infinitely many TMs (programs) A that do not halt on λ, and for which there is no proof in AV-mathematics for any of them that A does not halt on λ.*

Proof. Following Theorem 3 there exists a program P such that “P does not halt on λ” and there is no proof about 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.”

For instance, 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. □

Following Rice [11], a set $\mathcal{A} \subseteq$ code-TM is a semantically nontrivial decision problem on TMs if

(i) $\mathcal{A} \neq \emptyset$,

(ii) $\mathcal{A} \neq$ 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 $\overline{\mathcal{A}} =$ code-TM $- \mathcal{A}$ for any $\mathcal{A} \subseteq$ code-TM.

Observation 1. *The following is true for any $\mathcal{A} \subseteq$ 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 "$c(M') \notin \overline{\mathcal{A}}$" or "$c(M') \in \overline{\mathcal{A}}$."*

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 "$c(M) \in \overline{\mathcal{A}}$." □

Theorem 5 (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 Figure 1. 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 $c(D) \notin \mathcal{A}$ for all D with $L(D) = \emptyset$. Let $M_\emptyset$ be a fixed, simple TM with the property $L(M_\emptyset) = \emptyset$, and thus $c(M_\emptyset) \notin \mathcal{A}$. Let $\overline{M}$ be a TM such that $c(\overline{M}) \in \mathcal{A}$.

We prove Theorem 5 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 4.

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 (Figure 1) that transforms M into a TM M'_A with the following properties:

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

(1.2) $L(M'_A) = L(\overline{M})$ (and thus $c(M'_A) \in \mathcal{A}$) $\iff$ 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 of "$c(M'_A) \notin \mathcal{A}$" by constructing all words over Σ_{math} in canonical order and algorithmically checking for each text whether it is a proof of "$c(M'_A) \notin \mathcal{A}$" or of "$c(M'_A) \in \mathcal{A}$." If such a proof exists, one will find it

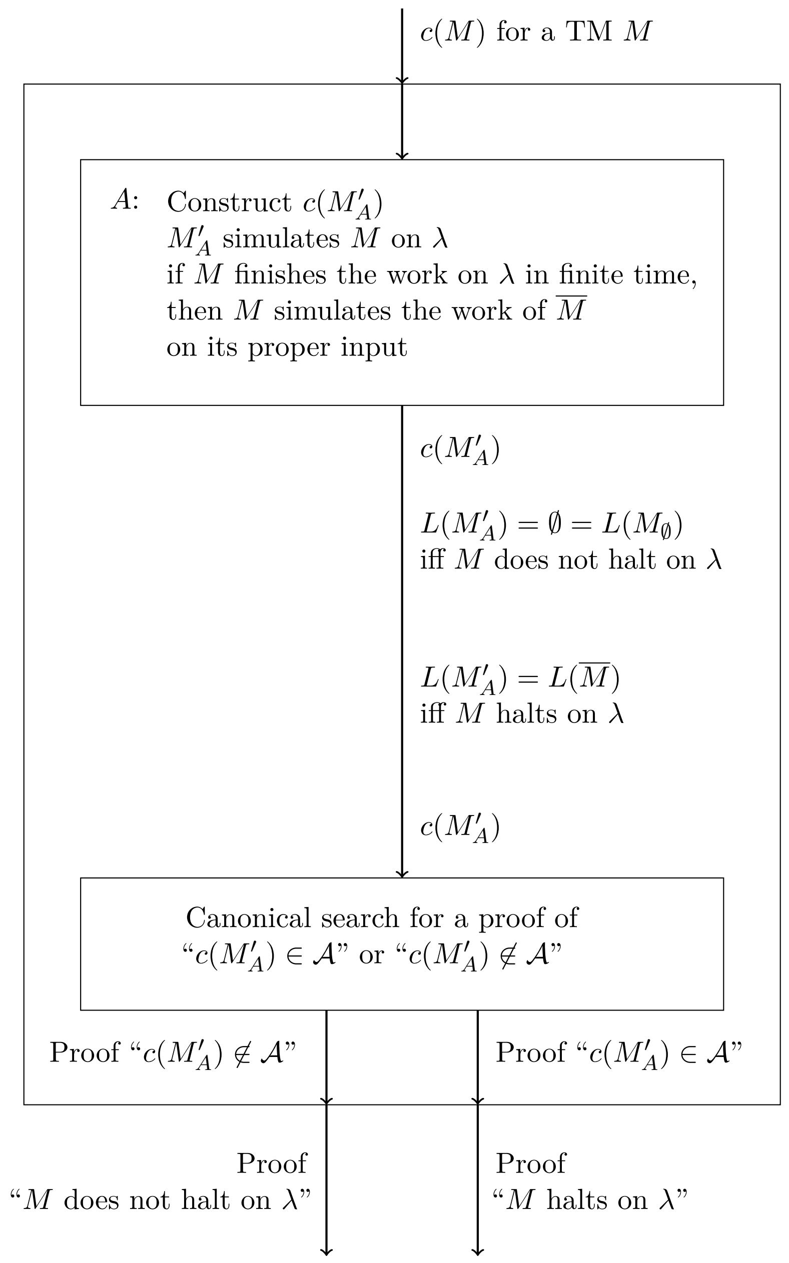
$c(M)$ for a TM M
A: Construct $c(M'_A)$
M'_A simulates M on λ
if M finishes the work on λ in finite time,
then M simulates the work of $\overline{M}$
on its proper input
$c(M'_A)$
$L(M'_A) = \emptyset = L(M_\emptyset)$
iff M does not halt on λ
$L(M'_A) = L(\overline{M})$
iff M halts on λ
$c(M'_A)$
Canonical search for a proof of
"$c(M'_A) \in \mathcal{A}$" or "$c(M'_A) \notin \mathcal{A}$"
Proof "$c(M'_A) \notin \mathcal{A}$"
Proof "$c(M'_A) \in \mathcal{A}$"
Proof
"M does not halt on λ"
Proof
"M halts on λ"

Figure 1. The idea of the proof of Theorem 5

in finite time. Due to (1.1) and (1.2), this proof can be viewed as (or modified to) a proof of that "M does not halt on λ" or of that "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 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 4. ☐

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

Corollary 2. *For infinitely many TMs M, one cannot prove in AV-mathematics whether $L(M)$ is in P or not.*

Proof. Choose $\mathcal{A} = \{c(M) \mid L(M) \in \mathsf{P}\}$ in Theorem 5. ☐

Corollary 3. *For infinitely many TMs, one cannot prove in AV-mathematics whether they accept SATISFIABILITY or not.*

Corollary 4. *For infinitely many TMs M, one cannot prove in AV-mathematics whether M is an algorithm working in polynomial time or not.*

Proof. Choose $\overline{M}$ to be a polynomial time algorithm in the proof of Theorem 5. ☐

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 we which we even have a proof that they indeed are algorithms. This is much closer to the common specifications of NP-hard problems that usually can be 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 $\mathrm{Time}_A(n)$ and $\mathrm{Space}_A(n)$,

(ii) algorithm B working in $\mathrm{Time}_B(n)$ and $\mathrm{Space}_B(n)$,

(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.

```
Input: $w$
begin
1.  Simulate at most $f(|w|)$ steps of $M$ on $\lambda$;
2.  if $M$ halts on $\lambda$ during this simulation then simulate $A$ on $w$;
    else simulate $B$ on $w$;
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 1. *If M halts on λ, then $L(X_{A,B,f}(M)) =_\infty L(A)$ and $X_{A,B,f}(M)$ works in time $Time_A(n) + \mathcal{O}(1)$ and space $Space_A(n) + \mathcal{O}(1)$.*

Claim 2. *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 $Time_B(n) + f(n)$ and space $Space_B(n) + f(n)$.*

If $L(A)$ and $L(B)$ are not almost everywhere equal, then one can exchange the implications in Claims 1 and 2 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 $\mathrm{Time}_B(n)$ grows asymptotically slower or faster than $\mathrm{Time}_A(n)$. Let $f(n) = n$. Then $L_{A,B,f}(M) = L(A)$ and

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

Corollary 5. *There are infinitely many algorithms for which one cannot distinguish in AV-mathematics whether they run in $\mathcal{O}(Time_A(n))$ or in $\mathcal{O}(Time_B(n))$.*

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

Theorem 6. *There exist infinitely many algorithms working in polynomial time, but for which this fact is not provable in AV-mathematics. There exist infinitely many algorithms working in exponential time, but it is not provable in AV-mathematics that they do not work in polynomial time.*

Theorem 6 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 $\mathrm{Time}_A(n) \in \mathcal{O}(n)$ and $\mathrm{Time}_B(n) \in \Omega(n^2)$, 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 7. *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 6 and 7 is that one cannot start with the set of all polynomial-time algorithms and try to show that none of them solves SATISIFIABILITY, 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.

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

$$\text{“}A \text{ does not work in polynomial time or } A \text{ does not solve SATISFIABILITY.”} \qquad (*)$$

On the other hand, if $\mathsf{P} = \mathsf{NP}$, then one can take A as a polynomial-time algorithm solving SATISFIABILITY and B as a superpolynomial algorithm computing something else, and consequently get the following theorem.

Theorem 8. *If $\mathsf{P} = \mathsf{NP}$, then there exist infinitely many algorithms X for which one cannot prove or disprove in AV-mathematics the statement “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 9. *If multiplication of two decimal numbers is doable 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.”*

Similarly, one can consider space complexity, look at DLOG vs. NLOG with respect to REACHABILITY, and prove similar versions of Theorems 6 to 9. One only needs to modify our scheme by taking a reasonable increasing function g that bounds the space complexity of the simulation of M on λ.

The previous results look promising, but we still did not prove the unprovability of “$\mathsf{P} \neq \mathsf{NP}$” in AV-mathematics. This is because we only proved for some algorithms that it is not provable 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 4). 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
          1.  Simulate at most |w| steps of the work of M on λ;
          2.  if M halts on λ within |w| steps then reject w;
              else simulate the work of C on w;
          end
```

The following statements are true.

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

If, for each TM M, there exists either a proof of "$X_C(M)$ works in polynomial time," or a proof of "$X_C(M)$ does not work in polynomial time," then correspondingly "M halts on λ" or "M does not halt on λ" would be provable for each M in AV-mathematics. Analogously, if, for each 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

$X_1 := X_C(M_1)$ 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 an algorithm X_1, for which it is neither decidable whether X_1 recognizes SATISFIABILITY nor decidable whether X_1 works in polynomial time or not. Unfortunately, this is not a proof of the fact that $(*)$ for X_1, i. e.,

"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 "$\mathsf{P} \neq \mathsf{NP}$" is not provable in AV-mathematics). Even the opposite is true. From the construction of $X_C(M)$, for any algorithm C, 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 "$\beta(X_1)$."

Acknowledgment

I would like to thank Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, and Richard Královič for interesting discussions related to the first verification of the proofs presented here. Essential progress was made during the at least 40th Mountain Workshop on Algorithms organized by Xavier Muñoz from UPC Barcelona that offered optimal conditions for research work. Since the final part of the paper was written while flying from Zurich to Mexico City, I am indebted to Hans-Joachim Böckenhauer and Dennis Komm for technical help with preparing this extended abstract.

References

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

[2] Th. P. Baker, J. Gill, and R. Solovay: Relativizations of the P =? NP question. *SIAM Journal on Computing* 4(4), 1975, pp. 431–442.

[3] G. Chaitin: On the lengths of programs for computing finite binary sequences. *Journal of the ACM* 13(4), 1966, pp. 547–569.

[4] G. Chaitin: On the simplicity and speed of programs for computing definite sets of natural numbers. *Journal of the ACM* 16(3), 1969, pp. 407–412.

[5] G. Chaitin: Information-theoretic limitations of formal systems. *Journal of the ACM* 21(3), 1974, pp. 403–424.

[6] K. Gödel: Über formal unentscheibare Sätze der Principia Mathematica und verwandte Systeme. *Monatshefte für Mathematik und Physik* 28, 1931, pp. 173–198.

[7] D. Hilbert: Die logischen Grundlagen der Mathematik. *Mathematische Annalen* 88, 1923, pp. 151–165.

[8] A. Kolmogorov: Three approaches for defining the concept of information quantity. *Probl. Information Transmission* 1, 1965, pp. 1–7.

[9] A. Kolmogorov: Logical basis for information theory and probability theory. *IEEE Transition on Information Theory* 14, 1968, pp. 662–664.

[10] A. A. Razborov and S. Rudich: Natural Proofs. *Journal of Computer and System Sciences* 55(1), 1997, pp. 24–35.

[11] H. Rice: Classes of recursively enumerable sets and their decision problems. *Transaction of ASM* 89, 1953, pp. 25-59.

[12] A. Turing: On computable numbers with an application to the Entscheidungsproblem. In: *Proceeding of London Mathematical Society* 42, 1936, pp. 230–265.