On Optimal Inverters

Yijia Chen
Department of Computer Science
Shanghai Jiaotong University
yijia.chen@cs.sjtu.edu.cn

Jörg Flum
Mathematisches Institut
Albert-Ludwigs-Universität Freiburg
joerg.flum@math.uni-freiburg.de

Abstract

Leonid Levin showed that every algorithm computing a function has an optimal inverter. Recently, we applied his result in various contexts: existence of optimal acceptors, existence of hard sequences for algorithms and proof systems, proofs of Gödel's incompleteness theorems, analysis of the complexity of the clique problem assuming the nonuniform Exponential Time Hypothesis. We present all these applications here. Even though a simple diagonalization yields Levin's result, we believe that it is worthwhile to be aware of the explicit result. The purpose of this survey is to convince the reader of our view.

1. Introduction

Let $\mathbb{F}$ be an algorithm computing a partial function, which we denote by $\mathbb{F}$, too. An *inverter* $\mathbb{I}$ of $\mathbb{F}$ is an algorithm that for all y in the range of $\mathbb{F}$ computes a preimage $\mathbb{I}(y)$ of y, i.e., $\mathbb{F}(\mathbb{I}(y)) = y$. Leonid Levin [17] observed that there is an *optimal inverter* $\mathbb{O}$ of $\mathbb{F}$; optimal with respect to the sum of the running times of the computation of the inverse $\mathbb{O}(y)$ and of the verification of $\mathbb{F}(\mathbb{O}(y)) = y$ (see Theorem 3.2 for the precise statement). For y not in the range of $\mathbb{F}$, the algorithm $\mathbb{O}$ does not halt.

Let Q denote the range of $\mathbb{F}$. Closely related to an inverter $\mathbb{I}$ of $\mathbb{F}$ is an algorithm $\mathbb{I}^{\text{acc}}$ accepting Q: it checks whether $\mathbb{F}(\mathbb{I}(y)) = y$ by successively running $\mathbb{I}$ and $\mathbb{F}$. For an optimal inverter $\mathbb{O}$, does $\mathbb{O}^{\text{acc}}$ inherit some kind of optimality from $\mathbb{O}$? And if so, optimality in what sense?

Verbitsky and Gurevich applied Levin's result to the algorithm $\mathbb{F} = \mathbb{F}_{\text{SAT}}$: on input (α, S), the algorithm $\mathbb{F}_{\text{SAT}}$ checks whether S is an assignment satisfying the propositional formula α; if so, it outputs α. Hence, the set SAT of satisfiable propositional formulas is the range of $\mathbb{F}_{\text{SAT}}$. Verbitsky [27] (see Proposition 3.7) proved for any optimal inverter $\mathbb{O}$ of $\mathbb{F}_{\text{SAT}}$ that

> *the algorithm $\mathbb{O}^{\text{acc}}$ is an algorithm accepting* SAT *that is length-optimal on satisfiable propositional formulas.*

Gurevich [10] (see Theorem 3.8) showed the following result, which is sometimes attributed to Levin himself:

> *if $\mathbb{O}$ (or, equivalently, $\mathbb{O}^{\text{acc}}$) does not run in polynomial time on satisfiable propositional formulas, then* P $\neq$ NP.

Recently, we applied Levin's optimal inverters in apparently quite unrelated contexts: existence of optimal acceptors [2, 3], existence of hard sequences for algorithms and proof systems [6], proofs of Gödel's incompleteness theorems [5], analysis of the complexity of the clique problem assuming the nonuniform exponential hypothesis [1].

A typical scenario may be described as follows: For a given problem (for which we can verify that a string is a solution of a positive instance) there exists an infinite class A of algorithms solving it; however, the algorithms in A are not given in an effective way. Levin's argument provides a method to combine all of them into one single algorithm thereby obtaining, concerning the running time, an optimality with respect to the algorithms in A. Thus, Levin carefully generalizes the idea that a family of algorithms for an NP-problem (where a solution can easily be verified) can be made uniform, at least for the positive instances, by running *all* algorithms in parallel.

As Levin's result is obtained by a straightforward diagonalization, in all applications one can give a direct proof. In some cases, this may even have the benefit of making it easier to grasp the intuition behind the argument in the concrete application. However, in some contexts it is advantageous to be familiar with Levin's result and its terminology: whenever one deals with problems searching for an optimal algorithm, one should check whether the algorithm $\mathbb{F}$ in Levin's result can be defined in such a way that an optimal inverter $\mathbb{O}$ (or the associated $\mathbb{O}^{\text{acc}}$) is the object sought. Furthermore, often it is easier to understand the overall structure of the corresponding proofs by directly applying Levin's result, thus avoiding an explicit diagonalization and the verification of the optimality properties of its outcome. Having this perspective in mind, we believe that Levin's optimal inverters are a valuable tool. The purpose of this survey is to convince the reader of this view.

The content of the different sections is the following. After fixing our notation in Section 2, in Section 3 we prove Levin's result and derive Verbitsky's and Gurevich's applications mentioned above. In Section 4 we show that, under plausible assumptions, for every optimal inverter $\mathbb{O}$ of $\mathbb{F}_{\text{SAT}}$, the algorithm $\mathbb{O}^{\text{acc}}$ is *not* an optimal acceptor of SAT (that is, $\mathbb{O}^{\text{acc}}$ is not an algorithm accepting SAT with optimal running time on satisfiable formulas). Furthermore, we prove a result due to Stockmeyer [25] that there exists a problem Q_0 solvable in exponential time without optimal acceptors. This result plays a central role in Section 7 in our proofs of Gödel's incompleteness theorems: Let T be a first-order theory as considered in Gödel's theorems. For every recursively enumerable set Q we introduce an algorithm $\mathbb{F}_{T,Q}$ with range Q. As already remarked, for every optimal inverter $\mathbb{O}$ of $\mathbb{F}_{T,Q}$ the algorithm $\mathbb{O}^{\text{acc}}$ accepts Q. The theory T proves the equivalence of (the formalizations of) the statements "$\mathbb{O}^{\text{acc}}$ accepts Q" and "T is consistent." For Stockmeyer's Q_0 we can show that T does not prove "$\mathbb{O}^{\text{acc}}$ accepts Q_0" and hence, does not prove its consistency. The reader only interested in this application of Levin's result may skip Section 5 and Section 6 (and even Proposition 4.5 in Section 4).

The relationship between (optimal) acceptors of a problem Q and (polynomially optimal) proof systems for Q has been addressed in various articles [16, 20, 24, 4]. In Section 5 we show that an optimal inverter of a polynomially optimal proof system for a problem Q is an optimal acceptor of Q.

The existence of a hard sequence for an algorithm $\mathbb{A}$ accepting Q witnesses that $\mathbb{A}$ is not an optimal acceptor; similarly, a hard sequence for a proof system $\mathbb{P}$ for Q witnesses that $\mathbb{P}$ is not polynomially optimal. In Section 6 we show how hard sequences for algorithms accepting Q translate into hard sequences of proof systems for Q.

In Section 8, for a suitable $\mathbb{F}$ we prove that the algorithm $\mathbb{O}^{\text{acc}}$ is an algorithm accepting the class of graphs $G = (V(G), E(G))$ having a clique of a given size k in time $2^{o(|V(G)|)}$; for this, we assume that the nonuniform Exponential Time Hypothesis fails.

Finally, in Section 9 we present a space version of Levin's result.

2. Preliminaries

For a partial function g from $\mathbb{N}$ to $\mathbb{N}$ we let $O(g)$ be the set of partial functions f from $\mathbb{N}$ to $\mathbb{N}$ such that

- $\text{dom}(f) \subseteq \text{dom}(g)$ (where $\text{dom}(h)$ denotes the domain of the function h);
- there are $c, k \in \mathbb{N}$ such that $f(n) \leq c \cdot g(n)$ for all $n \in \text{dom}(f)$ with $n \geq k$.

As is common, we often write "$f(n) \leq O(g(n))$," or "for all $n \in \text{dom}(f)$, $f(n) \leq O(g(n))$" instead of "$f \in O(g)$." If we write $O(1)$, we view 1 as the function with constant value 1. In particular, $n^{O(1)}$ denotes the class of all polynomially bounded functions on the natural numbers.

We let Σ be the alphabet $\{0, 1\}$ and denote the length of a string $x \in \Sigma^*$ by $|x|$. We identify decision problems with subsets Q of Σ^*.

Algorithms take strings in Σ^* as inputs. If an algorithm $\mathbb{A}$ on input $x \in \Sigma^*$ eventually halts, its output $\mathbb{A}(x)$ is the string written on the output device. Hence, every algorithm $\mathbb{A}$ computes a partial function from Σ^* to Σ^*, which we denote by $\mathbb{A}$, too. The equality $\mathbb{A}(x) = \mathbb{B}(x)$ for algorithms $\mathbb{A}$ and $\mathbb{B}$ and the string x means that either $\mathbb{A}$ and $\mathbb{B}$ halt on input x with the same output or that neither $\mathbb{A}$ nor $\mathbb{B}$ halts on x. Often we introduce an algorithm implicitly by defining the corresponding function; then this definition will suggest an algorithm.

If $\mathbb{A}$ is an algorithm, $x \in \Sigma^*$, and $\mathbb{A}(x) = 1$ $(\mathbb{A}(x) = 0)$, then we say that $\mathbb{A}$ *accepts* x ($\mathbb{A}$ *rejects* x). The algorithm $\mathbb{A}$ *accepts* the problem Q if

$$\mathbb{A} \text{ accepts } x \iff x \in Q$$

for all $x \in \Sigma^*$. We also say that $\mathbb{A}$ is an *acceptor* of Q.

If $\mathbb{A}$ is an algorithm and $x \in \Sigma^*$, we let $t_{\mathbb{A}}(x)$ be the number of steps of the run of $\mathbb{A}$ on input x. We set $t_{\mathbb{A}}(x) := \infty$ if $\mathbb{A}$ does not halt on input x.

3. Levin's result and first applications

In this section we prove Levin's result on optimal inverters and present the applications due to Verbitsky [27] and Gurevich [10] already mentioned in the Introduction. We start by introducing the concept of an inverter.

Definition 3.1. Let $\mathbb{F}$ be an algorithm. An *inverter of* $\mathbb{F}$ is an algorithm $\mathbb{I}$ that, given as input y in the range of the function computed by $\mathbb{F}$, halts and its output $\mathbb{I}(y)$ is a preimage of y under $\mathbb{F}$ (that is, $\mathbb{F}(\mathbb{I}(y)) = y$). Nothing is required for y not in the range of $\mathbb{F}$.

We often denote the range of the function computed by $\mathbb{F}$ by $\mathrm{rng}(\mathbb{F})$. The following result, Levin's Optimal Inverter Theorem, states that there is an *optimal inverter* $\mathbb{O}$ of $\mathbb{F}$; that means, $\mathbb{O}$ is an inverter that for $y \in \mathrm{rng}(\mathbb{F})$ is optimal (up to polynomial) with respect to the combined time complexity for the computation of the inverse $\mathbb{O}(y)$ and the verification of $\mathbb{F}(\mathbb{O}(y)) = y$ (by computation of the function $\mathbb{F}$). If $\mathbb{F}$ runs in polynomial time on its domain, then the running time of every optimal inverter is polynomially bounded in the running time of any inverter.

Theorem 3.2 (Levin's Optimal Inverter Theorem)**.** *Let $\mathbb{F}$ be an algorithm. Then there is an* optimal inverter*, that is, an inverter $\mathbb{O}$ of $\mathbb{F}$ such that:*

– *For every inverter $\mathbb{I}$ of $\mathbb{F}$ we have for $y \in \mathrm{rng}(\mathbb{F})$,*

$$t_{\mathbb{O}}(y) + t_{\mathbb{F}}(\mathbb{O}(y)) \leq \big(t_{\mathbb{I}}(y) + t_{\mathbb{F}}(\mathbb{I}(y))\big)^{O(1)}. \tag{1}$$

In particular, if $\mathbb{F}$ runs in polynomial time on its domain, then for all $y \in \mathrm{rng}(\mathbb{F})$,

$$t_{\mathbb{O}}(y) + t_{\mathbb{F}}(\mathbb{O}(y)) \leq \big(t_{\mathbb{I}}(y)\big)^{O(1)}. \tag{2}$$

– *The algorithm $\mathbb{O}$ does not halt on inputs $y \notin \mathrm{rng}(\mathbb{F})$.*

Proof: For an algorithm $\mathbb{A}$ define the algorithm $[\mathbb{F} : \mathbb{A}]$ by:[1]

> $[\mathbb{F} : \mathbb{A}]$ // $y \in \Sigma^*$
>
> 1. simulate $\mathbb{A}$ on y
> 2. simulate $\mathbb{F}$ on $\mathbb{A}(y)$
> 3. **if** $\mathbb{F}(\mathbb{A}(y)) = y$ **then** halt with output $\mathbb{A}(y)$ **else** run forever.

Note that:

(a) If $[\mathbb{F} : \mathbb{A}](y)$ is defined, then $y \in \mathrm{rng}(\mathbb{F})$ and $[\mathbb{F} : \mathbb{A}](y)$ is a preimage of y (with respect to $\mathbb{F}$), that is, $\mathbb{F}([\mathbb{F} : \mathbb{A}](y)) = y$.

(b) If $[\mathbb{F} : \mathbb{A}](y)$ is defined, then $t_{\mathbb{A}}(y) + t_{\mathbb{F}}(\mathbb{A}(y)) \leq t_{[\mathbb{F}:\mathbb{A}]}(y) \leq O(t_{\mathbb{A}}(y) + t_{\mathbb{F}}(\mathbb{A}(y)))$.

[1] The first line of the box contains the abbreviation for the algorithm (in this case $[\mathbb{F} : \mathbb{A}]$) and, after the double slash, the inputs we consider (in this case an arbitrary $y \in \Sigma^*$).

We fix an effective enumeration

$$\mathbb{A}_1, \mathbb{A}_2, \ldots.$$

of all algorithms. We obtain the desired optimal inverter by simulating, for any $y \in \Sigma^*$, all $[\mathbb{F} : \mathbb{A}_i]$'s on input y in a diagonal fashion till we get an output:

the first step of $[\mathbb{F} : \mathbb{A}_1]$;
the second step of $[\mathbb{F} : \mathbb{A}_1]$, the first step of $[\mathbb{F} : \mathbb{A}_2]$;
.....
the ith step of $[\mathbb{F} : \mathbb{A}_1]$, the $(i-1)$th step of $[\mathbb{F} : \mathbb{A}_2]$, ..., the first step of $[\mathbb{F} : \mathbb{A}_i]$;
.....

Let us explain more precisely how this inverter accesses $[\mathbb{F} : \mathbb{A}_1]$, $[\mathbb{F} : \mathbb{A}_2]$, ... For this purpose, we let $\mathbb{A}$ be an "enumeration" algorithm that (once having been started) eventually prints out all algorithms. For $i \geq 1$ we denote by $\mathbb{A}_i$ the last algorithm printed out by $\mathbb{A}$ in in the first i steps; in particular, $\mathbb{A}_i$ is undefined if $\mathbb{A}$ hasn't printed any algorithm in the first i steps. Then we let $\mathbb{O}$ be the algorithm:

$\mathbb{O}$ // $y \in \Sigma^*$

1. $\ell \leftarrow 1$
2. simulate the ℓth step of $\mathbb{A}$
3. **for** $i = 1$ **to** ℓ
4. **if** $\mathbb{A}_i$ is defined **then** simulate the $\ell - (i-1)$)th step of $[\mathbb{F} : \mathbb{A}_i]$ on y
5. **if** the simulation halts **then** halt with output $[\mathbb{F} : \mathbb{A}_i](y)$
6. $\ell \leftarrow \ell + 1$
7. goto 2.

The algorithm $\mathbb{O}$ is an inverter: If $\mathbb{O}$ halts on input y, then (see Line 5) $\mathbb{O}(y) = [\mathbb{F} : \mathbb{A}_i](y)$ for some $i \geq 1$. Thus, by (a), $\mathbb{F}(\mathbb{O}(y)) = y$.

The algorithm $\mathbb{O}$ is optimal: Let $\mathbb{I}$ be an inverter of $\mathbb{F}$. We choose the least $j \geq 1$ such that $\mathbb{I} = \mathbb{A}_j$. Then, on input y, the algorithm $\mathbb{O}$ will halt if it reaches Line 4 for $\ell := t_{[\mathbb{F}:I]}(y) + (j-1)$ and $i := j$ (perhaps, $\mathbb{O}$ already halted earlier). Thus, $(t_{[\mathbb{F}:I]}(y) + (j-1))^2$ is an upper bound for the number of those steps of $\mathbb{O}$ simulating one of the $[\mathbb{F} : \mathbb{A}_i]$'s. As j only depends on $\mathbb{I}$, the inequality in (1) follows from (b).

If $\mathbb{F}$ runs in polynomial time on its domain, then, for $y \in \mathrm{rng}(\mathbb{F})$, we have $t_{\mathbb{F}}(\mathbb{I}(y)) \leq |\mathbb{I}(y)|^{O(1)}$ and thus, $t_{\mathbb{F}}(\mathbb{I}(y)) \leq t_{\mathbb{I}}(y)^{O(1)}$. So we get the inequality (2) from (1). □

In the Introduction we described the typical scenario underlying most applications of Levin's result and Levin's result itself. The following remark should help the reader to recognize this scenario in the previous proof, of which we use the terminology.

Remark 3.3. For $y \in \mathrm{rng}(\mathbb{F})$ (a "positive instance") we can verify that a string x is a solution of the equation $\mathbb{F}(x) = y$ (just run $\mathbb{F}$ on x). The class A (mentioned in the description of the scenario in the Introduction) of algorithms solving the problem is the class of inverters of $\mathbb{F}$. It is not decidable, nor even recursively enumerable. The diagonalization over the algorithms $[\mathbb{F} : \mathbb{A}_i]$ carried out above, where $\mathbb{A}_i$ runs over *all* algorithms, yields an element of A with optimal running time on instances $y \in \mathrm{rng}(\mathbb{F})$.

Remark 3.4. Suppose we take multitape Turing machines with input tape and output tape as the computational model for algorithms. Then, in (1), we may replace $\big(t_{\mathbb{I}}(y) + t_{\mathbb{F}}(\mathbb{I}(y))\big)^{O(1)}$ by a quadratic polynomial in $(t_{\mathbb{I}}(y) + t_{\mathbb{F}}(\mathbb{I}(y))$ (the quadratic polynomial depending on $\mathbb{I}$).

Remark 3.5. In [17] Levin states the result with the term $O\big(t_{\mathbb{I}}(y) + t_{\mathbb{F}}(\mathbb{I}(y))\big)$ instead of $\big(t_{\mathbb{I}}(y) + t_{\mathbb{F}}(\mathbb{I}(y))\big)^{O(1)}$ in (1). This better bound is achieved by considering the algorithm that for any $y \in \Sigma^*$ simulates all $[\mathbb{F} : \mathbb{A}_i]$ in the following way:

2^0 step of $[\mathbb{F} : \mathbb{A}_1]$ is simulated;
2^1 steps of $[\mathbb{F} : \mathbb{A}_1]$ are simulated, 2^0 step of $[\mathbb{F} : \mathbb{A}_2]$ is simulated;
2^2 steps of $[\mathbb{F} : \mathbb{A}_1]$ are simulated, 2^1 steps of $[\mathbb{F} : \mathbb{A}_2]$ are simulated, 2^0 step of $[\mathbb{F} : \mathbb{A}_3]$ is simulated;
.....
2^ith steps of $[\mathbb{F} : \mathbb{A}_1]$ are simulated, 2^{i-1} steps of $[\mathbb{F} : \mathbb{A}_2]$ are simulated,
..., 2^0 step of $[\mathbb{F} : \mathbb{A}_{i+1}]$ is simulated;
.....

We learned this proof idea from [7].

Again let $\mathbb{F}$ be an algorithm and denote by Q its range. For every inverter $\mathbb{I}$ there is an algorithm $\mathbb{I}^{\text{acc}}$, canonically linked to $\mathbb{I}$, which accepts Q. In the following proposition we introduce the algorithm $\mathbb{I}^{\text{acc}}$ and relate its running time to that of $\mathbb{I}$. We will use this result again and again.

Proposition 3.6. *Let $\mathbb{F}$ be an algorithm with range Q. For every inverter $\mathbb{I}$ of $\mathbb{F}$ we define the algorithm $\mathbb{I}^{\text{acc}}$ by:*

> $\mathbb{I}^{\text{acc}}$ // $y \in \Sigma^*$
>
> 1. simulate $\mathbb{I}$ on y
> 2. simulate $\mathbb{F}$ on $\mathbb{I}(y)$
> 3. **if** $\mathbb{F}(\mathbb{I}(y)) = y$ **then** accept **else** run forever.

Then:

– *The algorithm $\mathbb{I}^{\text{acc}}$ accepts Q and for all $y \in Q$,*

$$t_{\mathbb{I}}(y) + t_{\mathbb{F}}(\mathbb{I}(y)) \leq t_{\mathbb{I}^{\text{acc}}}(y) \leq O(t_{\mathbb{I}}(y) + t_{\mathbb{F}}(\mathbb{I}(y))). \tag{3}$$

If $\mathbb{F}$ runs in polynomial time on its domain, then for all $y \in Q$,

$$t_{\mathbb{I}}(y) \leq t_{\mathbb{I}^{\text{acc}}}(y) \leq (t_{\mathbb{I}}(y))^{O(1)}. \tag{4}$$

– *The algorithm $\mathbb{I}^{\text{acc}}$ does not halt on inputs $y \notin Q$.*

Proof: All claims immediately follow from the definition of $\mathbb{I}^{\text{acc}}$. □

As first examples we present applications of Levin's optimal inverters to SAT, the satisfiability problem for formulas of propositional logic. We consider the algorithm $\mathbb{F}_{\text{SAT}}$ with

$$\mathbb{F}_{\text{SAT}}(x) := \alpha, \quad \text{if } x = (\alpha, S) \text{ and the assignment } S \text{ satisfies the propositional formula } \alpha. \tag{5}$$

On other inputs the algorithm $\mathbb{F}_{\text{SAT}}$ does not halt. The range of $\mathbb{F}_{\text{SAT}}$ is SAT. As one can verify in linear time whether an assignment S satisfies α, the algorithm $\mathbb{F}_{\text{SAT}}$ runs in polynomial time (even in linear time) on its domain. By the previous proposition, every inverter $\mathbb{I}$ of $\mathbb{F}_{\text{SAT}}$ yields the acceptor $\mathbb{I}^{\text{acc}}$ of SAT, which essentially runs in the same time as $\mathbb{I}$ on satisfiable formulas (see (4)).

Now let $\mathbb{O}$ be an optimal inverter of $\mathbb{F}_{\text{SAT}}$. Is $\mathbb{O}^{\text{acc}}$ an acceptor of SAT optimal in some sense? In the Introduction we have called the kind of optimality shown in the next proposition "length-optimality", in [22] it was called "Levin optimality."

Proposition 3.7 ([27])**.** *Let $\mathbb{O}$ be an optimal inverter of $\mathbb{F}_{\text{SAT}}$. For every algorithm $\mathbb{B}$ accepting* SAT *we have for all $\alpha \in$ SAT,*

$$t_{\mathbb{O}^{\text{acc}}}(\alpha) \leq \big(|\alpha| \cdot \max\{t_{\mathbb{B}}(\alpha') \mid \alpha' \in \text{SAT } \textit{and } |\alpha'| \leq |\alpha|\}\big)^{O(1)}.$$ [2]

Proof: For a propositional formula α and a propositional variable X of α we denote by $\alpha[X \leftarrow \text{TRUE}]$ and by $\alpha[X \leftarrow \text{FALSE}]$ the propositional formulas obtained from α by replacing X by TRUE and by FALSE, respectively. We may assume that the lengths of $\alpha[X \leftarrow \text{TRUE}]$ and of $\alpha[X \leftarrow \text{FALSE}]$ are at most $|\alpha|$.

Clearly, the formula α is satisfiable if and only if $\alpha[X \leftarrow \text{TRUE}]$ or $\alpha[X \leftarrow \text{FALSE}]$ is satisfiable. Using this self-reducibility of SAT we turn any algorithm $\mathbb{B}$ accepting SAT into an inverter $\mathbb{B}'$ of $\mathbb{F}_{\text{SAT}}$:

$\mathbb{B}'$ // α a propositional formula with variables $X_1, \ldots, X_n$

1. simulate $\mathbb{B}$ on α
2. **if** the simulation rejects **then** reject
3. $\alpha' \leftarrow \alpha$ and $S \leftarrow \emptyset$
4. **for** $i = 1$ **to** n **do**
5. $\quad \alpha_1 \leftarrow \alpha'[X_i \leftarrow \text{TRUE}]$
6. $\quad \alpha_2 \leftarrow \alpha'[X_i \leftarrow \text{FALSE}]$
7. $\quad$ in parallel simulate $\mathbb{B}$ on α_1 and α_2
8. $\quad$ **if** the simulation accepts α_1 first
   then $\alpha' \leftarrow \alpha_1$ and $S \leftarrow S \cup \{(X_i, \text{TRUE})\}$,
   else $\alpha' \leftarrow \alpha_2$ and $S \leftarrow S \cup \{(X_i, \text{FALSE})\}$
9. Output (α, S).

As the number n of variables of the formula α is at most $|\alpha|$, we get for $\alpha \in \text{SAT}$,

$$t_{\mathbb{B}'}(\alpha) \leq O\big((|\alpha|+1) \cdot \max\{t_{\mathbb{B}}(\alpha') \mid \alpha' \in \text{SAT and } |\alpha'| \leq |\alpha|\}\big). \tag{6}$$

Thus, for an optimal inverter $\mathbb{O}$ of $\mathbb{F}_{\text{SAT}}$ and $\alpha \in \text{SAT}$, we have:

$$\begin{aligned} t_{\mathbb{O}^{\text{acc}}}(\alpha) &\leq (t_{\mathbb{O}}(\alpha)))^{O(1)} && \text{(by (4) as } \mathbb{F}_{\text{SAT}} \text{ runs in polynomial time on its domain)} \\ &\leq (t_{\mathbb{B}'}(\alpha))^{O(1)} && \text{(by (2) as } \mathbb{F}_{\text{SAT}} \text{ runs in polynomial time on its domain)} \\ &\leq (|\alpha| \cdot \max\{t_{\mathbb{B}}(\alpha') \mid \alpha' \in \text{SAT and } |\alpha'| \leq |\alpha|\})^{O(1)} && \text{(by (6)).} \end{aligned}$$

□

As SAT is NP-complete, we know that P = NP if and only if there is a polynomial time algorithm deciding SAT. Or (see the proof of the implication (iii) ⇒ (i) below), P = NP if and only if there is a polynomial time algorithm accepting SAT and running in polynomial time on satisfiable formulas. We show that P = NP if and only if *the* algorithm $\mathbb{O}^{\text{acc}}$ (where $\mathbb{O}$ is an optimal inverter of $\mathbb{F}_{\text{SAT}}$) is such an algorithm:

Theorem 3.8 ([10])**.** *For an optimal inverter $\mathbb{O}$ of $\mathbb{F}_{\text{SAT}}$ the following statements are equivalent:*

(i) P = NP.

(ii) *$\mathbb{O}$ runs in polynomial time on satisfiable formulas.*

(iii) *$\mathbb{O}^{\text{acc}}$ runs in polynomial time on satisfiable formulas.*

Proof: (i) ⇒ (ii): Assume first P = NP. Then there is a polynomial time algorithm $\mathbb{B}$ deciding SAT. The corresponding inverter $\mathbb{B}'$ defined in the previous proof runs in polynomial time on satisfiable formulas by (6). Hence, the optimal inverter $\mathbb{O}$ runs in polynomial time on satisfiable formulas (by (2) as $\mathbb{F}_{\text{SAT}}$ runs in polynomial time on its domain).

[2] $\mathbb{F}_{\text{SAT}}$ is a linear time algorithm. Using Remark 3.4 or Remark 3.5 one gets bounds on the degree of the corresponding polynomial. However, in this survey paper we do not address this aspect any more.

The implication (ii) ⇒ (iii) follows from (4). We turn to (iii) ⇒ (i): Assume that $\mathbb{O}^{\text{acc}}$ runs in polynomial time on satisfiable formulas. Let $p \in \mathbb{N}[X]$ be a corresponding polynomial. Then we can decide SAT in polynomial time (and hence, P = NP) by running $\mathbb{O}^{\text{acc}}$ on every propositional formula α at most $p(|\alpha|)$ steps and rejecting if $\mathbb{O}^{\text{acc}}$ does not halt during these steps. □

We close this section with a general remark. Let $\mathbb{F}$ be an algorithm. We assume that its range Q is *decidable* and fix an algorithm $\mathbb{A}$ deciding Q. For every inverter $\mathbb{I}$ of $\mathbb{F}$ we get an algorithm $\mathbb{I}^{\text{dec}}$ *deciding* Q by running $\mathbb{I}^{\text{acc}}$ and $\mathbb{A}$ in parallel:

$\mathbb{I}^{\text{dec}}$ // $y \in \Sigma^*$

1. in parallel simulate $\mathbb{I}^{\text{acc}}$ and $\mathbb{A}$ on y
2. **if** $\mathbb{I}^{\text{acc}}$ accepts, **then** accept
3. **if** $\mathbb{A}$ rejects, **then** reject.

Then for $y \in Q$,

$$t_{\mathbb{I}^{\text{acc}}}(y) \leq t_{\mathbb{I}^{\text{dec}}}(y) \leq O(t_{\mathbb{I}^{\text{acc}}}(y)).$$

These inequalities allow to translate most of our results concerning the acceptors $\mathbb{I}^{\text{acc}}$ into corresponding statements on the decision algorithms $\mathbb{I}^{\text{dec}}$. For example the statement corresponding to Proposition 3.7 would read:

> *Let $\mathbb{O}$ be an optimal inverter of $\mathbb{F}_{\text{SAT}}$. For every algorithm $\mathbb{B}$* deciding SAT *we have for all $\alpha \in$ SAT,*
>
> $$t_{\mathbb{O}^{\text{dec}}}(\alpha) \leq \big(|\alpha| \cdot \max\{t_{\mathbb{B}}(\alpha') \mid \alpha' \in \text{SAT } \textit{and } |\alpha'| \leq |\alpha|\}\big)^{O(1)}$$
>
> *(note that $\mathbb{O}^{\text{dec}}$ is an algorithm deciding* SAT*).*

4. Optimal acceptors

An optimal acceptor is an algorithm accepting a problem with optimal running time on the YES-instances (positive instances). In this section we study whether the algorithm $\mathbb{O}^{\text{acc}}$ accepting SAT and considered in Theorem 3.8 is an optimal acceptor. Furthermore, we present a problem decidable in exponential time (with a linear exponent), which has no optimal acceptor.

Definition 4.1. Let $Q \subseteq \Sigma^*$ be a problem.

– Let $\mathbb{A}$ and $\mathbb{B}$ be algorithms accepting Q. The algorithm $\mathbb{A}$ is *as fast as* $\mathbb{B}$ *on* YES*-instances*, written $\mathbb{A} \leq_{\text{YES}} \mathbb{B}$, if for every $x \in Q$,

$$t_{\mathbb{A}}(x) \leq (|x| + t_{\mathbb{B}}(x))^{O(1)}.$$

Note that nothing is required for $x \notin Q$.

– An algorithm $\mathbb{A}$ accepting Q is *optimal* if $\mathbb{A} \leq_{\text{YES}} \mathbb{B}$ for every algorithm $\mathbb{B}$ accepting Q. We then say that $\mathbb{A}$ is an *optimal acceptor* of Q.

We write $\mathbb{A} <_{\text{YES}} \mathbb{B}$ if $\mathbb{A} \leq_{\text{YES}} \mathbb{B}$ but $\mathbb{B} \not\leq_{\text{YES}} \mathbb{A}$.

Remark 4.2. The concept of optimality just defined was first considered in [16] for algorithms deciding the set TAUT of tautologies of propositional logic. The name "optimal acceptor" was introduced in [19]. Let Q be a decidable problem and $\mathbb{A}_0$ any algorithm deciding Q. If $\mathbb{A}$ is an optimal acceptor of Q, we get an algorithm deciding Q, which is still optimal, by running $\mathbb{A}_0$ and $\mathbb{A}$ in parallel in the obvious way. In connection with decision algorithms the optimality notion of Definition 4.1 has sometimes (e.g. in [4]) been called *almost optimality* in order to emphasize that it only refers to YES-instances.

Example 4.3. Let $\mathbb{F}$ be an algorithm with range Q. Then, we have $\mathbb{O}^{\text{acc}} \leq_{\text{YES}} \mathbb{I}^{\text{acc}}$ for every inverter $\mathbb{I}$ of $\mathbb{F}$ and every optimal inverter $\mathbb{O}$ of $\mathbb{F}$. In fact, for $y \in Q$,

$$\begin{array}{rcll} t_{\mathbb{O}^{\text{acc}}}(y) & \leq & O(t_{\mathbb{O}}(y) + t_{\mathbb{F}}(\mathbb{O}(y))) & \text{(by (3))} \\ & \leq & (t_{\mathbb{I}}(y) + t_{\mathbb{F}}(\mathbb{I}(y)))^{O(1)} & \text{(by (1))} \\ & \leq & (t_{\mathbb{I}^{\text{acc}}}(y))^{O(1)} & \text{(by (3)).} \end{array}$$

Hence, $\mathbb{O}^{\text{acc}}$ is an algorithm accepting Q "optimal in the class of all $\mathbb{I}^{\text{acc}}$."

Often we will apply the following simple observation.

Lemma 4.4. *If M is a subset of Q decidable in polynomial time, then every optimal acceptor of Q runs in polynomial time on M.*

Proof: Let $\mathbb{M}$ be an algorithm deciding M in polynomial time and $\mathbb{A}$ an optimal acceptor of Q. We define the algorithm $\mathbb{B}$, which accepts Q by running $\mathbb{M}$ and $\mathbb{A}$ in parallel as follows:

```
B   // x ∈ Σ*
  1. in parallel simulate M and A on x
  2.       if M accepts, then accept
  3.       if A accepts, then accept.
```

Clearly, $\mathbb{B}$ runs in polynomial time on M. By the optimality of $\mathbb{A}$ we know that $t_{\mathbb{A}}(x) \leq (|x| + t_{\mathbb{B}}(x))^{O(1)}$ holds for all $x \in Q$ (and thus for all $x \in M$). Therefore the algorithm $\mathbb{A}$ also runs in polynomial time on M. □

Every problem Q in P (polynomial time) has an optimal acceptor. Indeed every polynomial time algorithm deciding Q is an optimal acceptor of Q. As shown in [19] there are problems in E $\setminus$ P with optimal acceptors (where E := DTIME$(2^{O(n)})$). [3] To the best of our knowledge it is still not known whether there is a problem in NP $\setminus$ P having an optimal acceptor (even assuming P $\neq$ NP). In view of Theorem 3.8 one could expect that the algorithm $\mathbb{O}^{\text{acc}}$, where $\mathbb{O}$ is an optimal inverter of $\mathbb{F}_{\text{SAT}}$, is such an algorithm for the problem SAT. However, we can show:

Proposition 4.5. *Assume* NP $\cap$ coNP $\neq$ P. [4] *Then, for every optimal inverter $\mathbb{O}$ of $\mathbb{F}_{\text{SAT}}$ the algorithm $\mathbb{O}^{\text{acc}}$ is not an optimal acceptor of* SAT.

Proof: The proof uses some standard results and techniques from complexity theory. The result of the proposition will not be used again, so the reader not familiar with these techniques may skip this proof.

Let Q be a problem in $\big(\text{NP} \cap \text{coNP}\big) \setminus \text{P}$. Then there exist two polynomial time decidable relations R_1 and R_2 and two polynomials $p_1, p_2 \in \mathbb{N}[X]$ such that for every $x \in \Sigma^*$,

(i) $x \in Q$ if and only if there exists a $y \in \Sigma^*$ with $|y| \leq p_1(|x|)$ and $(x, y) \in R_1$;

(ii) $x \notin Q$ if and only if there exists a $y \in \Sigma^*$ with $|y| \leq p_2(|x|)$ and $(x, y) \in R_2$.

Using standard polynomial time reductions of the statements on the right hand sides of (i) and (ii) to SAT, we can compute, for $x \in \Sigma^*$, propositional formulas β_x and γ_x in polynomial time which express the right hand sides of (i) and (ii), respectively. Moreover,

$$\begin{array}{l} \text{from a satisfying assignment of } \delta_x := (\beta_x \vee \gamma_x) \text{ we obtain in polynomial time} \\ \text{a } y \text{ as required in (i) or (ii).} \end{array} \tag{7}$$

[3] Here, as usual, given a class F of total functions from $\mathbb{N}$ to $\mathbb{N}$ we denote by DTIME(F) the class of problems decidable by an algorithm $\mathbb{A}$ with $t_{\mathbb{A}} \in O(f)$ for some $f \in F$.

[4] The complexity class coNP consists of the complements of problems in NP.

By (i) and (ii), δ_x is satisfiable for all $x \in \Sigma^*$. Finally, we can assume that from δ_x we can recover x in polynomial time: For this purpose one uses a "fresh" propositional variable X and passes, say, for $x = 10011$, from δ_x to

$$(X \wedge \neg X \wedge \neg X \wedge X \wedge X \wedge \delta_x).$$

Thus,

$$M := \{\delta_x \mid x \in \Sigma^*\}$$

is a polynomial time decidable subset of SAT. Now let $\mathbb{O}$ be an optimal inverter of $\mathbb{F}_{\text{SAT}}$. By Lemma 4.4, $\mathbb{O}^{\text{acc}}$ must run in polynomial time on M if it is an optimal acceptor of SAT. We show that this is not the case. By (4) it suffices to show that $\mathbb{O}$ (instead of $\mathbb{O}^{\text{acc}}$) does not run in polynomial time on M. We consider the following algorithm:

$\mathbb{B}$ // $x \in \Sigma^*$

1. $\alpha \leftarrow \delta_x$
2. simulate $\mathbb{O}$ on α and let (α, S) be its output
3. compute from S the string y according to (7)
4. **if** $(x, y) \in R_1$ **then** accept **else** reject.

Assume that $\mathbb{O}$ runs in polynomial time on M. Then $\mathbb{B}$ decides Q in polynomial time, contradicting $Q \notin$ P. □

In the proof of Gödel's incompleteness theorems we will apply the following result.

Theorem 4.6 (Stockmeyer's Theorem [25])**.** *There is a decidable problem Q_0 in* $\text{E} := \text{DTIME}(2^{O(n)})$ *without optimal acceptor. Furthermore, there is a computable function S which assigns to every algorithm $\mathbb{B}$ accepting Q_0 an algorithm $S(\mathbb{B})$ also acepting Q_0 such that*

$$S(\mathbb{B}) <_{\text{YES}} \mathbb{B}.$$

Proof: For an algorithm $\mathbb{A}$ let $c_{\mathbb{A}}$ be a string in Σ^* coding $\mathbb{A}$. We set

$$Q_0 := \left\{c_{\mathbb{A}} \;\middle|\; \mathbb{A} \text{ an algorithm such that } \left(\mathbb{A} \text{ does not accept } c_{\mathbb{A}} \text{ or } t_{\mathbb{A}}(c_{\mathbb{A}}) > 2^{|c_{\mathbb{A}}|}\right)\right\}.$$

Claim 1. If $\mathbb{B}$ accepts Q_0, then $c_{\mathbb{B}} \in Q_0$ and $t_{\mathbb{B}}(c_{\mathbb{B}}) > 2^{|c_{\mathbb{B}}|}$.

Proof of Claim 1: Suppose that $c_{\mathbb{B}} \notin Q_0$. Then, $\mathbb{B}$ does not accept $c_{\mathbb{B}}$ (as $\mathbb{B}$ accepts Q_0) and thus, $c_{\mathbb{B}} \in Q_0$ by definition of Q_0. So we know that $c_{\mathbb{B}} \in Q_0$ and thus, $\mathbb{B}$ accepts $c_{\mathbb{B}}$. Hence, again by definition of Q_0, we have $t_{\mathbb{B}}(c_{\mathbb{B}}) > 2^{|c_{\mathbb{B}}|}$. ⊣

Let $\mathbb{B}$ accept Q_0. We show that $\mathbb{B}$ is not optimal. For $n \in \mathbb{N}$ we obtain the algorithm $\mathbb{B}_n$ by adding n useless instructions to $\mathbb{B}$ in a standard fashion such that

$$\text{for all } n \in \mathbb{N} \text{ and } x \in \Sigma^*, \quad \mathbb{B}(x) = \mathbb{B}_n(x) \quad \text{and} \quad t_{\mathbb{B}}(x) = t_{\mathbb{B}_n}(x),$$

and such that

$$C := \left\{c_{\mathbb{B}_n} \;\middle|\; n \in \mathbb{N}\right\} \in \text{P}.$$

As each $\mathbb{B}_n$ also accepts Q_0, by Claim 1 we have

$$C \subseteq Q_0 \quad \text{and} \quad t_{\mathbb{B}}(c_{\mathbb{B}_n}) = t_{\mathbb{B}_n}(c_{\mathbb{B}_n}) > 2^{|c_{\mathbb{B}_n}|} \text{ for all } n \in \mathbb{N}.$$

That is, $\mathbb{B}$ does not run in polynomial time on C and hence is not optimal by Lemma 4.4.

We let $S(\mathbb{B})$ be the following algorithm: on input x, it first checks whether $x \in C$; if so, it accepts; otherwise, it simulates $\mathbb{B}$ on input x and answers accordingly. Clearly, $S(\mathbb{B})$ accepts Q_0 and for all $x \in Q_0$ we have $t_{S(\mathbb{B})}(x) \leq (|x| + t_{\mathbb{B}}(x))^{O(1)}$, i.e., $S(\mathbb{B}) \leq_{\text{YES}} \mathbb{B}$. As $S(\mathbb{B})$ runs in polynomial time on C, we have $S(\mathbb{B}) <_{\text{YES}} \mathbb{B}$. □

5. Polynomially optimal proof systems

We recall the concept of polynomially optimal proof system and use Levin's optimal inverters to derive a relationship between these proof systems and optimal acceptors. In this section Q will always denote a *nonempty* subset of Σ^*.

Definition 5.1. (1) A *proof system* for Q is a polynomial time algorithm $\mathbb{P}$ computing a function with domain Σ^* (that is, a total function on Σ^*) and range Q. If $\mathbb{P}(x) = y$, we say that x is a $\mathbb{P}$-*proof* of y.

(2) A proof system $\mathbb{P}$ for Q is *polynomially optimal* or *p-optimal* if for every proof system $\mathbb{P}'$ for Q there is a polynomial time algorithm $\mathbb{T}$ that translates $\mathbb{P}'$-proofs into equivalent $\mathbb{P}$-proofs, i.e., for all $x' \in \Sigma^*$ we have

$$\mathbb{P}(\mathbb{T}(x')) = \mathbb{P}'(x').$$

For example, every standard complete deductive system D for first-order logic (for propositional logic) can be viewed as a proof system $\mathbb{P}_D$ for the set $Q :=$ VALID of valid first-order sentences (for the set $Q :=$ TAUT, of propositional tautologies):

$$\mathbb{P}_D(x) := \begin{cases} \varphi, & \text{if } x = (\varphi, d) \text{ and } d \text{ is a deduction in } D \text{ of the formula } \varphi \text{ of first-order logic} \\ \varphi_0, & \text{else,} \end{cases}$$

where φ_0 is a fixed valid formula of first-order logic (we leave the definition of the proof system for TAUT to the reader).

The following total extension $\mathbb{P}_{\text{SAT}}$ of the algorithm $\mathbb{F}_{\text{SAT}}$ (see (5)) is a proof system for SAT:

$$\mathbb{P}_{\text{SAT}}(x) := \begin{cases} \alpha, & \text{if } x = (\alpha, S) \text{ and the assignment } S \text{ satisfies the propositional formula } \alpha \\ \text{TRUE}, & \text{else.} \end{cases}$$

Every $Q \in$ P has a p-optimal proof system: Let $\mathbb{A}$ be a polynomial time algorithm deciding Q and fix $y_0 \in Q$. Then the following algorithm $\mathbb{P}$ is a p-optimal proof system for Q: on input x it simulates $\mathbb{A}$ on x, then outputs x if $\mathbb{A}$ accepts and outputs y_0 otherwise.

It is not hard to show that Q has a p-optimal proof system if it is polynomial time reducible to a problem Q' with a p-optimal proof system (see [14]). It is not known whether there are problems outside P with a p-optimal proof system. VALID has no p-optimal proof system (see [2]). For TAUT and SAT it is still open.

The following result shows that a problem Q has an optimal acceptor if it has a p-optimal proof system. This was first proved for $Q =$ TAUT in [16]. Hidden in that proof, as in the proof of the extension of this result to $Q :=$ SAT in [22], is a diagonal argument similar to the one used to obtain Levin's theorem. We apply this theorem directly; using its terminology, we get a more informative statement:

Theorem 5.2. *Let $Q \subseteq \Sigma^*$. Every optimal inverter of a p-optimal proof system for Q is an optimal acceptor of Q, more precisely:*

Let $\mathbb{P}$ be any p-optimal proof system for Q and let $\mathbb{O}$ be an optimal inverter of $\mathbb{P}$. Then $\mathbb{O}^{\text{acc}}$ is an optimal acceptor of Q (for the definition of $\mathbb{O}^{\text{acc}}$ see Proposition 3.6).

In particular, if Q has a p-optimal proof system, then Q has an optimal acceptor.

Proof: Let $\mathbb{P}$ be a p-optimal proof system for Q and $\mathbb{O}$ an optimal inverter of $\mathbb{P}$. We show that $\mathbb{O}^{\text{acc}}$ is an optimal acceptor of Q. So let $\mathbb{B}$ be any algorithm accepting Q. We have to show that $\mathbb{O}^{\text{acc}} \leq_{\text{YES}} \mathbb{B}$. The algorithm $\mathbb{B}$ induces a proof system $\mathbb{P}_{\mathbb{B}}$ for Q,

$$\mathbb{P}_{\mathbb{B}}(y, d) := \begin{cases} y, & \text{if } \mathbb{B} \text{ accepts } y \text{ by the computation } d \\ y_0, & \text{otherwise,} \end{cases}$$

where y_0 is a fixed element of Q. By the p-optimality of $\mathbb{P}$ there is a polynomial time algorithm $\mathbb{T}$ such that

$$\mathbb{P}(\mathbb{T}(y, d)) = \mathbb{P}_{\mathbb{B}}(y, d) = y$$

if $y \in Q$ and d is the computation of $\mathbb{B}$ on y. Therefore, we get an inverter $\mathbb{I}$ of $\mathbb{P}$ setting

$$\mathbb{I}(y) := \mathbb{T}(y, d), \quad \text{if } \mathbb{B} \text{ accepts } y \text{ and } d \text{ is the computation of } \mathbb{B} \text{ on } y.$$

The algorithm $\mathbb{I}$ does not halt for $y \notin Q$. For $y \in Q$ we get the computation d of $\mathbb{B}$ on y in time $O(t_{\mathbb{B}}(y))$; therefore,

$$t_{\mathbb{I}}(y) \leq O(t_{\mathbb{B}}(y)) + \big(|y| + |d|\big)^{O(1)} \leq \big(|y| + t_{\mathbb{B}}(y)\big)^{O(1)}.$$

As $\mathbb{O}$ is an optimal inverter of $\mathbb{P}$ and $\mathbb{P}$ runs in polynomial time, we get for $y \in Q$ from (2),

$$t_{\mathbb{O}}(y) \leq \big(t_{\mathbb{I}}(y)\big)^{O(1)} \leq \big(|y| + t_{\mathbb{B}}(y)\big)^{O(1)}.$$

Now (4) yields for $y \in Q$,

$$t_{\mathbb{O}^{\text{acc}}}(y) \leq \big(|y| + t_{\mathbb{B}}(y)\big)^{O(1)}.$$

Thus, $\mathbb{O}^{\text{acc}} \leq_{\text{YES}} \mathbb{B}$. □

For Q with a padding function,[5] the following converse of the previous theorem is also known: If $Q \neq \emptyset$ has an optimal acceptor, then Q has a p-optimal proof system (see [19, 23]). It is not known whether the result still holds for Q without padding.

6. Hard sequences for algorithms and proof systems

Stockmeyer's Theorem presents a problem Q_0 without optimal acceptor. The basic idea behind the proof is to exhibit for every algorithm $\mathbb{B}$ accepting Q_0 a polynomial time definable sequence $(c_{\mathbb{B}_s})_{s\in\mathbb{N}}$ of elements of Q_0 such that $\mathbb{B}$ does not run in polynomial time on it; this allows to superpolynomially speed up $\mathbb{B}$ on $\{c_{\mathbb{B}_s} \mid s \in \mathbb{N}\}$. Such hard sequences for algorithms and proof systems have turned out to be a useful tool in the study of the existence of optimal acceptors and p-optimal proof systems [15, 6]. Here, applying Levin's optimal inverters, we show how hard sequences for acceptors of a given problem Q translate into hard sequences for proof systems for Q.

We start with a precise definition of the notion of hard sequence:

Definition 6.1. Let $Q \subseteq \Sigma^*$ be recursively enumerable.

1. Let $\mathbb{A}$ be an algorithm accepting Q. A sequence $(x_s)_{s\in\mathbb{N}}$ is *hard for* $\mathbb{A}$ if
   - $\{x_s \mid s \in \mathbb{N}\} \subseteq Q$;
   - the function $1^s \mapsto x_s$ is computable in polynomial time (here, $1^s = \overbrace{11\ldots1}^{s}$);
   - $t_{\mathbb{A}}(x_s)$ is not polynomially bounded in s (that is, for no polynomial $p \in \mathbb{N}[X]$ we have $t_{\mathbb{A}}(x_s) \leq p(s)$ for all $s \in \mathbb{N}$).
2. The problem Q *has hard sequences for acceptors* if every acceptor of Q has a hard sequence.
3. Let $\mathbb{P}$ be a proof systems for Q. A sequence $(x_s)_{s\in\mathbb{N}}$ is *hard for* $\mathbb{P}$ if
   - $\{x_s \mid s \in \mathbb{N}\} \subseteq Q$;
   - the function $1^s \mapsto x_s$ is computable in polynomial time;
   - there is no polynomial time algorithm $\mathbb{W}$ with $\mathbb{P}(\mathbb{W}(1^s)) = x_s$ for all $s \in \mathbb{N}$.
4. The problem Q *has hard sequences for proof systems* if every proof system for Q has a hard sequence.

One easily verifies that (see [6]):

(a) No acceptor with a hard sequence is optimal, and similarly no proof system with a hard sequence is p-optimal.

[5] We do not introduce the notion of padding function as we are not going to use it. We refer the interested reader to [21, Definition 14.2].

(b) If Q is polynomial time reducible to a problem Q' and has hard sequences for acceptors, then so does Q'.

We have seen that the problem Q_0 of Stockmeyer's Theorem has hard sequences for acceptors (as already mentioned, for every algorithm $\mathbb{B}$ accepting Q_0 the sequence $(c_{\mathbb{B}_s})_{s\in\mathbb{N}}$ defined in the proof of that theorem is a hard sequence for $\mathbb{B}$). As $Q_0 \in \mathrm{E}$ (see Stockmeyer's Theorem), we have $Q_0 \in \mathrm{EXP} = \mathrm{DTIME}(2^{n^{O(1)}})$ (exponential time).[6] Thus, by (a) and (b):

Corollary 6.2. *If Q is* EXP*-hard under polynomial reductions, then Q has no optimal acceptors.*

We turn to the result announced in the first paragraph of this section.

Theorem 6.3 ([6])**.** *Let $Q \subseteq \Sigma^*$ and let $\mathbb{P}$ be a proof system for Q. Then, for every optimal inverter $\mathbb{O}$ of $\mathbb{P}$, every hard sequence for $\mathbb{O}^{\mathrm{acc}}$ is a hard sequence for $\mathbb{P}$.*

In particular, if Q has hard sequences for algorithms, then Q has hard sequences for proof systems.

Proof: Let $\mathbb{O}$ be an optimal inverter of $\mathbb{P}$ and let $(x_s)_{s\in\mathbb{N}}$ be a hard sequence for $\mathbb{O}^{\mathrm{acc}}$. We show that $(x_s)_{s\in\mathbb{N}}$ is a hard sequence for $\mathbb{P}$.

By the hardness of $(x_s)_{s\in\mathbb{N}}$ for $\mathbb{O}^{\mathrm{acc}}$ there is a polynomial time algorithm $\mathbb{G}$ that on input 1^s outputs $x_s \in Q$ such that

$$t_{\mathbb{O}^{\mathrm{acc}}}(x_s) \text{ is not polynomially bounded in } s. \tag{8}$$

Let $\mathbb{G}'$ be the following algorithm that halts on inputs $x \in \{x_s \mid s \in \mathbb{N}\}$ and then outputs 1^s for the (least) s with $x = x_s$:

```
G'   // x ∈ Σ*
  1. ℓ ← 1
  2. for s = 1 to ℓ
  3.       simulate the (ℓ − (s − 1))th step of G on 1^s
  4.       if this simulation outputs y and y = x, then halt with output 1^s
  5. ℓ ← ℓ + 1
  6. goto 2.
```

As $\mathbb{G}$ is a polynomial time algorithm, we have:

$$t_{\mathbb{G}'}(x_s) \text{ is polynomial in } s. \tag{9}$$

Suppose that $(x_s)_{s\in\mathbb{N}}$ is not a hard sequence for $\mathbb{P}$. Then there is a polynomial time algorithm $\mathbb{W}$ with

$$\mathbb{P}(\mathbb{W}(1^s)) = x_s \tag{10}$$

for all $s \in \mathbb{N}$. We consider the following algorithm $\mathbb{I}$:

```
I   // x ∈ Σ*
  1. in parallel simulate O and G' on x
  2.       if O halts, then halt with output O(x)
  3.       if G' halts, then simulate W on G'(x) and halt with output W(G'(x)).
```

This algorithm is an inverter of $\mathbb{P}$: If $\mathbb{O}$ halts first, then $x \in Q$ and $\mathbb{P}(\mathbb{I}(x)) = \mathbb{P}(\mathbb{O}(x)) = x$ (by Theorem 3.2); if $\mathbb{G}'$ halts first, then $\mathbb{G}'(x) = 1^s$ for some s with $x = x_s$. Thus $\mathbb{P}(\mathbb{I}(x)) = \mathbb{P}(\mathbb{W}(\mathbb{G}'(x))) = \mathbb{P}(\mathbb{W}(1^s)) = x_s = x$ by (10).

[6] It is easy to show that Q_0 is even complete for EXP under polynomial reductions.

By (9) and as $\mathbb{W}$ runs in polynomial time, $t_{\mathbb{I}}(x_s)$ is polynomial in s. Therefore, $t_{\mathbb{O}}(x_s)$ is polynomial in s, too (by (2) as $\mathbb{P}$ is a polynomial time algorithm). But then, by (4), the same holds for the acceptor $\mathbb{O}^{\text{acc}}$, i.e., $t_{\mathbb{O}^{\text{acc}}}(x_s)$ is polynomial in s; this contradicts (8). □

It is not known whether the following converse of the preceding theorem holds: If Q has hard sequences for proof systems, then Q has hard sequences for acceptors.

By a result of [6] we know that the equivalence

$$Q \text{ has hard sequences for acceptors} \iff Q \text{ has no optimal acceptor} \tag{11}$$

holds for every problem Q complete for one of the classes Π_t^p with $t \geq 1$ (the tth class of the polynomial hierarchy) or complete for the class EXP.

In particular, the equivalence holds for the coNP-complete problem TAUT (recall that coNP $= \Pi_1^p$). There are some limitations when trying to derive (11) for all decidable problems Q, since it was shown in [6]:

> *If the Measure Hypothesis holds, then there is a decidable problem which has no optimal acceptor but is accepted by an algorithm without hard sequences.*

The *Measure Hypothesis* [11], a hypothesis sometimes used in the theory of resource bounded measures, is the assumption "NP does not have measure 0 in E." For the corresponding notion of measure we refer the reader to [18].

7. Gödel's incompleteness theorems

Turing was the first to realize that a proof of at least Gödel's First Incompleteness Theorem can be obtained in terms of computability theory. In his seminal paper [26], referring to Gödel's publication [9], he writes: "By the correct application of one of these arguments, conclusions are reached which are superficially similar to those of Gödel." In [26], Turing showed that the halting problem for Turing machines is not decidable and hence, the set

$$\text{FOREVER} := \big\{\mathbb{M} \;\big|\; \mathbb{M} \text{ is a Turing machine that, with the empty string as input, runs forever}\big\}$$

is not recursively enumerable. Using this result one easily gets Gödel's First Incompleteness Theorem as follows:

Let T be a decidable, true,[7] and sufficiently strong first-order theory so that for every Turing machine $\mathbb{M}$ we can formalize the statement

$$\text{the Turing machine } \mathbb{M}\text{, with the empty string as input, runs forever.}$$

Assume that $T \vdash$ "$\mathbb{M}$ with the empty string as input runs forever" (that is, T proves the formalization of the statement '$\mathbb{M}$ with the empty string as input runs forever'). Since T is a true theory, then $\mathbb{M}$ with the empty string as input runs forever. Thus,

$$\big\{\mathbb{M} \;\big|\; T \vdash \text{"}\mathbb{M} \text{ with the empty string as input runs forever"}\big\} \subseteq \text{FOREVER}.$$

As the set on the left hand side but not FOREVER is recursively enumerable, the sentence "$\mathbb{M}$ with the empty string as input runs forever" is true but not provable in T for some Turing machine $\mathbb{M}$.

In our approach the true but not provable statements have the form

$$\text{the algorithm } \mathbb{O}^{\text{acc}} \text{ accepts the problem } Q, \tag{12}$$

where the algorithm $\mathbb{O}$ is any optimal inverter of some algorithm (depending on T) with range Q, a recursively enumerable subset of Σ^*. Recall that the algorithm $\mathbb{O}^{\text{acc}}$ was defined in Proposition 3.6. Furthermore,

[7] By "T is true" we mean that T consists of formalizations of statements true in the metatheory. We assume the consistency of the metatheory.

for these optimal inverters $\mathbb{O}$ the provability of the statement in (12) is even equivalent to the provability of the consistency of T (see (16) in Theorem 7.2, where it is assumed that Q is not decidable in polynomial time).

Already Hutter [12] considered 'provable' algorithms, where 'provable' refers to a recursively enumerable, more or less specified true theory T. He constructed an algorithm "which is the fastest and the shortest" deciding a given problem. As Hutter said, Peter van Emde Boas pointed out to him that it is not provable that his algorithm decides the given problem and that his proof is a "meta-proof which cannot be formalized within the considered proof system." He added that "a formal proof of its correctness would prove the consistency of the proof system, which is impossible by Gödel's Second Incompleteness Theorem."

We turn to our proofs. Let us fix:

- a recursively enumerable subset Q of Σ^*;
- an effective enumeration $\mathbb{A}_1, \mathbb{A}_2, \ldots$ of all algorithms;
- a decidable, true, and sufficiently strong first-order theory T.

In first-order logic we can formalize the statement

$$\mathbb{A}_i \text{ accepts } Q.$$

We denote the formalization by "$\mathbb{A}_i$ accepts Q." Moreover we can ascertain that a string π is a proof of this formalization from T, written

$$\pi : T \vdash \text{"}\mathbb{A}_i \text{ accepts } Q\text{."}$$

As T is a true theory, we know that $\mathbb{A}_i$ accepts Q if $\pi : T \vdash$ "$\mathbb{A}_i$ accepts Q" for some π. We assume that there is an $i_0 \geq 1$ and a π_0 such that

$$\pi_0 : T \vdash \text{"}\mathbb{A}_{i_0} \text{ accepts } Q\text{."} \tag{13}$$

We consider the algorithm $\mathbb{F}_{T,Q}$ with

$$\mathbb{F}_{T,Q}(i,\pi,x) := x, \qquad \text{if } \pi : T \vdash \text{"}\mathbb{A}_i \text{ accepts } Q\text{" and } \mathbb{A}_i \text{ accepts } x. \tag{14}$$

On other inputs the algorithm $\mathbb{F}_{T,Q}$ does not halt. On inputs (i,π,x) as above, we have

$$t_{\mathbb{F}_{T,Q}}(i,\pi,x) \leq f(i) \cdot |\pi| \cdot t_{\mathbb{A}_i}(x) \tag{15}$$

for some computable function $f : \mathbb{N} \to \mathbb{N}$. By (13), the range of $\mathbb{F}_{T,Q}$ is Q.

By the previous remarks, the following claims is immediate.

Claim 1. Let $j \geq 1$ and π be such that $\pi : T \vdash$ "$\mathbb{A}_j$ accepts Q." Then the algorithm $\mathbb{I}_{j,\pi}$ with

$$\mathbb{I}_{j,\pi}(x) := (j,\pi,x)$$

for $x \in \Sigma^*$ is an inverter of $\mathbb{F}_{T,Q}$ with $t_{\mathbb{I}_{j,\pi}}(x) \leq O(|x|)$. Furthermore, for all $x \in Q$,

$$t_{\mathbb{F}_{T,Q}}(\mathbb{I}_{j,\pi}(x)) = t_{\mathbb{F}_{T,Q}}(j,\pi,x) \leq O(t_{\mathbb{A}_j}(x))$$

(the inequality holds by (15)).

For every optimal inverter $\mathbb{O}$ of $\mathbb{F}_{T,Q}$ we show, using Claim 1, that the algorithm $\mathbb{O}^{\text{acc}}$ accepts Q as fast as any $\mathbb{A}_j$ accepting Q provably in T. More precisely:

Claim 2. Let $\mathbb{O}$ be an optimal inverter of $\mathbb{F}_{T,Q}$. Assume $j \geq 1$ and let π be such that $\pi : T \vdash$ "$\mathbb{A}_j$ accepts Q." Then $\mathbb{O}^{\text{acc}} \leq_{\text{YES}} \mathbb{A}_j$.

Proof: Let $\mathbb{O}$, $j \geq 1$, and π be as in the statement of the claim. For $x \in Q$ we have:

$$\begin{array}{lll} t_{\mathbb{O}^{\text{acc}}}(x) & \leq & O\big(t_{\mathbb{O}}(x) + t_{\mathbb{F}_{T,Q}}(\mathbb{O}(x))\big) \qquad \text{(by (3))} \\ & \leq & \big(t_{\mathbb{I}_{j,\pi}}(x) + t_{\mathbb{F}_{T,Q}}(\mathbb{I}_{j,\pi}(x))\big)^{O(1)} \qquad \text{(Claim 1 and (1))} \\ & \leq & \big(|x| + t_{\mathbb{A}_j}(x)\big)^{O(1)} \qquad \text{(Claim 1).} \end{array}$$

⊣

Theorem 7.1 (Gödel's First Incompleteness Theorem)**.** *For every decidable, true, and sufficiently strong first-order theory T there exists a true sentence φ such that $T \nvdash \varphi$.*

Proof: Let the problem Q_0 and the function S have the properties stated in Stockmeyer's Theorem (Theorem 4.6). For an optimal inverter $\mathbb{O}$ of $\mathbb{F}_{T,Q_0}$ we know that $\mathbb{O}^{\text{acc}}$ accepts Q_0 (by Proposition 3.6) and that $S(\mathbb{O}^{\text{acc}}) <_{\text{YES}} \mathbb{O}^{\text{acc}}$ (by Stockmeyer's Theorem). But $S(\mathbb{O}^{\text{acc}})$ is one of the algorithms of the enumeration $\mathbb{A}_1, \mathbb{A}_2, \ldots$, say, $\mathbb{A}_j$. Thus, $\mathbb{A}_j <_{\text{YES}} \mathbb{O}^{\text{acc}}$. Hence, $T \nvdash$ "$\mathbb{A}_j$ accepts Q_0" by Claim 2. □

As we gave an explicit definition of Q_0 and S in Theorem 4.6, we can construct a true sentence φ such that $T \nvdash \varphi$ explicitly.

We see that a decidable problem may be solvable by an algorithm whose proof of correctness needs tools not available in the given theory T. Moreover, stronger theories may know of faster algorithms solving the problem. In a discussion with the authors of [5], Sy-David Friedman posed the question whether $T \cup \{Con_T\}$ can be characterized as a minimal extension of T in this complexity-theoretic context (here Con_T denotes a sentence formalizing the consistency of T in a standard way). Theorem 7.2 contains such a characterization.

Theorem 7.2 ([5])**.** *Assume that T is a decidable, true, and sufficiently strong first-order theory. Let Q be a decidable problem, which is not decidable in polynomial time. Furthermore let $\mathbb{O}$ be an optimal inverter of the algorithm $\mathbb{F}_{T,Q}$ (cf.* (14)*). Then, for every theory $T' \supseteq T$ we have*

$$T' \vdash \text{“}\mathbb{O}^{\text{acc}} \textit{ accepts } Q\text{”} \iff T' \vdash Con_T.$$

In particular,

$$T \vdash \text{“}\mathbb{O}^{\text{acc}} \textit{ accepts } Q\text{”} \iff T \vdash Con_T. \tag{16}$$

We use this result to show:

Theorem 7.3 (Gödel's Second Incompleteness Theorem)**.** *Every decidable, true, and sufficiently strong first-order theory T does not prove its own consistency, i.e., $T \nvdash Con_T$.*

Proof: Again we consider the problem Q_0 and the function S defined in the proof of Stockmeyer's Theorem. By (16) we know that for an optimal inverter $\mathbb{O}$ of $\mathbb{F}_{T,Q_0}$,

$$T \vdash \text{“}\mathbb{O}^{\text{acc}} \text{ accepts } Q_0\text{”} \iff T \vdash Con_T.$$

Thus, it suffices to show that $T \nvdash$ "$\mathbb{O}^{\text{acc}}$ accepts Q_0." Suppose, for a contradiction, that

$$T \vdash \text{“}\mathbb{O}^{\text{acc}} \text{ accepts } Q_0\text{.”} \tag{17}$$

For the algorithm $S(\mathbb{O}^{\text{acc}})$ we know, by Stockmeyer's Theorem, that

$$S(\mathbb{O}^{\text{acc}}) <_{\text{YES}} \mathbb{O}^{\text{acc}}. \tag{18}$$

On the other hand, as we assumed T to be sufficiently strong, the simple part of the proof of Stockmeyer's Theorem showing that $S(\mathbb{O}^{\text{acc}})$ accepts Q_0 (as $\mathbb{O}^{\text{acc}}$ accepts Q_0) can be carried out in T. Hence, by (17),

$$T \vdash \text{“}S(\mathbb{O}^{\text{acc}}) \text{ accepts } Q_0\text{.”}$$

But then $\mathbb{O}^{\text{acc}} \leq_{\text{YES}} S(\mathbb{O}^{\text{acc}})$ by Claim 2, which contradicts (18). □

8. The Exponential Time Hypothesis and the clique problem

The Exponential Time Hypothesis (ETH) is a computational hardness assumption. It states that the problem 3SAT cannot be solved in subexponential time (see [13]). In [8], under the assumption (ETH) it has been shown that the parameterized clique problem is not uniformly fixed-parameter tractable; thus (ETH) implies

that FPT $\neq$ W[1] (for the classes FPT and W[1] of parameterized complexity).[8] In [1] the nonuniform Exponential Time Hypothesis (ETH_{nu}) was considered and similar results were proven in the world of nonuniform parameterized complexity. Using an optimal inverter we derive one of the results of [1]; by the way, we originally obtained this result using inverters.

By CLIQUE we denote the problem

CLIQUE	
Instance:	A graph $G = \big(V(G), E(G)\big)$ with vertex set $V(G)$ and edge set $E(G)$, and $k \in \mathbb{N}$.
Question:	Is there a k-clique in G, i.e., is there a $C \subseteq V(G)$ with $\lvert C\rvert = k$ such that all distinct $u, v \in C$ are adjacent in G?

The problem CLIQUE is NP-complete, thus unlikely to be solvable in polynomial time. The algorithm, which decides CLIQUE by systematically checking, on input (G, k), all subsets of vertices of G of size k, has running time $O(2^{|V(G)|})$. The best known algorithm for CLIQUE has running time

$$O\left(2^{\varepsilon \cdot |V(G)|}\right)$$

for some ε with $0 < \varepsilon < 1$.

By a result derived in [13] the following definition of the Exponential Time Hypothesis (ETH) in terms of CLIQUE is equivalent to the original definition.

Definition 8.1. (1) The nonuniform Exponential Time Hypothesis (ETH_{nu}) is the statement

$$\text{CLIQUE} \notin \bigcap_{\varepsilon > 0} \text{DTIME}(2^{\varepsilon \cdot |V(G)|}).$$

(2) The Exponential Time Hypothesis (ETH) is the statement

$$\text{CLIQUE} \notin \text{DTIME}(2^{o(|V(G)|)}).$$ [9]

We believe that both, (ETH_{nu}) and (ETH), are true. However, it is known [1] that the underlying complexity classes are distinct. More precisely, the strict inclusion $\text{DTIME}(2^{o(n)}) \subset \bigcap_{\varepsilon > 0} \text{DTIME}(2^{\varepsilon \cdot n})$ holds. Clearly, (ETH_{nu}) implies (ETH). Using optimal inverters, we prove a partial converse:

Theorem 8.2 ([1])**.** *The following statements are equivalent:*

(i) (ETH_{nu}) *fails, i.e.,* $\text{CLIQUE} \in \bigcap_{\varepsilon > 0} \text{DTIME}(2^{\varepsilon \cdot |V(G)|})$.

(ii) There is an algorithm $\mathbb{A}$ *accepting* CLIQUE *with*

$$t_{\mathbb{A}}(G, k) \leq 2^{o(|V(G)|)}$$

for every YES*-instance* (G, k) *of* CLIQUE*, i.e., for every* $(G, k) \in \text{CLIQUE}$*.*

Proof: The implication (ii) $\Rightarrow$ (i) is easy: Let $\mathbb{A}$ be an algorithm as in (ii). Furthermore, let $\varepsilon > 0$. We present an algorithm $\mathbb{B}$ that witnesses $\text{CLIQUE} \in \text{DTIME}(2^{\varepsilon \cdot |V(G)|})$. By (ii), there is an $n_0 \in \mathbb{N}$ such that for YES-instances (G, k) of CLIQUE with $|V(G)| > n_0$ we have $t_{\mathbb{A}}(G, k) \leq 2^{\varepsilon \cdot |V(G)|}$. The algorithm $\mathbb{B}$, for instances (G, k) of CLIQUE with $|V(G)| > n_0$, simulates $\mathbb{A}$ for at most $2^{\varepsilon \cdot |V(G)|}$ steps and rejects if $\mathbb{A}$ does not accept within this time bound. For smaller graphs, the algorithm $\mathbb{B}$ checks all sets of vertices of size k and answers accordingly.

[8] We do not repeat the definitions of these complexity classes as we do not use them here.

[9] Recall that $f \in o(g)$ for functions $f, g : \mathbb{N} \to \mathbb{N}$ if there are $c, k \in \mathbb{N}$ such that $f(n) \leq \frac{1}{c} \cdot g(n)$ for all $n \in \text{dom}(f)$ with $n \geq k$. We use similar notations as in the context of the big-O notation.

(i) $\Rightarrow$ (ii): Let $\mathbb{F}$ be the algorithm with

$$\mathbb{F}(x) := (G, k), \quad \text{if } x = (G, C) \text{ and } C \text{ is a } k\text{-clique of the graph } G. \tag{19}$$

For other inputs $\mathbb{F}$ does not halt. Clearly, $\mathbb{F}$ runs in linear time on its domain and $\text{rng}(\mathbb{F}) = \text{CLIQUE}$, that is, the range of $\mathbb{F}$ is the class of YES-instances of the problem CLIQUE. Let $\mathbb{O}$ be an optimal inverter of $\mathbb{F}$ and $\mathbb{O}^{\text{acc}}$ the acceptor of CLIQUE defined in Proposition 3.6. Assuming (i) we want to show that

$$t_{\mathbb{O}^{\text{acc}}}(G, k) \leq 2^{o(|V(G)|)}$$

for YES-instances (G, k). Thus, for every $\iota > 0$ we have to show that

$$t_{\mathbb{O}^{\text{acc}}}(G, k) \leq 2^{\iota \cdot |V(G)|}$$

for sufficiently large graphs G with a k-clique.

Let $\varepsilon > 0$ (later we will fix the value of ε). By (i), there is an algorithm $\mathbb{A}_\varepsilon$ deciding CLIQUE in time $O(2^{\varepsilon \cdot |V(G)|})$. The following algorithm $\mathbb{I}_\varepsilon$ is an inverter of $\mathbb{F}$:

$\mathbb{I}_\varepsilon$ // $G = (V(G), E(G))$ a graph and $k \in \mathbb{N}$

1. simulate $\mathbb{A}_\varepsilon$ on (G, k)
2. **if** the simulation rejects **then** reject
3. $S \leftarrow V(G)$
4. **for** $i = 1$ **to** $|V(G)|$ **do**
5. $\quad v \leftarrow$ the ith vertex of $V(G)$
6. $\quad$ simulate $\mathbb{A}_\varepsilon$ on $G[S \setminus \{v\}]$ [10]
7. $\quad$ **if** the simulation accepts **then** $S \leftarrow S \setminus \{v\}$
8. output (G, S).

For every vertex of the input graph there is at most one simulation of $\mathbb{A}_\varepsilon$ (Line 6). Thus, if G has a k-clique, i.e., if $(G, k) \in \text{CLIQUE} = \text{rng}(\mathbb{F})$, then

$$t_{\mathbb{I}_\varepsilon}(G, k) \leq O\left(|V(G)| \cdot 2^{\varepsilon \cdot |V(G)|}\right). \tag{20}$$

Therefore, for some $d \in \mathbb{N}$ we have for all $(G, k) \in \text{rng}(\mathbb{F})$,

$$\begin{aligned} t_{\mathbb{O}}(G, k) &\leq \left(t_{\mathbb{I}_\varepsilon}(G, k) + t_{\mathbb{F}}(\mathbb{I}_\varepsilon(G, k))\right)^d && \text{(by (1))} \\ &\leq O\Big(\left(|V(G)| \cdot 2^{\varepsilon \cdot |V(G)|} + |V(G)| \cdot 2^{\varepsilon \cdot |V(G)|}\right)^d\Big) && \text{(by (20) and as } \mathbb{F} \\ & && \text{runs in linear time)} \\ &\leq O\left(|V(G)|^d \cdot 2^{\varepsilon \cdot d \cdot |V(G)|}\right). \end{aligned}$$

The algorithm $\mathbb{F}$ runs in linear time, hence, by (3),

$$t_{\mathbb{O}^{\text{acc}}}(G, k) \leq O\left(|V(G)|^d \cdot 2^{\varepsilon \cdot d \cdot |V(G)|}\right)$$

holds for $(G, k) \in \text{rng}(\mathbb{F})$. Thus, for $\varepsilon := \iota/d - 1$, we have

$$t_{\mathbb{O}^{\text{acc}}}(G, k) \leq 2^{\iota \cdot |V(G)|}$$

for sufficiently large YES-instances (G, k) of CLIQUE. □

We already mentioned the following result:

[10] As usual, for a subset M of the vertex set $V(G)$ of a graph G we denote by $G[M]$ the subgraph of G induced on M.

Theorem 8.3 ([8])**.** *Assume* (ETH)*. Then there is no algorithm deciding* CLIQUE *in time*

$$f(k) \cdot |G|^{O(1)},$$

where $f : \mathbb{N} \to \mathbb{N}$ is an arbitrary function. Equivalently, the parameterized clique problem (parameterized by k) is not uniformly fixed-parameter tractable.

Note that this result does not rule out that for some fixed $d \in \mathbb{N}$ every slice of the problem CLIQUE is in $\text{DTIME}(n^d)$; that is, that for every fixed k there is an algorithm deciding whether a graph G has a k-clique in time $O(|G|^d)$. If (ETH_{nu}) holds, this cannot be the case. Indeed using Levin's result one gets:

Theorem 8.4 ([1])**.** *Assume (*ETH_{nu}*). Then for every $d \in \mathbb{N}$ there is a $k_0 \in \mathbb{N}$ such that for all $k \geq k_0$ the problem to decide whether a graph has a k-clique is not in* $\text{DTIME}(n^d)$*.*

9. A space version of Levin's result

Recently [3] we proved a space version of Levin's result and put it to good use [3, 4]. For example, given a problem Q we introduced the notion of a space optimal proof system for Q and relate it to space optimal acceptors of Q. Among others, we obtained results, which correspond to those in Section 5.

Here we just mention this space version. More or less, it can be proved along the lines of Theorem 3.2. For an algorithm $\mathbb{A}$ and a string x we denote by $s_{\mathbb{A}}(x)$ the space required by $\mathbb{A}$ on input x; if it is unbounded, we set $s_{\mathbb{O}}(y) = \infty$.

Theorem 9.1. *Let $\mathbb{F}$ be an algorithm computing a (partial) function from Σ^* to Σ^*. Then there is a* space-optimal inverter*, that is, an inverter $\mathbb{O}$ of $\mathbb{F}$ such that:*

- *For every inverter $\mathbb{I}$ of $\mathbb{F}$ and all $y \in \text{rng}(\mathbb{F})$ we have*

$$s_{\mathbb{O}}(y) \leq \big(s_{\mathbb{I}}(y) + \log |\mathbb{I}(y)| + s_{\mathbb{F}}(\mathbb{I}(y))\big)^{O(1)}.$$

- *$s_{\mathbb{O}}(y) = \infty$ for $y \notin \text{rng}(\mathbb{F})$ (in particular, $\mathbb{O}$ does not stop on such inputs).*

References

[1] Y. Chen, K. Eickmeyer, and J. Flum. The exponential time hypothesis and the parameterized clique problem. In *Proceedings of the 7th International Symposium on Parameterized and Exact Computation (IPEC'12)*, Lecture Notes in Computer Science, pages 13–24. Springer, 2012.

[2] Y. Chen and J. Flum. On p-optimal proof systems and logics for PTIME. In *Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP'10, Track B)*, Lecture Notes in Computer Science 6199, pages 321–332. Springer, 2010.

[3] Y. Chen and J. Flum. Listings and logics. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science (LICS'11)*, pages 165–174. IEEE Computer Society, 2011.

[4] Y. Chen and J. Flum. From almost optimal algorithms to logics for complexity classes via listings and a halting problem. *Journal of the ACM*, 59(4):17, 2012.

[5] Y. Chen, J. Flum, and M. Müller. Consistency and optimality. In *Proceedings of the 7th Computability in Europe, Mathematical Theory and Computational Practice (CiE'11)*, volume 6735 of *Lecture Notes in Computer Science*, pages 61–70, 2011.

[6] Y. Chen, J. Flum, and M. Müller. Hard instances of algorithms and proof systems. In *Proceedings of How the World Computes - Turing Centenary Conference and the 8th Computability in Europe, Mathematical Theory and Computational Practice (CiE'12)*, volume 7318 of *Lecture Notes in Computer Science*, pages 118–128, 2012.

[7] N. Christennsen. *Levin's Optimal Search Theorem and Blum's Speedup Theorem*. Master Thesis, University of Copenhagen, 1999.

[8] R. Downey and M. Fellows. Fixed-parameter tractability and completeness III: Some structural aspects of the W-hierarchy. In *Complexity Theory*, pages 166–191. Cambridge University Press, 1993.

[9] K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931.

[10] Y. Gurevich. On Kolmogorov machines and related issues. *Bulletin of the European Association for Theoretical Computer Science*, 35:71–81, 1988.

[11] J. M. Hitchcock and A. Pavan. Hardness hypotheses, derandomization, and circuit complexity. In *Proceedings of the 24th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'04)*, pages 336–347, 2004.

[12] M. Hutter. The fastest and shortest algorithm for all well-defined problems. *International Journal of Foundations of Computer Science*, 13(3):431–443, 2002.

[13] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63:512–530, 2001.

[14] J. Köbler and J. Messner. Complete problems for promise classes by optimal proof systems for test sets. In *Proceedings of the 13th Annual IEEE Conference on Computational Complexity (CCC'98)*, pages 132–140. Springer, 1998.

[15] J. Krajíček. *Bounded arithmetic, propositional logic, and complexit theory*. Cambridge University Press, 1995.

[16] J. Krajíček and P. Pudlák. Propositional proof systems, the consistency of first order theories and the complexity of computations. *The Journal of Symbolic Logic*, 54(3):1063–1079, 1989.

[17] L. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1973.

[18] E. Mayordomo. Almost every set in exponential time is P-bi-immune. *Theoretical Computer Science*, 136(2):487–506, 1994.

[19] J. Messner. On optimal algorithms and optimal proof systems. In *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science, (STACS'99)*, Lecture Notes in Computer Science, pages 541–550. Springer, 1999.

[20] J. Messner. *On the simulation order of proof systems*. PhD thesis, University of Erlangen, 2000.

[21] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[22] Z. Sadowski. On an optimal deterministic algorithm for SAT. In *Proceedings of Computer Science Logic 1998 (CSL 98)*, Lecture Notes in Computer Science 1584, pages 179–187. Springer, 1998.

[23] Z. Sadowski. On an optimal propositional proof system and the structure of easy subsets. *Theoretical Computer Science*, 288(1):181–193, 2002.

[24] Z. Sadowski. Optimal proof systems, optimal acceptors and recursive presentability. *Fundamenta Informaticae*, 79(1-2):169–185, 2007.

[25] L. Stockmeyer. *The complexity of decision problems in automata theory*. PhD thesis, MIT, 1974.

[26] A.M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2:230–265, 1936.

[27] O. V. Verbitsky. Optimal algorithms for coNP-sets and the problem EXP=NEXP. *Matematicheskie zametki*, 50(2):37–46, 1979.